FORMAL VERIFICATION OF DECENTRALIZED COORDINATION IN AUTONOMOUS MULTI-AGENT AEROSPACE SYSTEMS

Saswata Paul
GE Research
19 PUBLICATIONS **82** CITATIONS

FORMAL VERIFICATION OF DECENTRALIZED COORDINATION IN AUTONOMOUS MULTI-AGENT AEROSPACE SYSTEMS

Saswata Paul

Submitted in Partial Fullfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

Approved by:
Dr. Carlos A. Varela, Chair
Dr. Stacy Patterson
Dr. Selmer Bringsjord
Dr. César A. Muñoz

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York

[May 2022]
Submitted April 2022

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

2D: Two Dimensional

3D: Three Dimensional

AM: Varela's Dialect of Agha, Mason, Smith, & Talcott's Actor Language

AMST: Agha, Mason, Smith, & Talcott's Actor Language

ATC: Air Traffic Control

ATM: Air Traffic Management

ATP: Automated Theorem Prover

CCT: Code Carrying Theory

CDF: Cumulative Distribution Function

CODEGEN: Code Generator

DAC: Decentralized Admission Control

DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems

DANTi: Detect and Avoid iN The Cockpit

DDDAS: Dynamic Data-Driven Applications Systems

FAA: Federal Aviation Administration

FAM: Failure-Aware Actor Model

FLP: Fischer, Lynch, and Paterson

HOL: Higher-Order Logic

ICAROUS: Independent Configurable Architecture for Reliable Operations of Unmanned Systems with Distributed On-board Services

IID: Independent and Identically Distributed

IoP: Internet of Planes

LAP: Low-Altitude Platform

MANET: Mobile Ad-Hoc Network

MTR: Multicopy Two-Hop Relay

NAS: National Airspace System

NASA: National Aeronautics and Space Administration

NMAC: Near Mid-Air Collision

PDF: Probability Density Function

PVS: Prototype Verification System

SPASS: Synergetic Prover Augmenting Superposition with Sorts

STAR: Standard Terminal Arrival Route

TAP: Two-Phase Acknowledge Protocol

TCAS: Traffic Collision Avoidance System

TLA: Temporal Logic of Actions

TLAPS: TLA$^+$ Proof System

TTL: Time-to-Live

UAM: Urban Air Mobility

UAS: Uncrewed Aircraft System

UAV: Uncrewed Aerial Vehicle

UDS: USS Discovery Service

USS: UAS Service Supplier

UTM: UAS Traffic Management

UTM CONOPS: UTM Concept of Operations

V2V: Vehicle-to-Vehicle

VANET: Vehicular Ad-Hoc Network

WCV: Well-Clear Volume

ACKNOWLEDGMENT

As I near the end of my PhD, I would like to look back at the last six years of my life as a graduate student in the United States and express my gratitude to the people who have supported me, in different capacities, during this journey.

First and foremost, I would like to thank my advisor, Dr. Carlos Varela, who gave me the most valuable thing anyone could have given me six years ago — *an opportunity*. I recall my struggle towards finding an advisor, with no prior research experience, when Dr. Varela decided to take a risk and give me a chance to prove myself. Since then, he has been a valuable mentor in my academic and professional journey. Without his experienced guidance and constant encouragement to do better research, this dissertation would not have been possible. For that, I owe him my utmost gratitude and respect.

I would like to thank my parents, to whom I owe the privileged life that allowed me to pursue a doctoral degree without having to worry about issues, financial or otherwise. They have been a constant source of guidance, support, and encouragement since childhood, and are solely responsible for shaping me into the person I am today. I would also like to thank my sister and my extended family for their best wishes and support.

Special thanks to Dr. Stacy Patterson, whom I have had the privilege of working with very closely throughout my doctoral research journey. She has been my co-advisor and a co-contributor in most of my important research milestones and has also served as a member of my doctoral committee. I would also like to thank the other members of my doctoral committee, Dr. Selmer Bringsjord and Dr. César A. Muñoz, for their valuable inputs and insightful feedback on my research that has helped me improve this dissertation. I am grateful to my professors at the Department of Computer Science at Rensselaer, especially Dr. Chuck Stewart and Dr. Sibel Adali, who have always been there for us graduate students and have helped us navigate various hurdles of graduate school at Rensselaer.

Living alone in a foreign country thousands of miles away from one's home is an emotionally taxing task in its own right. This is exacerbated by the never-ending series of obstacles presented by doctoral studies. Fortunately, I made some wonderful friends in Troy who never let me feel lonely during my time here. Anirban and Sudeshna have been a constant since I set foot in Troy back in 2016. Special thanks to Sounak and Sarthak for the wonderful memories we made in the small living room at 107A 14th street, often during the odd hours of the night. I am also grateful to Karabi, Prattoy, Kanu, Saptarshida, Indranidi, and Ashraful for creating our small tribe in Troy that allowed us to support each other through all the stressful times. Thanks to Kuntal and Shama, who accompanied me on my first flight to the US and had been a source of support during my early days in Troy. I will always remember the good times spent here in Troy, thanks to all these delightful people.

I want to express my gratitude to Michael Durling, Baoluo Meng, Suzanne Wolff, and members of the High-Assurance Systems Team at General Electric Research for providing me with industrial research exposure and other valuable opportunities during my internships in 2020 & 2021. Working with such amazing people has been an illuminating experience that has helped me perceive research from outside the perspective of academia.

I am grateful to Tracy Hoffman, Shannon Carrothers, and the late Terry Hayden who have kindly helped me maneuver through many official hurdles throughout the PhD journey. I would also like to thank Dawn Cairns-Weaver, Colleen Smith, and the staff of the ISSS for their patience and understanding in assisting me with various administrative issues.

I would like to thank my past and present labmates at the Worldwide Computing Laboratory — Shigeru, Elkin, Frederick, and Alexandra. Shigeru had helped me in my initial days at the lab by showing me the ropes to managing the lab duties and the lab infrastructure. Elkin, Frederick, and Alexandra have helped me with my research by generating data from the flight simulator and providing valuable insights and feedback on manuscripts.

Finally, I would like to thank the Rensselaer School of Science, the Air Force Office of Scientific Research (AFOSR), and the National Science Foundation (NSF) for their financial support throughout my doctoral journey.

ABSTRACT

As *autonomous vehicular technologies* such as self-driving cars and *uncrewed aircraft systems* (UAS) evolve to become more accessible and cost-efficient, *autonomous multi-agent systems*, that comprise of such entities, will become ubiquitous in the near future. The close operational proximity between such autonomous agents will warrant the need for *multi-agent coordination* to ensure safe operations. In this thesis, we adopt a *formal methods*-based approach to investigate multi-agent coordination for *safety-critical* autonomous multi-agent systems. We explore algorithms that can be used for *decentralized* multi-agent coordination among autonomous mobile agents by communicating over *asynchronous vehicle-to-vehicle* (V2V) networks that can be prone to *agent failures*. In particular, we study two types of distributed algorithms that are useful for decentralized coordination — *consensus*, which can be used by autonomous agents to agree on a set of compatible operations; and *knowledge propagation*, which can be used to ensure sufficient *situational awareness* in autonomous multi-agent systems. We develop the first machine-checked proof of *eventual progress* for the *Synod* consensus algorithm, that does not assume a *unique leader*. To consider agent failures while reasoning about progress, we introduce a novel *Failure-Aware Actor Model* (FAM). We then propose a formally verified *Two-Phase Acknowledge Protocol* (TAP) for knowledge propagation that can establish a *safe state of knowledge* suitable for autonomous vehicular operations. The non-deterministic and dynamic operating conditions of distributed algorithms deployed over asynchronous V2V networks make it challenging to provide appropriate formal guarantees for the algorithms. To address this, we introduce *probabilistic correctness properties* that can be developed by *stochastically* modeling the systems. We present a formal *proof library* that can be used for reasoning about probabilistic properties of distributed algorithms deployed over V2V networks. We also propose a *Dynamic Data-Driven Applications Systems* (DDDAS)-based approach for the *runtime verification* of distributed algorithms. This approach uses *parameterized proofs*, which can be instantiated at runtime, and *progress envelopes*, which can divide the operational state space into distinct regions where a proof of progress may or may not hold. To motivate our verification of decentralized coordination, we introduce an autonomous *air traffic management* (ATM) technique for multi-aircraft systems called *Decentralized Admission Control* (DAC).

CHAPTER 1
INTRODUCTION

In the near future, *autonomous vehicles* such as self-driving cars and *uncrewed aircraft systems* (UAS) will be used for applications such as passenger transportation, package delivery, disaster relief, agriculture surveillance, and weather tracking. Systems comprising of such autonomous entities can be termed as *autonomous multi-agent systems*. As autonomous vehicular technologies evolve to become more accessible and cost-efficient, such autonomous multi-agent systems will become ubiquitous in urban and semi-urban areas. The close operational proximity between such autonomous agents will warrant the need for *multi-agent coordination* to ensure their safe and secure operations. In this thesis, we investigate multi-agent coordination for future autonomous vehicular operations.

1.1 Autonomous Multi-Agent Coordination

Effective autonomous multi-agent coordination requires complex interactions between the agents in the form of messages exchanged via a suitable communication network. There are two primary ways in which autonomous agents can achieve coordination via communicating with one another — *centralized* and *decentralized* (Fig. 1.1).

1.1.1 Centralized Coordination

In centralized coordination, there is a distinguished agent known as the *coordinator* that can communicate with all other agents. The agents can only communicate with the

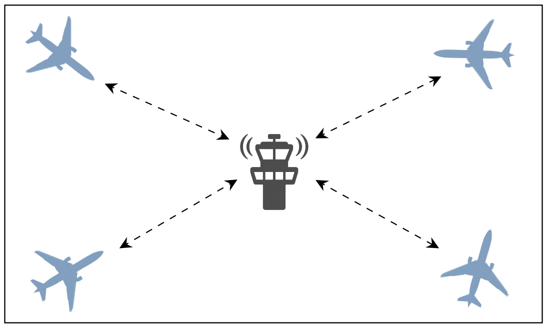
Portions of this chapter previously appeared as: Paul, S., Patterson, S., Varela, C.A.: Formal guarantees of timely progress for distributed knowledge propagation. In: Formal Methods for Autonomous Systems (FMAS), pp. 73-91 (2021). doi:10.4204/EPTCS.348.5

Portions of this chapter previously appeared as: Paul, S., Agha, G.A., Patterson, S., Varela, C.A.: Verification of eventual consensus in Synod using a Failure-Aware Actor Model. In: NASA Formal Methods Symposium (NFM), pp. 249-267 (2021). doi:10.1007/978-3-030-76384-8_16

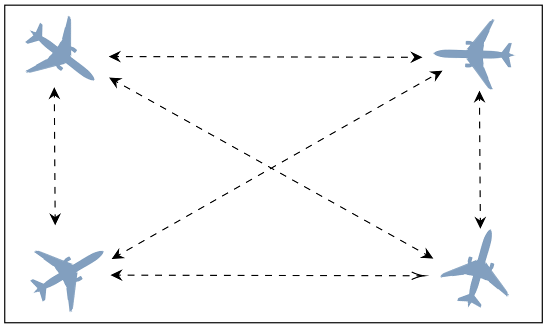
Portions of this chapter previously appeared as: Paul, S., Patterson, S., Varela, C.A.: Collaborative situational awareness for conflict-aware flight planning. In: IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 1-10 (2020). doi:10.1109/dasc50938.2020.9256620

Portions of this chapter previously appeared as: Paul, S., Kopsaftopoulos, F., Patterson, S., Varela, C.A.: Dynamic data-driven formal progress envelopes for distributed algorithms. In: Dynamic Data-Driven Application Systems (DDDAS), pp. 245-252 (2020). doi:10.1007/978-3-030-61725-7_29

Portions of this chapter previously appeared as: Paul, S., Patterson, S., Varela, C.A.: Conflict-aware flight planning for avoiding near mid-air collisions. In: IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 1-10 (2019). doi:10.1109/dasc43569.2019.9081658



(a)



(b)

Figure 1.1: Communication in (a) centralized vs (b) decentralized coordination.

coordinator and not directly among themselves. This model of coordination has poor *fault-tolerance* – if the coordinator malfunctions or fails for some reason, then there can be no coordination. It also creates an operational bottleneck since all messages have to be processed via the coordinator, which prevents this model of coordination from scaling efficiently to considerably large systems involving a large number of agents.

1.1.2 Decentralized Coordination

In decentralized coordination, there is no central coordinator and the agents can directly communicate with all other agents. If one agent fails, then the other agents may still manage to coordinate successfully, making it more fault-tolerant and robust than the centralized model. This type of coordination model also scales well to arbitrarily large systems as there is no operational bottleneck. However, guaranteeing successful coordination is significantly more complex in the decentralized model than in the centralized model as there is no central authority to act as an intermediary.

Autonomous vehicular systems are expected to be ubiquitous and involve an arbitrarily large number of vehicles connected in an *ad-hoc* manner in the near future. Establishing and maintaining dedicated fault-tolerant infrastructures to coordinate the safe operations of such systems will be very expensive. In situations when such centralized infrastructures have failed or are unavailable, the autonomous agents must be capable of coordinating in a decentralized manner to ensure safe operations. Therefore, in this thesis, we study the decentralized model of multi-agent coordination that can be used by autonomous multi-vehicular systems in situations when centralized infrastructures are not available.

1.2 Formal Methods

Autonomous vehicular systems are *safety-critical* in nature, *i.e.,* their failure or malfunction can be catastrophic. In engineering, imprecision can lead to errors, thereby presenting an incentive for using precise mathematical techniques for specifying and reasoning about the properties of critical systems. *Formal methods* bring forward 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 [123]. *Formal verification* of safety-critical systems requires tools and 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 defined syntax and semantics.
- *Verification* - the process of determining if the specified system satisfies the required properties using *theorem proving* and/or *model Checking*.

1.2.1 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], Temporal Logic of Actions (TLA) Proof System (TLAPS) [33], PVS [132], and Coq [35] can automatically search for and check proofs in defined domains.

1.2.2 Model Checking

In *model checking* [36], 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. 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 tools like Kind 2 [27], Alloy [90], and TLA$^+$ Model Checker [167].

Compared to model checking, theorem proving is a more expressive approach for the verification of properties 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. Unlike model checking, theorem proving also enables reasoning about complex data structures and recursive data types of arbitrary sizes, such as graphs, trees, and lists, by using *structural induction* [7]. 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.

Automated reasoning involves the use of computing systems to automatically make logical inferences [73]. *E.g.,* model checking can be used to automatically analyze the model of a system to check if some properties are satisfied and *automated theorem provers* (ATP) can be used to automatically search for proofs of properties from a set of logical statements. *Interactive theorem proving* involves the development of formal machine verifiable proofs manually in a theorem proving system and then mechanically verifying it. In this thesis, we employ formal theorem proving techniques for the verification of decentralized coordination.

1.3 Problem Description

The road to the realization of safe and efficient decentralized multi-agent coordination for autonomous vehicular systems is rife with several significant technical challenges. These challenges include, but are not limited to — ensuring sufficient situational awareness among the mobile agents by passing information through *asynchronous vehicle-to-vehicle* (V2V) networks [160]; the unpredictable and dynamic operating conditions of distributed protocols operating in such networks; and the safety-critical nature of autonomous vehicular operations. We first discuss some of these challenges and then define our problem statement.

1.3.1 The Challenges Towards Decentralized Multi-Agent Coordination

1.3.1.1 Ensuring Effective and Correct Cooperation Among Agents

In autonomous vehicular systems, the agents are *interdependent* as the operations undertaken by the individual agents are related. This is because the local decisions made by any arbitrary agent in a system can significantly affect the collective safe operation of all agents in the system. *E.g.*, in the case of multi-aircraft systems, if the aircraft plan their operations without considering the operations of other aircraft, then they may encounter catastrophic *mid-air collisions.* For ensuring the safe operation of autonomous vehicular systems, certain safety constraints must be satisfied globally. This requires that the autonomous agents cooperatively plan their collective operations instead of planning independent operations in isolation. Therefore, the agents must share operational intents and coordinate with one another by communicating through a communication medium like a V2V network.

1.3.1.2 Nature of Communication Between Mobile Agents

Mobile ad-hoc networks (MANET) are temporary communication networks that are formed by wireless mobile nodes which can freely and dynamically self-organize into arbitrary and temporary network topologies [34]. MANETs allow seamless internetworking in areas where there is no centralized communication infrastructure available. This makes them useful for decentralized communication in autonomous vehicular systems. MANETs comprised of vehicular agents are known as *vehicular ad-hoc networks* (VANET) [74]. Real-world communication networks are *asynchronous, i.e.,* the message processing and transmission delays are unbounded. Moreover, mobile agents, like aircraft, can experience temporary or permanent disruptions (or *failures*) in their communication capabilities which may be caused by them getting disconnected from the network. This makes it difficult to ensure correct and timely coordination between mobile agents connected via asynchronous ad-hoc networks as it is impossible to deterministically predict message transmission and processing delays, and consequently detect agent communication failures.

1.3.1.3 Unpredictable and Dynamic Operating Conditions

Owing to the ubiquitous nature of autonomous vehicular operations, the operating conditions of agents in such systems cannot be deterministically modeled in advance. Some examples of the dynamic factors that characterize the unpredictable operating conditions of vehicular systems are – the varying topologies of ad-hoc networks, agent failures, the effect of agent mobility patterns, the unbounded message transmission and processing delays, the real-time operational intents of the agents, and emergency conditions that may arise during operation (*e.g.* – aircraft can experience loss in engine performance during flight [135, 137]). Therefore, decentralized coordination algorithms used by multi-agent vehicular systems must be robust enough to dynamically adapt to such uncertain operating conditions.

1.3.1.4 Verification of Safety-Critical Multi-Agent Coordination

Owing to the safety-critical nature of vehicular operations, the multi-agent coordination algorithms used in vehicular systems must be rigorously verified to ensure that they operate with some strong *correctness guarantees*. *E.g.,* for an algorithm that can be used to transmit information between aircraft, a possible correctness guarantee may be that messages are sent to the intended recipients. Formal methods can be used for the rigorous verification of such correctness guarantees. However, agents in autonomous vehicular systems are expected to operate in uncertain and dynamic operating environments, which makes it challenging to provide formal guarantees that are relevant after the deployment of the systems. Guarantees that are developed in the pre-deployment stages may cease to be relevant if the *runtime* operating conditions do not follow the assumptions used in the formal development of the guarantees. Therefore, it is necessary to investigate approaches that can aid in the formal verification of multi-agent coordination algorithms deployed in the uncertain operating environments presented by autonomous vehicular operations.

1.3.2 Problem Statement

The primary question we pose in this thesis is — *"In the absence of a centralized coordinator, how can a group of autonomous agents, that are connected through an asynchronous communication network and are prone to failures, collaboratively coordinate to plan safe operations in a provably correct manner?"*

1.4 Approach

In this thesis, we try to address some of the aforementioned challenges by taking a formal methods-based approach to investigate decentralized multi-agent coordination in autonomous vehicular systems. We study *distributed algorithms* that can be used for decentralized coordination and verify that they follow some desirable correctness properties. In addition to using existing tools and techniques, we propose novel approaches to aid in our analysis of both deterministic and non-deterministic properties of distributed algorithms.

1.4.1 Decentralized Multi-Agent Coordination via Distributed Algorithms

In autonomous vehicular systems, each vehicle may have its own operational goals to fulfill that may not necessarily be compatible with the operational goals of the other vehicles. Therefore, to ensure safety in the absence of a centralized coordinator, the vehicles must cooperate and collaborate by communicating with one another to plan a set of compatible and safe operations.

Effective decentralized coordination involves several complex processes such as reaching an agreement on the same data and establishing sufficient awareness about data in a network. For achieving such decentralized coordination, autonomous vehicles can use distributed algorithms by communicating over the network. We study two main types of distributed algorithms that we believe are pertinent for achieving multi-agent coordination in autonomous vehicular systems — *consensus* and *knowledge propagation*.

1.4.1.1 Consensus

Consensus, which requires a set of agents to reach *an agreement* on some value, is a fundamental problem in distributed systems. *E.g.*, in a group of agents trying to access a shared resource, the agents must decide on the order in which to access the resource. Consensus algorithms typically involve a group of agents *voting* on values to *choose* which value should be agreed upon. Successful consensus implies that a sufficiently large subset of the set of agents has reached an agreement on some value. Various types of *consensus algorithms* such as Paxos [101], Synod [100], and Raft [131] have been proposed in the literature. Consensus algorithms are widely used for applications such as distributed lock services [24], cryptocurrency networks [168], smart power grids [165], and autonomous vehicles [150].

1.4.1.2 Knowledge Propagation

Knowledge can be defined as the state of an agent being aware of some fact. Several *states of knowledge* arise when analyzing the properties of knowledge relative to a group of agents. *E.g.*, if there are two agents Alice and Omar, and there is a fact that it is raining, some possible states of knowledge are – both Alice and Omar know that it is raining; only Alice knows that it is raining; and no one knows that it is raining. *Knowledge propagation* is the process of propagating a fact within a network of agents in order to attain a desired state of knowledge. A suitable *knowledge propagation algorithm* for an application must establish an appropriate application-specific state of knowledge upon completion.

We study distributed consensus and knowledge propagation algorithms for autonomous vehicular systems deployed using VANETs to guarantee two types of correctness properties that are necessary for safety-critical applications — (1) *the algorithm will behave as intended* (called *safety*) and (2) *the algorithm will achieve its intended goal* (called *progress*).

1.4.1.3 The System Model

In our formal analysis of distributed consensus and knowledge propagation algorithms, we make some common assumptions on the system properties. These assumptions are:

- the total number of agents in the network is arbitrary but constant,
- agents are non-Byzantine and non-adversarial, can temporarily or permanently fail, and have stable storage that can tolerate failures,
- message transmission and message processing delays are arbitrary, and
- messages cannot be lost.

While proving the correctness properties of the individual distributed algorithms that we study, we make additional assumptions beyond the common system model as necessary.

1.4.2 Formal Verification of Distributed Coordination Algorithms

We investigate the formal verification of distributed coordination algorithms deployed over ad-hoc networks by employing machine verifiable theorem proving techniques.

1.4.2.1 Reasoning About Asynchronous Communication

Mobile agents connected through ad-hoc networks like VANETs may experience permanent or temporary communication failures that may render them unable to send or receive any messages. This may be caused by all messages to and from an agent getting delayed because of transmission problems or due to internal processing failures in the agent. In asynchronous communication, since message transmission and processing delays are unbounded, it is not possible to distinguish transmission delays from processing delays or failures. However, when reasoning about distributed algorithms, it is necessary to take into account if an agent has failed at any given time, *i.e.*, if it is incapable of sending or receiving messages.

The *actor model* [1, 85] is a theoretical model of concurrent computation that can be used for formal reasoning about distributed algorithms. It assumes some *fairness* guarantees that are useful for reasoning about the progress of distributed systems and assumes asynchronous communication to be the most primitive form of interaction between *actors*. The actor model is suitable for reasoning about the properties of algorithms operating in open distributed systems as it allows the creation of new actors, replacement of existing components, and dynamic changes in component interactions. Agha, Mason, Smith, and Talcott presented an actor language (AMST) [3], which provides a suitable foundation for reasoning about the properties of actor systems, as an extension of a simple functional language. However, AMST does not support reasoning about actor failures. Therefore, we propose a novel *Failure-Aware Actor Model* (FAM) that can be used to formally reason about temporary or permanent failures in agents connected through ad-hoc networks.

1.4.2.2 Probabilistic Properties of Distributed Algorithms

The operating conditions of distributed coordination algorithms deployed over VANETs are rife with uncertainties that cannot be predicted accurately in advance, making it difficult to provide guarantees for deterministic correctness properties. *E.g.,* it is not possible to deterministically predict the amount of time that may be required to attain consensus in a VANET as this depends on many dynamic factors such as the number of messages involved, the message transmission and processing delays, and the mobility pattern of the agents.

When deterministic properties cannot be guaranteed, it is still beneficial to have some properties that may not be completely accurate but can help the agents make important operational decisions. *E.g.,* a property may probabilistically bound the total time required for the progress of a distributed algorithm to some maximum value. Therefore, we investigate mechanically verified guarantees of *probabilistic correctness properties* for distributed algorithms deployed over VANETs. We stochastically model the different uncertain aspects of the operating conditions, such as the message transmission and processing delays, to provide guarantees of probabilistic properties. This enables the generation of useful formal properties that are cognizant of the uncertainties of the operating environment.

1.4.2.3 Proof Library to Reason About Distributed Algorithms

Formal verification of high-level properties of distributed algorithms deployed over ad-hoc networks involves holistic consideration of various domain-specific aspects. 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. Developing such expressive formal constructs in a machine-readable language is a challenging task since it requires — domain knowledge of all aspects of the systems that need to be specified; knowledge of formal logic and reasoning techniques; and mastery in the language in which the constructs are to be developed. Therefore, we present a *formal proof library* in Athena to aid in the development of formal machine verifiable guarantees of correctness for distributed algorithms deployed over VANETs.

1.4.2.4 Runtime Verification of Distributed Algorithms

The dynamic operating conditions of distributed coordination algorithms deployed over ad-hoc networks make it challenging to provide formal guarantees that are relevant after deployment. This is because formal proofs of correctness, that are developed in the pre-deployment stages, may cease to be relevant at runtime if the operating conditions do not conform to the assumptions made during reasoning. Since developing machine verifiable formal proofs of complex properties takes a considerable amount of skill, expertise, and time, it is not feasible to develop them from scratch at runtime to keep up with the dynamic operating conditions. This creates a need for methods that can enable the use of formal theorem proving techniques to verify the properties of systems operating in dynamic conditions.

Dynamic Data-Driven Applications Systems (DDDAS) is a paradigm that allows an application to dynamically incorporate new data to enhance an existing model of a system [40]. DDDAS enables an application to influence the data collection and measurement processes in real-time and give way to more effective collection and measurement of data, thus leading to a better quality of data specifically suited for the application. Using the DDDAS paradigm, we propose a set of novel techniques — a *formal DDDAS feedback loop*, *parameterized proofs*, and *progress envelopes* — that can be used to extend the practicality of formal theorem proving methods to the pre-deployment stages. Our proposed techniques allow the instantiation of appropriate pre-developed formal proofs at runtime using real-time data, thereby allowing the generation of new formal properties at runtime that retain the mathematical rigor of formal theorem proving methods.

1.5 Multi-Agent Coordination in Aerospace Systems

Urban air mobility (UAM) [156] is an emerging concept that focuses on the safe and efficient air traffic operations in metropolitan areas for crewed and uncrewed aircraft systems. One of the most important considerations for UAM is the *safe integration* of UAS in the National Airspace System (NAS) to ensure that UAS operations in the NAS do not cause any harm to life, property, or the environment. Potential UAM operations for commercial passenger transportation have been put forth by several private companies. Government agencies such as the Federal Aviation Administration (FAA) and the National Aeronautics and Space Administration (NASA) have also been involved in the research and development of prospective UAM operations in the NAS [58, 156]. Nevertheless, successful realization of UAM operations is presently constrained by many factors such as designing suitable *air traffic management* (ATM) techniques, safety-critical concerns, and the development of proper infrastructure to support the high-density air traffic of the future. For this reason, programs such as the *Next Generation Air Transportation System* (NextGen) and the *Single European Sky Air Traffic Management Research* (SESAR) are being developed in the United States and Europe respectively [43].

1.5.1 Autonomous Air Traffic Management (ATM)

Air traffic management encompasses all systems and techniques that are used to ensure the safe operation of aircraft within an airspace by maintaining *safe separation* between the aircraft. The loss of safe separation between aircraft can have catastrophic consequences like *near mid-air collisions* (NMAC) [66] and *wake-vortex* induced *rolls* [113]. Currently, *air traffic control* (ATC) is performed by human *air traffic controllers* and is complemented by *tacti-*

cal conflict-management tools, such as the *Traffic Collision Avoidance System* (TCAS) [57], that help minimize the risk of NMACs at high-altitude flight levels. This centralized, human-operated system of ATC is prone to human errors and its efficiency is affected by the cognitive workload of the air traffic controllers. In fact, several fatal accidents, such as in 2002 at Überlingen [23], have resulted, in part, due to errors made by human air traffic controllers. This creates an incentive for designing algorithms and standards that can enable aircraft to autonomously coordinate in a decentralized manner. Such autonomous operations are free from human errors and are amenable to formal verification, making them suitable for *UAS traffic management* (UTM) in safety-critical UAM operations.

1.5.2 The Internet of Planes (IoP)

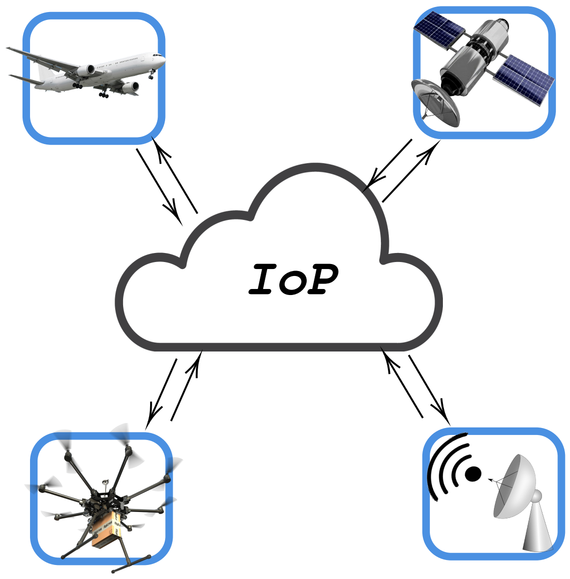


Figure 1.2: The Internet of Planes.

Aircraft participating in autonomous ATM operations must be capable of communicating with each other to exchange air traffic information. For this reason, we envision an *Internet of Planes* (IoP), which is a V2V network of aircraft, ground stations, and satellites that can be used by autonomous aircraft to share important air traffic data. Similar airborne networks have been proposed previously in the literature for communication between aerospace systems (*e.g.* – *Airborne Internet* [88, 116], *Space Internet* [17], and *Ad-Hoc UAS and Ground Network* (AUGNet) [61]). In this work, we assume that the IoP is an asynchronous ad-hoc network where message transmission and processing delays are unbounded, communication is affected by factors such as the mobility of aircraft, and the aircraft can experience temporary or permanent communication failures.

1.5.3 Decentralized Admission Control (DAC)

We propose a technique called *Decentralized Admission Control* (DAC) that can be used by aircraft to collaboratively plan operations through a common four-dimensional airspace.

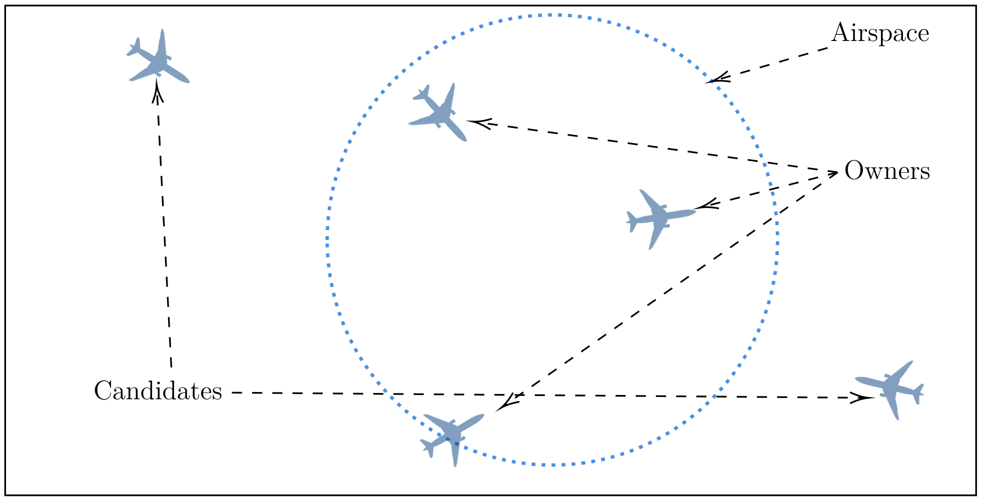


Figure 1.3: Aircraft trying to enter a controlled airspace using DAC.

DAC is an autonomous *separation assurance* [51] approach that allows aircraft to collaboratively plan safe operations through well-defined four-dimensional airspaces in a completely decentralized manner. In DAC, a four-dimensional 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 NMACs between any subset of the 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 flight plans of the owners (Fig. 1.3). A candidate can then propose its conflict-free flight plan to the owners and request authorization to use the airspace. When a new candidate aircraft is authorized to use the airspace, the set of owners of the airspace changes to include the new aircraft. To ensure safe operations, all future candidates must use the new set of owners to compute their respective conflict-free proposals.

1.5.4 Multi-Aircraft Coordination in DAC

In DAC, participating aircraft must coordinate in a decentralized manner to plan safe operations through a common four-dimensional airspace. Decentralized coordination in DAC involves the following consecutive phases, in order:

- *The consensus phase:*

  As there may be multiple candidates concurrently competing for admission into an airspace, the owners can only admit the candidates sequentially. This is because the competing candidates do not consider each other in their proposals — in fact, a candidate may not even be aware of other candidates. Thus, admitting one candidate potentially invalidates the proposals of all other candidates. For this reason, the owners must use a distributed consensus algorithm in the first stage to ensure that only a single candidate's proposal is chosen at a time.

- *The knowledge propagation phase:*

  After a candidate is authorized, the set of owners changes to include the new candidate. Hence, after successful consensus on a single candidate, all members of the new set of owners (that includes the newly admitted candidate) must be informed about the agreement. In fact, to ensure that the aircraft can operate in a decentralized manner with a sense of "safety", an appropriate *safe state of knowledge* about the agreement is required with respect to the new set of owners. However, successful consensus in the first phase only guarantees that a non-empty subset of the set of owners, which participated in consensus, will learn about the agreement (Fig. 1.4). Therefore, a knowledge propagation algorithm is required in the second phase to sufficiently propagate the knowledge of consensus to the new set of owners for attaining the safe state.

Since DAC is completely autonomous, it is free from human errors and is amenable to formal verification, making it suitable for safety-critical UAM operations. DAC is also conceptually similar to the *UTM Concept of Operations* (UTM ConOps) [58] approach that involves *UAS Service Suppliers* (USS) and a *USS Discovery Service* (UDS). In the UTM ConOps, there are multiple USS, each of which is responsible for managing the operations of a single set of aircraft. There is a single UDS that maintains the information of a set of operations that have already been authorized. A USS uses the UDS of a particular

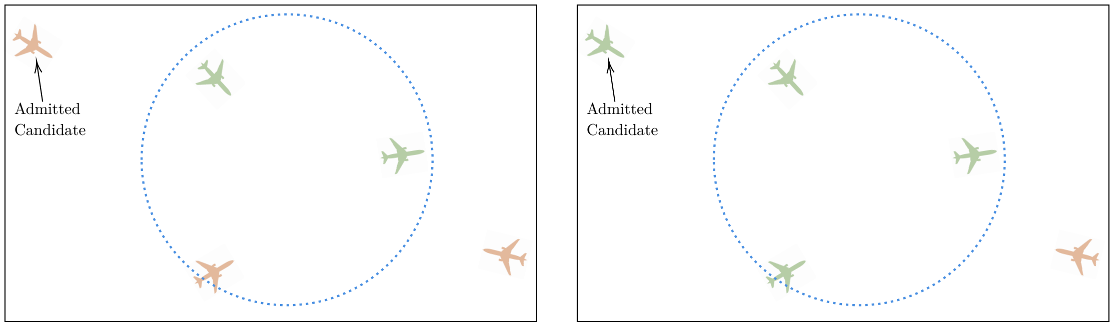


Figure 1.4: Representation of the state of knowledge guaranteed by (a) consensus and (b) knowledge propagation for DAC (green represents that the agent will learn a fact upon completion while red represents that the agent may not learn the fact upon completion).

operational area like a *lookup table* to learn about the authorized operations of other USS in the area in order to propose new conflict-free operations. In DAC, the set of owners and their flight plans, that is maintained and updated collaboratively by the aircraft, acts as the lookup table or the UDS, while each aircraft acts as its own unique USS.

In the rest of the thesis, we will use DAC to motivate our formal analysis of decentralized coordination algorithms for safety-critical autonomous multi-agent systems.

1.6 Thesis Contributions

The specific contributions of this thesis are:

1. Verification of Distributed Algorithms for Decentralized Coordination
   (a) The first machine-checked proof of eventual progress in the Synod consensus protocol under completely asynchronous conditions [136].
   (b) A formally-verified *Two-Phase Acknowledge Protocol* (TAP) for attaining a safe state of knowledge suitable for decentralized multi-aircraft coordination [140].
2. Formal Methods for Distributed Systems
   (a) A Failure-Aware Actor Model that can be used for formal reasoning about temporary or permanent agent failures in VANETs [136].

(b) Formal guarantees of probabilistic properties of timely progress for distributed algorithms deployed over VANETs [141].

(c) A formal proof library in Athena that is tailored towards reasoning about probabilistic properties of distributed algorithms deployed over VANETs [141].

(d) A data-driven approach for the runtime verification of dynamic distributed applications using a formal DDDAS feedback loop, progress envelopes, and parameterized proofs [138].

3. A formally-verified algorithm that can be used by candidates in DAC to compute conflict-free flight plans with respect to a set of owners [139].

1.7 Related Work

In this section, we provide a global view of related work in the literature on the verification of distributed systems and stochastic systems. Interested readers are suggested to refer to the "Related Work" section of the different chapters of the thesis for a more detailed analysis of relevant prior work.

1.7.1 Formal Verification of Distributed Systems

Küfner *et al.* [96] provided a methodology to develop machine-checkable proofs of fault-tolerant round-based distributed systems. Schiper *et al.* [152] have formally verified the safety property of a totally ordered broadcast algorithm using EventML [18] and the NuPRL [129] proof assistant. Howard *et al.* [86] have presented *Flexible Paxos* by introducing flexible quorums for the Paxos consensus protocol and have model checked its safety property using the TLA$^+$ model checker. McMillan *et al.* [115] machine-checked and verified the proofs of safety and progress properties of *Stoppable Paxos* [114] using Ivy [134]. Dragoi *et al.* [47] introduced PSync, a language that allows writing, execution, and verification of high-level implementations of fault-tolerant distributed systems. Hawblitzel *et al.* [82, 83] introduced IronFleet, a framework for designing provably correct distributed algorithms by embedding TLA$^+$ specifications in Dafny [108]. Chand *et al.* [28] have used TLAPS [33] to prove the safety property of the *Multi-Paxos* consensus algorithm. Musser and Varela [127] presented a formalization of the actor model in Athena and developed several important theorems about the fundamental properties of actor systems.

1.7.2 Verification of Dynamic and Stochastic Systems

Hasan [75] presented formalizations of statistical properties of discrete random variables in the *HOL Theorem Prover* [67]. Hasan *et al.* have formalized properties of the standard uniform random variable [76], discrete and continuous random variables [77, 79], tail distribution bounds [78], and conditional probability [80]. Mhamdi *et al.* [120, 121] have extended Hasan *et al.*'s work by formalizing *measure theory* and the *Lebesque integral* in HOL. HOL formalizations of the Poisson process, *continuous chain Markov process*, and M/M/1 queue have been presented in [29]. Qasim [146] has used Mhamdi *et al.*'s work to formalize the *standard normal variable*. Mitsch and Platzer [122] have presented ModelPlex, a runtime validation tool that can monitor the runtime behavior of a distributed system against verified properties. Pike *et al.* [142] presented Copilot, a language and compiler that can be used to generate runtime monitors for real-time distributed systems.

1.8 Thesis Outline

The rest of the thesis is structured as follows: Chapter 2 presents our work on verification of the eventual progress property of Synod using the Failure-Aware Actor Model; Chapter 3 presents TAP, our distributed knowledge propagation protocol for DAC; Chapter 4 presents our approach for developing probabilistic properties for distributed algorithms deployed in uncertain operating environments and our Athena proof library for machine-checking such properties; Chapter 5 presents our approach for the runtime verification of distributed algorithms using real-time observations about dynamic operating conditions; Chapter 6 presents a conflict-aware flight planning algorithm for DAC that can be used by the candidates to compute conflict-free proposals; and Chapter 7 concludes the thesis with a discussion on our work and presents potential future directions of work.

CHAPTER 2
PROGRESS IN THE SYNOD CONSENSUS PROTOCOL

2.1 Chapter Introduction

Consensus, which requires a set of agents to reach an agreement on some value, is a fundamental problem in distributed systems. In autonomous vehicular applications like Decentralized Admission Control, where there is no centralized coordinator to manage safe operations, the agents must use distributed consensus protocols for coordination. Any consensus protocol used for safety-critical applications such as DAC must satisfy two important correctness properties — *safety*, which implies that only one value will be agreed upon, and *progress*, which implies that an agreement *will happen*.

In [100], Lamport described three variants of a consensus protocol that strongly guarantees the safety property that only one value will be chosen:

- *The Basic Synod protocol* or Synod guarantees safety if one or more agents are allowed to initiate new proposals for consensus.
- *The Complete Synod protocol* or Paxos is a variant of Synod in which only a distinguished agent (*leader*) is allowed to initiate proposals [101]. Other agents may only introduce values through the leader. This is done to guarantee the progress property that consensus will eventually be achieved.
- *The Multi-Decree protocol* or Multi-Paxos allows multiple values to be chosen using a separate instance of Paxos for each value, but using the same leader for each of those instances.

Both Paxos and Multi-Paxos use Synod as the underlying consensus protocol. However, a progress guarantee contingent upon a unique leader has several drawbacks from the perspective of DAC. First, *leader election* itself is a consensus problem. Therefore, a progress guarantee that relies on successful leader election as a precondition would be circular, and therefore fallacious. Second, the Fischer, Lynch, and Paterson *impossibility result* (FLP) [60] implies that unique leadership cannot be guaranteed in asynchronous systems where agents

Portions of this chapter previously appeared as: Paul, S., Agha, G.A., Patterson, S., Varela, C.A.: Verification of eventual consensus in Synod using a Failure-Aware Actor Model. In: NASA Formal Methods Symposium (NFM), pp. 249-267 (2021). doi:10.1007/978-3-030-76384-8_16 may unpredictably fail. In some cases, network partitioning may also erroneously cause multiple leaders to be elected [6]. Third, airborne networks like the IoP are expected to be highly dynamic where membership frequently changes. Consensus is used in DAC for agreeing on only a single candidate, after which the set of owners changes by design. Hence, the benefits of electing a stable leader that apply for *state machine replication* [93], where reconfiguration is expected to be infrequent [105], are not applicable to DAC. Finally, channeling proposals through a unique leader creates a communication bottleneck and introduces the leader as a single point of failure: there can be no progress if the specified leader fails. In the absence of a unique leader, a system implementing Paxos falls back to the more general Synod protocol. Therefore, in this chapter, we focus on identifying sufficient conditions under which the fundamental Synod protocol can make eventual progress.

The *actor model* [1, 85] is a theoretical model of concurrent computation that can be used for formally reasoning about the properties of distributed algorithms. It assumes asynchronous communication as the most primitive form of interaction and it also assumes *fairness*, which is useful for reasoning about the progress of actor systems. In the context of DAC, aircraft may experience temporary or permanent communication failures where they are unable to send or receive any messages. This may be caused by all messages to and from an aircraft getting delayed because of transmission problems or due to internal processing delays (or processing failures) in the aircraft. In asynchronous communication, since message delays are unbounded, it is not possible to distinguish transmission delays from processing delays or failures. However, our verification of eventual progress in Synod must consider the possibility of such communication failures in actors.

In this chapter, we first identify a set of sufficient asynchronous conditions under which it is possible to guarantee eventual progress in Synod without restricting proposal rights to a distinguished agent (or leader). Then, to support explicit reasoning about temporary or permanent failures in actors, we introduce a Failure-Aware Actor Model that assumes *predicate fairness* [147] in addition to the actor model's fairness properties. Predicate fairness states that if a predicate is *infinitely often enabled* in a given path, then it must be eventually satisfied. To model failures, FAM incorporates modified semantics of Varela's dialect [158] of Agha, Mason, Smith, and Talcott's actor language AMST [3]. Finally, we use FAM to model Synod and its progress properties and develop a formal proof of eventual progress in Synod under the identified set of conditions. Furthermore, in order to ensure that our formalization is correct, we machine-check our proof using the Athena proof assistant. At the time of publication, this is the first machine-checked proof of progress for the Synod consensus protocol, under completely asynchronous conditions, in the literature.

The rest of the chapter is structured as follows: Section 2.2 informally describes Synod and presents our set of conditions for eventual progress; Section 2.3 introduces FAM and discusses some interesting properties of FAM; Section 2.4 presents the formal verification of eventual progress in Synod using FAM; Section 2.5 discusses prior research related to our study of consensus and the actor model; and Section 2.6 concludes the chapter with a summary and presents some potential future directions of work.

2.2 The Synod Protocol

For Synod, we assume an asynchronous, non-Byzantine system model in which agents can fail and have stable storage that can tolerate failures. Messages can be duplicated, and have arbitrary transmission times, but cannot be corrupted. We also assume that messages cannot be lost. The protocol consists of two logically separate sets of agents:

- *Proposers* - The set of agents that can propose values to be chosen.
- *Acceptors* - The set of agents that can vote on which value should be chosen.

Synod requires a subset of acceptors, which satisfy a *quorum*, to proceed. To ensure a unique agreement, if there is a consensus in one quorum, then there cannot also be another quorum with a different consensus. For this reason, the safety property of Synod requires that *any two quorums must intersect*[1]. A simple example of a quorum is a subset of the set of all acceptors which constitutes a simple majority. [86] presents other methods for determining quorums, including flexible quorums, but in this paper, we only consider fixed quorums. Additionally, we assume that the set of agents participating in an instance of the protocol does not change with time.

There are four types of messages in Synod:

- *prepare (1a)* messages include a *proposal number*.
- *accept (2a)* messages include a proposal number and a *value*.

[1]This condition is only required for proving safety and is not required for proving progress independently.

- *promise (1b)* messages include a proposal number and a value.
- *voted (2b)* messages include a proposal number and a value.

For each proposer, the algorithm proceeds in two distinct *phases* [101]:

- **Phase 1**
  (a) A proposer P selects a *unique proposal number* b and sends a *prepare* request with b to a subset Q of acceptors, where Q constitutes a quorum.
  (b) When an acceptor A receives a *prepare* request with the proposal number b, it checks if b is greater than the proposal numbers of all *prepare* requests to which A has already responded. If this condition is satisfied, then A responds to P with a *promise* message. The *promise* message implies that A will not accept any other proposal with a proposal number less than b. The promise includes (i) the highest-numbered proposal b' that A has previously accepted and (ii) the value corresponding to b'. If A has not accepted any proposals, it simply sends a default value.
- **Phase 2**
  (a) If P receives a *promise* message in response to its *prepare* requests from all members of Q, then P sends an *accept* request to all members of Q. These *accept* messages contain the proposal number b and a value v, where v is the value of the highest-numbered proposal among the responses or an arbitrary value if all responses reported the default value.
  (b) If an acceptor A receives an *accept* request with a proposal number b and a value v from P, it accepts the proposal unless it has already responded to a *prepare* request having a number greater than b. If A accepts the proposal, it sends P a *voted* message which includes b and v [100].

A proposer *determines* that a proposal was successfully *chosen* if and only if it receives *voted* messages from a quorum for that proposal.

A proposer may initiate multiple proposals in Synod, but each proposal that is initiated in Synod must have a unique proposal number.

2.2.1 Safety in Synod

Safety in Synod implies that only one value will be agreed upon by the acceptors. Synod allows multiple proposals to be chosen. Safety is ensured by the invariant - *"If a proposal with value v is chosen, then every higher numbered-proposal that is chosen has value v"* [101]. Therefore, there may be situations where a proposer P proposes a proposal number after one or more lower-numbered proposals have already been chosen, resulting in some value v being agreed upon. By design, Synod will ensure that P proposes the same value v in its Phase 2. Since proposers can initiate proposals in any order and communication is asynchronous, the only fact that can be guaranteed about any chosen value is that it must have been proposed by the first proposal to have been chosen by any quorum. From the context of admission control, it implies that if a candidate P_2 successfully completes both phases after another candidate P_1 has completed both phases, then P_2 will simply learn that P_1 has been granted admission. So P_2 will update its set of owners to include P_1, create a new conflict-aware flight plan, and request admission by starting the Synod protocol again.

2.2.2 Progress in Synod

Progress in Synod implies that eventually, some value will be agreed upon. Some obvious scenarios in which progress may be affected in Synod are:

- Two proposers P_1 and P_2 may complete Phase 1 with proposal numbers b_1 and b_2 such that $b_2 > b_1$. This will cause P_1 to fail Phase 2. P_1 may then propose a fresh proposal number $b_3 > b_2$ and complete Phase 1 before P_2 completes its Phase 2. This will cause P_2 to fail Phase 2 and propose a fresh proposal number $b_4 > b_3$. This process may repeat infinitely [101] (*livelock*).

- Progress may be affected even if one random agent fails unpredictably [60].

Paxos assumes that a distinguished proposer (leader) is elected as the only proposer that can initiate proposals [100]. Lamport states *“If the distinguished proposer can communicate successfully with a majority of acceptors, and if it uses a proposal with number greater than any already used, then it will succeed in issuing a proposal that is accepted.”* [101]. The FLP impossibility result [60] implies that in purely asynchronous systems, where agent failures cannot be differentiated from message delays, leader election cannot be guaranteed. Moreover, it is possible that due to network partitioning, multiple proposers are elected as leaders [6]. In the absence of a unique leader, a system implementing Paxos falls back to Synod. Therefore, it is important to identify the fundamental conditions under which progress can be formally guaranteed in Synod in the absence of a unique leader.

To guarantee progress, it suffices to show that *some* proposal number b will be chosen. This will happen if b satisfies the following conditions:

P1 When an acceptor A receives a *prepare* message with b, b should be greater than all other proposal numbers that A has previously seen.

P2 When an acceptor A receives an *accept* message with b, b should be greater than or equal to all other proposal numbers that A has previously seen.

P1 and *P2* simply suggest that for b, there will be a long enough period without *prepare* or *accept* messages with a proposal number greater than b, allowing messages corresponding to b to get successfully processed without being interrupted by messages corresponding to any higher-numbered proposal.

Since progress cannot be guaranteed if too many agents permanently fail or if too many messages are lost, Lamport [104] presents some conditions for informally proving progress in Paxos. A *nonfaulty* agent is defined as *"an agent that eventually performs the actions that it should"*, and a *good* set is defined as a set of nonfaulty agents, such that, if an agent repeatedly sends a message to another agent in the set, it is eventually received by the recipient. It is then assumed that the unique leader and a quorum of acceptors form a good set and that they infinitely repeat all messages that they have sent. These conditions are quite strong since they depend on the future behavior of a subset of agents and may not always be true of an implementation. However, since they have been deemed reasonable for informally proving progress even in the presence of a unique leader, we partially incorporate them in our conditions under which progress in Synod can be formally guaranteed in the absence of a unique leader. Our complete conditions for guaranteeing progress in Synod, therefore, informally state that *"eventually, a nonfaulty proposer must propose a proposal number, that will satisfy P1 and P2, to a quorum of nonfaulty acceptors, and the Synod-specific messages between these agents must be eventually received"*.

We can see that the conditions for eventual progress in Paxos constitute a special case of our conditions for progress in Synod where *P1* and *P2* are satisfied by a proposal proposed by the unique leader. If the unique leader permanently fails, then the corresponding guarantee is only useful if leader re-election is successful. However, if leader election is assumed to have already succeeded, there will be no need for further consensus, rendering the guarantee moot. Synod's eventual progress guarantee remains useful as long as at least one proposer is available to possibly propose at least one successful proposal, thereby remaining pertinent even if multiple (not all) proposers arbitrarily fail. Moreover, our conditions do not assume that consensus (leader election) will have already succeeded.

It is important to note here that a guarantee of eventual progress only states that eventually, consensus will be achieved. It does not provide any bound on the amount of time that may be required for consensus. As air traffic data usually has a short useful lifetime and aircraft have limited time to remain airborne, a guarantee of eventual progress *alone* is insufficient for UAM applications. To be useful, a progress guarantee must provide some time bounds that the aircraft can use to make important decisions. *E.g.*, if there is a guarantee that consensus will take at most 1 second, then a candidate can decide to only compute flight plans that start after 1 second [138]. However, a guarantee of eventual progress is a necessary precondition for providing a bound for the time that may be required for progress, making our work important for time-critical applications like DAC.

2.3 The Failure-Aware Actor Model (FAM)

Actors are self-contained, concurrently interacting entities of a computing system that communicate by message passing [2]. We use the actor model to formally reason about eventual consensus in Synod since it assumes asynchronous communication and fairness, which is helpful for reasoning about progress in asynchronous airborne communication networks like the Internet of Planes [3].

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. This may be caused by all messages to and from an aircraft getting delayed because of transmission problems or due to internal processing delays (or processing failures) in the aircraft. In asynchronous communication, since message delays are unbounded, it is not possible to distinguish transmission delays from processing delays or failures. However, it is important to take into account if an actor has failed at any given time, *i.e.*, if it is incapable of sending or receiving messages. To consider such asynchronous actor failures while reasoning about progress, we introduce our Failure-Aware Actor Model.

$$\frac{e \rightarrow_\lambda e'}{\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright e \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{fun}:a]} \langle\!\langle \alpha, [\mathsf{R} \blacktriangleright e' \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle}$$

$$\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{new}(b) \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{new}:a,a']} \langle\!\langle \alpha, [\mathsf{R} \blacktriangleright a' \blacktriangleleft]_a, [\texttt{ready}(b)]_{a'} \parallel \bar{\alpha} \parallel \underset{a' \textit{ fresh}}{\mu} \rangle\!\rangle$$

$$\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{send}(a', v) \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{snd}:a]} \langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{nil} \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \mu \uplus \{\langle a' \Leftarrow v \rangle\} \rangle\!\rangle$$

$$\langle\!\langle \alpha, [\mathsf{R} \blacktriangleright \texttt{ready}(b) \blacktriangleleft]_a \parallel \bar{\alpha} \parallel \{\langle a \Leftarrow v \rangle\} \uplus \mu \rangle\!\rangle \xrightarrow{[\mathbf{rcv}:a,v]} \langle\!\langle \alpha, [b(v)]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle$$

Figure 2.1: Operational semantics for the base-level transition rules.

$$\langle\!\langle \alpha, [e]_a \parallel \bar{\alpha} \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{stp}:a]} \langle\!\langle \alpha|_{\mathit{dom}(\alpha)-\{a\}} \parallel \bar{\alpha}, [e]_a \parallel \mu \rangle\!\rangle$$

$$\langle\!\langle \alpha \parallel \bar{\alpha}, [e]_a \parallel \mu \rangle\!\rangle \xrightarrow{[\mathbf{bgn}:a]} \langle\!\langle \alpha, [e]_a \parallel \bar{\alpha}|_{\mathit{dom}(\bar{\alpha})-\{a\}} \parallel \mu \rangle\!\rangle$$

Figure 2.2: Operational semantics for the meta-level transition rules.

FAM models two states for an actor at any given time—*available* or *failed*. Actors can switch states as *transitions* between *configurations*. From the perspective of message transmission and reception, a failed actor cannot send or receive *any* messages, but an available actor can. This allows FAM to formally model communication failures between aircraft in the IoP. The failure model of FAM also assumes that every actor has a stable storage that is persistent across failures. In addition to the actor model's fairness assumptions, FAM assumes *predicate fairness* [147], which states that a predicate that is *infinitely often enabled* in a given path will eventually be satisfied.

2.3.1 The Semantics of FAM

Varela [158] presents a dialect (AM) of AMST's *lambda-calculus*-based actor language [3] whose operational semantics are a set of *labeled transitions* from *actor configurations* to actor configurations[2]. An actor configuration κ is a temporal snapshot of actor system components, namely the individual actors and the messages "en route". It is denoted by $\langle\!\langle \alpha \parallel \mu \rangle\!\rangle$, where α is a map from actor names to *actor expressions*, and μ is a multi-set of messages. An actor expression e is either a value v or a *reduction context* R filled with a *redex* r, de-

[2]Interested readers can refer to section 4.5 of [158] for more details about the language.

noted as $e = \mathsf{R} \blacktriangleright \mathsf{r} \blacktriangleleft$. $\kappa_1 \xrightarrow{l} \kappa_2$ denotes a *transition rule* where κ_1, κ_2, and l are the initial configuration, the final configuration, and the transition label respectively. There are four possible transitions – **fun**, **new**, **snd**, and **rcv**. A *computation sequence* is a finite sequence of labelled transitions $[\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n]$ for some $n \in \mathbb{N}$. Sequences are partially ordered by the *initial segment* relation. A *computation path* from a configuration κ is a maximal linearly ordered set of sequences in the *computation tree*, $\tau(\kappa)$.

To model failures, we modify Varela's dialect of AMST and categorize its original transitions (**fun**, **new**, **snd**, and **rcv**) as *base-level* transitions. For a base-level transition in FAM to be enabled to occur for an *actor in focus* at any time, the actor needs to be available at that time. To denote available and failed actors at a given time in FAM, we redefine an actor configuration as $\langle\langle \alpha \parallel \bar{\alpha} \parallel \mu \rangle\rangle$, where α is a map from actor names to actor expressions for available actors, $\bar{\alpha}$ is a map from actor names to actor expressions for failed actors, and μ is a multi-set of messages "en route".

To model actor failure and restart, we define two *meta-level* transitions **stp** (*stop*) and **bgn** (*begin*) that can stop an available actor or start a failed actor in its persistent state before failure. The **stp** transition is only enabled for an actor in the available state and the **bgn** transition is only enabled for an actor in the failed state. Fig. 2.1 and Fig. 2.2 show the operational semantics of our actor language as labelled transition rules[3,4].

For an actor configuration $\kappa = \langle\langle \alpha \parallel \bar{\alpha} \parallel \mu \rangle\rangle$ to be syntactically well-formed in our actor language, it must conform to the following[5]:

1. $\forall a,\, a \in \mathit{dom}(\alpha) \cup \mathit{dom}(\bar{\alpha}),\; \mathit{fv}(\alpha(a)) \subseteq \mathit{dom}(\alpha) \cup \mathit{dom}(\bar{\alpha})$
2. $\forall m,\, m \in \mu,\, m = \langle a \Leftarrow v \rangle,\; \mathit{fv}(a) \cup \mathit{fv}(v) \subseteq \mathit{dom}(\alpha) \cup \mathit{dom}(\bar{\alpha})$
3. $\mathit{dom}(\alpha) \cap \mathit{dom}(\bar{\alpha}) = \emptyset$

2.3.2 Fairness in FAM

FAM assumes a general fairness property called *predicate fairness* [147]. Predicate fairness states that *"if a predicate is infinitely often enabled in a given path, then eventually*

[3] $\rightarrow_\lambda$ denotes lambda calculus semantics, essentially beta-reduction. `new`, `send`, and `ready` are actor redexes. $\langle a \Leftarrow v \rangle$ denotes a message for actor a with value v. $\alpha, [e]_a$ denotes the extended map α', which is the same as α except that it maps a to e. $\uplus$ denotes multiset union. $\alpha|_S$ denotes restriction of mapping α to elements in set S. $\mathit{dom}(\alpha)$ is the domain of α.

[4] More details about actor language semantics can be found in [2], [3], and [158].

[5] $\mathit{fv}(e)$ is the set of free variables in the expression e.

the predicate must be true" (this is a recursive definition in that the path could begin from any configuration along the path). To better illustrate predicate fairness, let us assume a predicate ψ over actor configurations. At any configuration κ in a path, ψ is *enabled* if and only if there exists a finite sequence $[\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n]$ such that *if* the sequence is executed with $\kappa_0 = \kappa$, then $\psi(\kappa_n)$ will be true. Predicate fairness in FAM states that in a fair computation path, if the predicate ψ is infinitely often enabled, *i.e.*, infinitely often it is the case that a finite sequence $[\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n \mid \psi(\kappa_n) = \texttt{true}]$ can potentially happen, then eventually $\psi(\bar{\kappa})$ will be true for some configuration $\bar{\kappa}$. Now, any individual base-level transition can be considered a finite sequence consisting of a single transition. Therefore, predicate fairness entails the simpler fairness condition that *"a base-level transition that is infinitely often enabled will eventually happen"*.

We use predicate fairness in FAM in order to argue that the non-interruption condition described in Section 2.2.2, which is required for ensuring eventual consensus, can be guaranteed in a fair implementation. The condition follows from an assumption that the proposers will keep retrying with higher-numbered proposals if they fail to get their proposals accepted. If a nonfaulty proposer p is ready to propose some proposal number b at a configuration κ such that all prior proposals were lower than b, then there is at least one possible finite sequence $[\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n]_{P1,P2}$ (with $\kappa_0 = \kappa$) that can cause the non-interruption condition to be true at κ_n, thereby enabling the condition. A witness for $[\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n]_{P1,P2}$ is the sequence in which all proposers except p stop permanently, preventing any higher proposals in the future. Now, since proposal numbers are unique in Synod, if the nonfaulty proposers keep retrying, then always, eventually some nonfaulty proposer will become ready to propose the highest proposal number yet. This will cause our progress conditions to be infinitely often enabled in any path. Hence, by predicate fairness, our progress conditions will eventually be true, allowing some proposal number to eventually get accepted by a quorum of acceptors.

It should be noted that the fairness assumption of FAM applies only to the base-level transitions and not to the meta-level transitions. This is because actor failures and restarts in FAM can be arbitrary and it is not possible to control them programmatically in an implementation.

2.3.3 Some Interesting Properties of FAM

We present some interesting properties about the relationship of FAM to AM[6].

Theorem 2.1 *(FAM subsumes AM) Any valid computation path in AM is a valid path in FAM.*

Proof of **Theorem 2.1** -
Let $\pi = [\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < \infty]$ be a valid computation path in AM. Let $\mathbf{F}$ be a function from configurations in AM to configurations in FAM as follows:

$$\mathbf{F}(\langle\!\langle \alpha \;\parallel\; \mu \rangle\!\rangle) = \langle\!\langle \alpha \;\parallel\; \emptyset \;\parallel\; \mu \rangle\!\rangle$$

Then $\pi' = [\mathbf{F}(\kappa_i) \xrightarrow{l_i} \mathbf{F}(\kappa_{i+1}) \mid i < \infty]$ is a valid path in FAM by induction on computation sequences since **fun**, **new**, **snd**, and **rcv** in FAM are identical to AM transitions without modifying $\bar{\alpha}$, the map for failed actors, which remains empty throughout π'.
□

Theorem 2.2 *(AM can simulate finite paths in FAM) If all the* **stp** *and* **bgn** *transitions are removed from a finite computation path in FAM, then the resulting path will correspond to a valid path in AM.*

Proof of **Theorem 2.2** -
Let $\pi = [\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n]$ be a valid computation path in FAM. Let $\mathbf{G}$ be a function from configurations in FAM to configurations in AM as follows:
$\mathbf{G}(\langle\!\langle \alpha \;\parallel\; \bar{\alpha} \;\parallel\; \mu \rangle\!\rangle) = \langle\!\langle \alpha \cup \bar{\alpha} \;\parallel\; \mu \rangle\!\rangle = \langle\!\langle \alpha' \;\parallel\; \mu \rangle\!\rangle$
Given π, we construct a finite path π' in AM by removing all the **stp** and **bgn** transitions from π. This is a valid path by structural induction on transitions. To show this, let $\kappa_i \xrightarrow{l_i} \kappa_{i+1}$ be a transition in π and let $\kappa_i \xrightarrow{l_i} \kappa_{i+1} = \langle\!\langle \alpha_i \;\parallel\; \bar{\alpha}_i \;\parallel\; \mu_i \rangle\!\rangle \xrightarrow{l_i} \langle\!\langle \alpha_{i+1} \;\parallel\; \bar{\alpha}_{i+1} \;\parallel\; \mu_{i+1} \rangle\!\rangle$. Then there can be two cases:
Case 1: l_i is of the form $[\mathbf{fun} : a]$, $[\mathbf{new} : a, a']$, $[\mathbf{snd} : a]$, and $[\mathbf{rcv} : a, v]$ in FAM. Then there exists a transition l'_j in π' such that $\kappa'_j \xrightarrow{l'_j} \kappa'_{j+1}$ where $\kappa'_j = \mathbf{G}(\kappa_i)$, $l'_j = l_i$, and $\kappa'_{j+1} = \mathbf{G}(\kappa_{i+1})$ since $\langle\!\langle \alpha'_j \;\parallel\; \mu'_j \rangle\!\rangle \xrightarrow{l'_j} \langle\!\langle \alpha'_{j+1} \;\parallel\; \mu'_{j+1} \rangle\!\rangle$ is valid in AM because—(i) a, the *actor in focus* is in $\alpha'_j = \alpha_i \cup \bar{\alpha}_i$, since it is in α_i; (ii) $\alpha'_{j+1}(a) = \alpha_{i+1}(a)$; and (iii) $\mu'_{j+1} = \mu_{i+1}$.
Case 2: l_i is of the form $[\mathbf{stp} : a]$, $[\mathbf{bgn} : a]$ in FAM. Then we can remove the transition in

[6]The mechanical verification of these properties is beyond the scope of this work.

π' since $\mathbf{G}(\kappa_i) = \mathbf{G}(\kappa_{i+1})$ because $\alpha, [e]_a \cup \bar{\alpha} = \alpha|_{dom(\alpha)-\{a\}} \cup \bar{\alpha}, [e]_a$ for the **stp** transition and $\alpha \cup \bar{\alpha}, [e]_a = \alpha, [e]_a \cup \bar{\alpha}|_{dom(\bar{\alpha})-\{a\}}$ for the **bgn** transition.

□

Theorem 2.2 cannot be claimed for infinite paths in FAM. *E.g.,* consider a path in FAM in which an actor fails permanently. Then all future transitions that are enabled for the actor will never happen. If the **stp** transition is removed from this path, then it will be an unfair path in AM.

2.4 Formal Verification of Eventual Progress in Synod

This section presents our mechanically-verified proof of eventual progress in the Synod protocol under the conditions described in Section 2.2.2. The behavior of the Synod agents and all necessary assumptions have been formally expressed using FAM. The logical notations used in this section have been introduced in Table 2.1 and Table 2.2.

Table 2.1: Set symbols used in our formal specification of Synod.

Symbol	Description	Symbol	Description
$\mathcal{A}$	Set of all actors	$\mathcal{M}$	Set of all messages
$\mathbb{P}$	Set of all proposer actors	$\mathbb{A}$	Set of all acceptor actors
$\mathcal{V}$	Set of all values	$\mathcal{Q}$	Set of all quorums
$\mathcal{B}$	Set of all proposal numbers	$\mathbb{M}$	Set of all sets of messages
$\mathcal{C}$	Set of all actor configurations	$\mathcal{S}$	Set of all transition steps
$\mathcal{I}$	Set of all finite computation sequences	$\mathcal{P}$	Set of all predicates over $\mathcal{C}$
$\mathcal{T}$	Set of all fair transition paths	$\mathbb{N}$	Set of all natural numbers

A message is represented by a tuple $\langle \mathtt{s}{\in}\mathcal{A}, \mathtt{r}{\in}\mathcal{A}, \mathtt{k}{\in}\xi, \mathtt{b}{\in}\mathcal{B}, \mathtt{v}{\in}\mathcal{V}\rangle$ where $\xi = \{1a, 1b, 2a, 2b\}$, `s` is the sender, `r` is the receiver, `k` is the type of message, `b` is a proposal number, and `v` is a value. $\bar{v} \in \mathcal{V}$ is a null value constant used in $1a$ (*prepare*) messages.

The local state of an actor x can be extracted from a configuration κ as a tuple $\langle \eta^x_\kappa{\in}\mathbb{M}, \beta^x_\kappa{\in}\mathcal{B}, v^x_\kappa{\in}\mathcal{V}\rangle$ where η^x_κ is the set of messages received but not yet responded to, β^x_κ is the highest proposal number seen by an acceptor, and v^x_κ is the value corresponding to the highest proposal number accepted by an acceptor.

Transition paths represent the dynamic changes to actor configurations as a result of transition steps [127] and have been used to model computation paths. *Indexed positions* in transition paths correspond to logical steps in time and are used to express eventuality. κ^T_i represents the actor configuration at an indexed point i in a path T.

Table 2.2: Relation symbols used in our formal specification of Synod.

Symbol	Description	Input	Output
ς	Get last configuration	$\mathcal{T}$	$\mathcal{C}$
ρ	Get transition path up to index	$\mathcal{T} \times \mathbb{N}$	$\mathcal{T}$
T	Transition path constructor	$\mathcal{T} \times \mathcal{S}$	$\mathcal{T}$
σ	Choose a value to propose based on configuration	$\mathcal{C} \times \mathcal{A}$	$\mathcal{V}$
$\mathfrak{s}$	Construct a `snd` transition step	$\mathcal{A} \times \mathcal{M}$	$\mathcal{S}$
$\mathfrak{r}$	Construct a `rcv` transition step	$\mathcal{A} \times \mathcal{M}$	$\mathcal{S}$
$\mathfrak{m}$	Get set of messages "en route"	$\mathcal{C}$	$\mathbb{M}$
$\mathfrak{a}$	Actor is in α	$\mathcal{C} \times \mathcal{A}$	`Bool`
$\Re$	Actor is ready for a step	$\mathcal{T} \times \mathcal{A} \times \mathcal{S}$	`Bool`
ϕ	Proposer has promises from a quorum	$\mathbb{P} \times \mathcal{B} \times \mathcal{Q} \times \mathcal{C}$	`Bool`
Φ	Proposer has votes from a quorum	$\mathbb{P} \times \mathcal{B} \times \mathcal{Q} \times \mathcal{C}$	`Bool`
đ	Actor is nonfaulty	$\mathcal{A}$	`Bool`
þ	Proposal number satisfies *P1* and *P2*	$\mathcal{B}$	`Bool`
Ł	A proposer has learned of successful consensus	$\mathbb{P} \times \mathcal{B} \times \mathcal{C}$	`Bool`
e	A predicate is enabled at a configuration	$\mathcal{C} \times \mathcal{P}$	`Bool`
λ	A sequence happens at a configuration	$\mathcal{C} \times \mathcal{I}$	`Bool`
φ	Nonfaulty proposer ready to propose the highest proposal number yet to nonfaulty quorum	$\mathcal{C} \times \mathbb{P} \times \mathcal{Q}$	`Bool`
ψ	Some nonfaulty proposer is ready to propose a proposal number that will not be interrupted	$\mathcal{C}$	`Bool`

$\mathbb{P} \subset \mathcal{A}$ and $\mathbb{A} \subset \mathcal{A}$ are the sets of proposers and acceptors respectively and a quorum is a possibly equal non-empty subset of $\mathbb{A}$, *i.e.*, $\forall Q \in \mathcal{Q} : Q \subseteq \mathbb{A} \ \land \ Q \neq \emptyset$.

2.4.1 Fairness Assumptions for Actor Transitions

We assume two fairness axioms for the **snd** and **rcv** transitions that follow from the fairness assumptions of FAM. The `F-Snd-axm` and the `F-Rcv-axm` state that if a **snd** or **rcv** transition is enabled at some time, it must either eventually happen or eventually, it must become permanently disabled.

`F-Snd-Axm` $\equiv$

$\forall x \in \mathcal{A}, m \in \mathcal{M}, T \in \mathcal{T}, i \in \mathbb{N} : (\Re(\rho(T,i), x, \mathfrak{s}(x,m)) \implies$

$\quad((\exists j \in \mathbb{N} : (j \geq i) \ \land \ \rho(T, j+1) = \mathsf{T}(\rho(T,j), \mathfrak{s}(x,m)))$

$\quad\lor \ (\exists k \in \mathbb{N} : (k > i) \ \land \ (\forall j \in \mathbb{N} : (j \geq k) \implies \neg\Re(\rho(T,j), x, \mathfrak{s}(x,m))))))$

`F-Rcv-Axm` $\equiv$

$\forall x \in \mathcal{A}, m \in \mathcal{M}, T \in \mathcal{T}, i \in \mathbb{N} :$

$((m \in \mathfrak{m}(\varsigma(\rho(T,i))) \ \land \ \Re(\rho(T,i), x, \mathfrak{r}(x,m))) \implies$

$$((\exists j \in \mathbb{N} : (j \geq i) \ \wedge \ \rho(T, j+1) = \intercal(\rho(T, j), \mathfrak{r}(x, m)))$$
$$\vee \ (\exists k \in \mathbb{N} : (k > i) \ \wedge \ (\forall j \in \mathbb{N} : (j \geq k) \implies \neg(m \in \mathfrak{m}(\varsigma(\rho(T, j))) \ \wedge \ \Re(\rho(T, j), x, \mathfrak{r}(x, m)))))))$$

2.4.2 Rules Specifying the Actions of Synod Actors

The Synod protocol is presented in [100] as a high-level abstraction of the behavior of the agents while leaving out the implementation details to the discretion of the system developers [131]. We specify rules over actor local states that dictate if an available Synod actor should become ready to send a message. Since Synod does not specify *when* a proposer should send a *prepare* message, we leave that behavior unspecified. For the proof of eventual progress under the conditions identified in Section 2.2.2, we will assume that eventually, a nonfaulty proposer will be ready to send *prepare* messages to a quorum of nonfaulty acceptors.

`Snd-1b-Rul` $\equiv$
$\forall a : \mathbb{A}, p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B} :$
$(\langle p, a, 1a, b, \bar{v} \rangle \in \eta^{a}_{\varsigma(\rho(T,i))} \ \wedge \ \beta^{a}_{\varsigma(\rho(T,i))} < b \ \ \wedge \ \mathfrak{a}(\varsigma(\rho(T, i)), a)) \implies \Re(\rho(T, i), a, \mathfrak{s}(a, \langle a, p, 1b, b, v^{a}_{\varsigma(\rho(T,i))} \rangle)))$

`Snd-2a-Rul` $\equiv$
$\forall p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B}, Q \in \mathcal{Q} :$
$(\phi(p, b, Q, \varsigma(\rho(T, i))) \ \wedge \ \mathfrak{a}(\varsigma(\rho(T, i)), p)) \implies (\forall a \in Q : \Re(\rho(T, i), p, \mathfrak{s}(p, \langle p, a, 2a, b, \sigma(\varsigma(\rho(T, i)), p) \rangle))))$

`Snd-2b-Rul` $\equiv$
$\forall a : \mathbb{A}, p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B}, v \in \mathcal{V} :$
$(\langle p, a, 2a, b, v \rangle \in \eta^{a}_{\varsigma(\rho(T,i))} \ \wedge \ \beta^{a}_{\varsigma(\rho(T,i))} \leq b \ \ \wedge \ \mathfrak{a}(\varsigma(\rho(T, i)), a))) \implies \Re(\rho(T, i), a, \mathfrak{s}(a, \langle a, p, 2b, b, v \rangle))$

Since the response to a message in Synod is a finite set of actions, if there is a message in the multi-set for a Synod actor and the actor is also available, then a receive transition is enabled.

`Rcv-Rul` $\equiv$
$\forall s, r \in \mathcal{A}, T \in \mathcal{T}, i \in \mathbb{N}, k \in \xi, b \in \mathcal{B}, v \in \mathcal{V} :$
$(\langle s, r, k, b, v \rangle \in \mathfrak{m}(\varsigma(\rho(T, i)) \ \wedge \ \mathfrak{a}(\varsigma(\rho(T, i)), r))) \implies \Re(\rho(T, i), r, \mathfrak{r}(r, \langle s, r, k, b, v \rangle))$

2.4.3 Assumptions About the Future Behavior of Agents

To prove progress in Synod, we borrow some assumptions about the future behavior of nonfaulty agents used by Lamport for informally proving progress in Paxos [104]. It is worth noting that being nonfaulty does not prohibit an agent from temporarily failing. It simply means that for every action that needs to be performed by the agent, eventually the agent is available to perform the action and the action happens. In FAM, a nonfaulty actor can be modeled by asserting that an enabled **snd** or **rcv** transition for the actor will eventually happen. However, given the `F-Snd-Axm` and `F-Rcv-Axm` axioms, it suffices to assume that for a nonfaulty actor, if a **snd** or **rcv** transition is enabled, then it will either eventually occur or it will be infinitely often enabled. As FAM does not model message loss, any message in the multi-set will persist until it is received.

We introduce a predicate đ to specify an actor as nonfaulty, such that:

- $\text{đ}(x)$ implies that x will be eventually available if there is a message "en route" that x needs to receive.
- $\text{đ}(x)$ implies that x will be eventually available if x's local state dictates that it needs to send a message.
- $\text{đ}(x)$ implies that if a **snd** or **rcv** transition is enabled for x, then it will either eventually occur or it will be infinitely often enabled.

`Prp-NF-Axm` $\equiv$

$$\begin{aligned}
&\forall p \in \mathbb{P} : \text{đ}(p) \implies \\
&\qquad (\forall b \in \mathcal{B}, k \in \xi, v \in \mathcal{V}, T \in \mathcal{T}, i \in \mathbb{N}, a \in \mathbb{A}, Q \in \mathcal{Q} : \\
&\qquad\qquad (\phi(p, b, Q, \varsigma(\rho(T,i))) \ \vee \ \langle a, p, k, b, v\rangle \in \mathfrak{m}(\varsigma(\rho(T,i)))) \implies \\
&\qquad\qquad\qquad (\mathfrak{a}(\varsigma(\rho(T,i)), p) \ \vee \ (\exists j \in \mathbb{N} : (j > i) \ \wedge \ \mathfrak{a}(\varsigma(\rho(T,j)), p))))
\end{aligned}$$

`Acc-NF-Axm` $\equiv$

$$\begin{aligned}
&\forall a \in \mathbb{A} : \text{đ}(a) \implies \\
&\qquad (\forall p \in \mathbb{P}, k \in \xi, v \in \mathcal{V}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B} : \\
&\qquad\qquad (((\langle p, a, 1a, b, v\rangle \in \eta^a_{\varsigma(\rho(T,i))} \ \wedge \ (\beta^a_{\varsigma(\rho(T,i))} < b)) \\
&\qquad\qquad \vee \ (\langle p, a, 2a, b, v\rangle \in \eta^a_{\varsigma(\rho(T,i))} \ \wedge \ (\beta^a_{\varsigma(\rho(T,i))} \le b)) \\
&\qquad\qquad \vee \ (\langle p, a, k, b, v\rangle \in \mathfrak{m}(\varsigma(\rho(T,i))))) \implies \\
&\qquad\qquad\qquad (\mathfrak{a}(\varsigma(\rho(T,i)), a) \ \vee \ (\exists j \in \mathbb{N} : (j > i) \ \wedge \ \mathfrak{a}(\varsigma(\rho(T,j)), a))))
\end{aligned}$$

`NF-IOE-Axm` $\equiv$

$$\begin{aligned}
&\forall x \in \mathcal{A} : \text{đ}(x) \implies \\
&(\forall T \in \mathcal{T}, i \in \mathbb{N}, m \in \mathcal{M} : \\
&\qquad ((m \in \mathfrak{m}(\varsigma(\rho(T,i))) \ \wedge \ \Re(\rho(T,i), x, \mathfrak{r}(x,m))) \implies \\
&\qquad\qquad ((\exists j \in \mathbb{N} : (j \ge i) \ \wedge \ \rho(T, j+1) = \mathtt{T}(\rho(T,j), \mathfrak{r}(x,m))) \\
&\qquad\qquad \vee \ (\forall k \in \mathbb{N} : (k > i) \implies
\end{aligned}$$

$$(\exists j \in \mathbb{N} : (j \geq k) \;\wedge\; (m \in \mathfrak{m}(\varsigma(\rho(T, j))) \;\wedge\; \Re(\rho(T, j), x, \mathfrak{r}(x, m)))))))$$
$$\wedge\; (\Re(\rho(T, i), x, \mathfrak{s}(x, m)) \implies$$
$$((\exists j \in \mathbb{N} : (j \geq i) \;\wedge\; \rho(T, j+1) = \mathsf{T}(\rho(T, j), \mathfrak{s}(x, m)))$$
$$\vee\; (\forall k \in \mathbb{N} : (k > i) \implies$$
$$(\exists j \in \mathbb{N} : (j \geq k) \;\wedge\; \Re(\rho(T, j), x, \mathfrak{s}(x, m))))))$$

We then introduce a predicate þ that is true of a proposal number if and only if it satisfies the conditions P1 and P2 described in Section 2.2.2.

$\texttt{P1-P2-Def} \equiv$
$\forall b \in \mathcal{B} : þ(b) \iff$
$$(\forall p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, v \in \mathcal{V}, a \in \mathbb{A} :$$
$$((\langle p, a, 1a, b, \bar{v}\rangle \in \eta^{a}_{\varsigma(\rho(T,i))} \implies \beta^{a}_{\varsigma(\rho(T,i))} < b)$$
$$\wedge\; (\langle p, a, 2a, b, v\rangle \in \eta^{a}_{\varsigma(\rho(T,i))} \implies \beta^{a}_{\varsigma(\rho(T,i))} \leq b)))$$

Finally, our conditions for formally guaranteeing progress state that – *in all fair transition paths, some nonfaulty proposer p will be eventually ready to propose some proposal number b, that will satisfy P1 and P2, to some quorum Q whose members are all nonfaulty.*

$\texttt{CND} \equiv$
$\forall T \in \mathcal{T} :$
$(\exists i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} : (đ(p) \;\wedge\; þ(b) \;\wedge\; (\forall a \in Q : (đ(a) \;\wedge\; \Re(\rho(T, i), p, \mathfrak{s}(p, \langle p, a, 1a, b, \bar{v}\rangle)))))))$

2.4.4 The Proof of Eventual Progress

To prove progress in Synod, it suffices to prove that eventually, at least one proposal number will be chosen by some quorum (Section 2.2). In our conditions `CND`, we have assumed that some proposer p will eventually propose a proposal number b, that will satisfy *P1* and *P2*, to a quorum Q. Our proof strategy is to show that eventually, p will learn that b has been chosen by all members of Q. Theorem 2.3 formally states our main progress guarantee while Lemma 2.1 and Lemma 2.2 state progress in Phase 1 and Phase 2 respectively.

Theorem 2.3 *(Eventual progress in Synod) Given* `CND`*, in all fair transition paths, eventually some proposer p will learn that some proposal number b has been chosen.*

$\texttt{Theorem-2.3} \equiv$
$\texttt{CND} \implies (\forall T \in \mathcal{T} : (\exists i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B} : Ł(p, b, \varsigma(\rho(T, i)))))$

Lemma 2.1 *In a fair transition path, if eventually a nonfaulty proposer p becomes ready to propose a proposal number b, that satisfies P1 and P2, to a quorum Q whose members are all nonfaulty, then eventually p will receive promises from all members of Q for b.*

$$\texttt{Lemma-2.1} \equiv$$
$$\forall T \in \mathcal{T}, i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} :$$
$$(\text{đ}(p) \;\wedge\; \text{þ}(b) \wedge\; (\forall a \in Q : (\text{đ}(a) \;\wedge\; \Re(\rho(T,i), p, \mathfrak{s}(p, \langle p, a, 1a, b, \bar{v}\rangle))))) \implies$$
$$(\exists j \in \mathbb{N} : (j \geq i) \;\wedge\; \phi(p, b, Q, \varsigma(\rho(T,j))))$$

Lemma 2.2 *In a fair transition path, if eventually a nonfaulty proposer p receives promises for a proposal number b, that satisfies P1 and P2, from a quorum Q whose members are all nonfaulty, then eventually p will receive votes from all members of Q for b.*

$$\texttt{Lemma-2.2} \equiv$$
$$\forall T \in \mathcal{T}, i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} :$$
$$(\text{đ}(p) \;\wedge\; \text{þ}(b) \;\wedge\; \phi(p, b, Q, \varsigma(\rho(T,i))) \;\wedge\; (\forall a \in Q : \text{đ}(a))) \implies$$
$$(\exists j \in \mathbb{N} : (j \geq i) \;\wedge\; \Phi(p, b, Q, \varsigma(\rho(T,j))))$$

Given below are the proof sketches of Theorem 2.3, Lemma 2.1, and Lemma 2.2:

Proof of **Theorem 2.3** -

(1) By Lemma 2.1 and `CND`, some nonfaulty proposer p will eventually receive promises from some quorum Q, whose members are all nonfaulty, for some proposal number b that satisfies *P1* and *P2*.

(2) By Lemma 2.2, p will eventually receive votes from Q for b and learn that b has been chosen.

□

Proof of **Lemma 2.1** -

(1) By `Prp-NF-Axm`, `NF-IOE-Axm`, and `F-Snd-Axm` *prepare* messages from p will eventually be sent to all members of Q.

(2) By `Acc-NF-Axm`, `NF-IOE-Axm`, `F-Rcv-Axm` all members of Q will eventually receive the *prepare* messages.

(3) By `P1-P2-Def`, `Snd-1b-Rul`, and `Acc-NF-Axm`, each member of Q will eventually be ready to send *promise* messages to p.

(4) By `Acc-NF-Axm`, `NF-IOE-Axm`, and `F-Snd-Axm` the *promise* messages from each member of Q will eventually be sent.

(5) By `Prp-NF-Axm`, `F-Rcv-Axm`, and `NF-IOE-Axm` p will eventually receive the *promise* messages from all members of Q.
$\square$

Proof of **Lemma 2.2** -

(1) By `Snd-2a-Rul`, and `Prp-NF-Axm`, p will eventually be ready to send *accept* messages to all members of Q with proposal number b.

(2) By `Prp-NF-Axm`, `NF-IOE-Axm`, and `F-Snd-Axm`, *accept* messages from p will eventually be sent to all members of Q.

(3) By `Acc-NF-Axm`, `NF-IOE-Axm`, `F-Rcv-Axm` all members of Q will eventually receive the *accept* messages.

(4) By `P1-P2-Def`, `Snd-2b-Rul`, and `Acc-NF-Axm`, each member of Q will eventually be ready to send *voted* messages to p.

(5) By `Acc-NF-Axm`, `NF-IOE-Axm`, and `F-Snd-Axm` the *voted* messages from each member of Q will eventually be sent

(6) By `Prp-NF-Axm`, `F-Rcv-Axm`, and `NF-IOE-Axm` p will eventually receive the *voted* messages from all members of Q.
$\square$

2.4.5 The Proof of Eventual Progress Under Predicate Fairness

Our fundamental conditions for progress in Synod, which are specified as `CND` in Section 2.4.3, describe a behavior of the system by which in all paths, eventually, some nonfaulty proposer becomes ready to propose some proposal number that will not be interrupted, to a quorum of nonfaulty acceptors. In this section, we formally show that under the assumption of predicate fairness in FAM, it is possible to theoretically guarantee that this behavior will be eventually true in a set of actors implementing Synod.

In order to formalize the specification of predicate fairness in FAM, we first define what it means for a predicate P over actor configurations to be *enabled* at a configuration. `Enabled-Def` states that in a path T, a predicate P is enabled at a configuration κ_i^T if and only if there exists a sequence I such that *if* I happens at κ_i^T, then P will be true in the consequent configuration κ_j^T[7].

`Enabled-Def ≡`
$$\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C}, P \in \mathcal{P} : e(\kappa_i^T, P) \iff (\exists I \in \mathcal{I} : \lambda(\kappa_i^T, I) \implies (\exists \kappa_j^T \in \mathcal{C} : [\kappa_i^T \overset{I}{\dashrightarrow} \kappa_j^T] \wedge P(\kappa_j^T)))$$

Using the enabling condition, we can now formally specify predicate fairness for FAM as the statement `F-Predicate-Axm`.

`F-Predicate-Axm ≡`
$$\begin{aligned} &\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C}, P \in \mathcal{P} : e(\kappa_i^T, P) \implies \\ &\qquad (\exists \kappa_j^T \in \mathcal{C} : (j \geq i) \wedge P(\kappa_j^T)) \\ &\qquad \vee (\exists \kappa_k^T \in \mathcal{C} : (k > i) \wedge (\forall \kappa_j^T \in \mathcal{C} : (j \geq k) \implies \neg e(\kappa_j^T, P))) \end{aligned}$$

To show that `CND` will be satisfied under predicate fairness, let us define a Synod-specific predicate ψ over configurations such that $\psi(\kappa)$ is true if and only if at κ, some nonfaulty proposer is ready to propose a proposal number, that will satisfy the non-interruption condition, to some nonfaulty quorum (`Synod-Pred`). Clearly, if ψ is eventually satisfied in all paths, the conditions `CND` will be satisfied. Using predicate fairness and the behavior of the proposers, we can show that ψ will be eventually true in all fair paths.

`Synod-Pred ≡`
$$\begin{aligned} &\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C} : \\ &\quad (\psi(\kappa_i^T) \iff \\ &\qquad (\exists p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} : (\text{đ}(p) \ \wedge \ \text{þ}(b) \\ &\qquad \wedge \ (\forall a \in Q : (\text{đ}(a) \ \wedge \ \Re(\kappa_i^T, p, \mathfrak{s}(p, \langle p, a, 1a, b, \bar{v}\rangle))))))) \end{aligned}$$

As previously mentioned in Section 2.3.2, we assume that the proposers can retry with higher-numbered proposals if their proposals are not chosen. Since proposal numbers are unique in Synod, no two proposals can be equal. Therefore, if all nonfaulty proposers retry, then always, eventually, some nonfaulty proposer will become ready to propose the highest proposal number yet to a nonfaulty quorum. We formalize this property as `Prp-Retry`, which states that at any configuration κ_i^T, if a nonfaulty proposer p becomes ready to propose the

[7]The symbols $e(\kappa_i^T, P)$, $\lambda(\kappa_i^T, I)$, and $[\kappa_i^T \overset{I}{\dashrightarrow} \kappa_j^T]$ denote that P is enabled at κ_i^T, I happens at κ_i^T, and κ_j^T is a consequence of I happening at κ_i^T respectively.

highest proposal number yet proposed to a nonfaulty quorum[8], then either ψ will be satisfied at some future configuration κ_j^T, or always, eventually, some nonfaulty proposer will become ready to propose the highest proposal number yet proposed to a nonfaulty quorum.

```
Prp-Retry ≡
```
$$\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C} : (\exists p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_i^T, p, Q)) \implies$$
$$((\exists \kappa_j^T \in \mathcal{C} : (j \geq i) \wedge \psi(\kappa_j^T))$$
$$\vee(\forall \kappa_k^T \in \mathcal{C} : (k > i) \implies (\exists \kappa_j^T \in \mathcal{C} : (j \geq k) \wedge (\exists p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_j^T, p, Q)))))$$

Now, at any configuration κ_i^T, if a nonfaulty proposer p becomes ready to propose the highest proposal number yet, then there is a possible witness sequence $\hat{I}$ such that if $\hat{I}$ happens at κ_i^T, then ψ will be true at the consequent configuration κ_j^T. This sequence $\hat{I}$ is the sequence in which all proposers except p fail permanently, preventing any future proposals. Therefore, we can state that if $\varphi(\kappa_i^T, p, Q)$ is `true` at any configuration κ_i^T, then if $\hat{I}$ happens at κ_i^T, then $\psi(\kappa_j^T)$ will be true at the consequent configuration κ_j^T (`Wit-Seq`).

```
Wit-Seq ≡
```
$$\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C} :$$
$$(\exists p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_i^T, p, Q)) \implies$$
$$(\lambda(\kappa_i^T, \hat{I}) \implies (\exists \kappa_j^T \in \mathcal{C} : [\kappa_i^T \overset{\hat{I}}{\dashrightarrow} \kappa_j^T] \wedge \psi(\kappa_j^T)))$$

Finally, for consensus to happen in Synod, at least one proposal needs to be made by some nonfaulty proposer. Therefore, we assume that eventually, some nonfaulty proposer will become ready to propose the highest proposal number yet (`Some-Prp-Ready`). This will be trivially satisfied the first time some nonfaulty proposer becomes ready to initiate the highest-numbered proposal until that time.

```
Some-Prp-Ready ≡
```
$$\forall T \in \mathcal{T} : (\exists \kappa_i^T \in \mathcal{C}, p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_i^T, p, Q))$$

Theorem 2.4 states that eventually, the predicate ψ will be satisfied in all fair paths, which is, by definition, equivalent to the conditions `CND`.

Theorem 2.4 *(CND follows from predicate fairness) In all fair transition paths, eventually, the predicate ψ will be true at some configuration.*

`Theorem-2.4` $\equiv \forall T \in \mathcal{T} : \exists \kappa_i^T \in \mathcal{C} : \psi(\kappa_i^T)$

[8]The symbol $\varphi(\kappa_i^T, p, Q)$ denotes that p is ready to propose the highest proposal number yet proposed to Q at κ_i^T.

Proof of **Theorem 2.4 -**

(1) By `Some-Prp-Ready`, `Wit-Seq`, and `Enabled-Def`, ψ will be enabled at least once.

(2) By `Prp-Retry`, ψ will be infinitely often enabled.

(3) By contradiction with `F-Predicate-Axm`, ψ will eventually be true.
$\square$

The proofs of Lemma 2.1, Lemma 2.2, Theorem 2.3, and Theorem 2.4 have been mechanically verified for correctness using Athena. We made use of Athena's interactive proof development environment where the high-level structures of the proofs were created manually in a hierarchical manner consisting of trivial well-connected steps. The SPASS [163] automatic theorem prover was then guided with appropriate premises for proving each step (see Appendix B for more details). The proof consists of around 7200 lines of Athena code[9].

2.5 Related Work

De Prisco *et al.* [41] present a rigorous hand-written proof of safety for Paxos along with an analysis of time performance and fault tolerance. Chand *et al.* [28] provide a specification of Multi-Paxos in TLA$^+$ [102] and use TLAPS [33] to prove its safety. Padon *et al.* [133] have verified the safety property for Paxos, *Vertical Paxos* [105], *Fast Paxos* [104], and *Stoppable Paxos* [114] using deductive verification. Küfner *et al.* [96] provide a methodology to develop machine-checkable proofs of fault-tolerant round-based distributed systems and verify the safety property for Paxos. Schiper *et al.* [152] have formally verified the safety property of a Paxos-based totally ordered broadcast protocol using EventML [18] and the Nuprl [129] proof assistant. Howard *et al.* [86] have presented *Flexible Paxos* by introducing flexible quorums for Paxos and have model checked its safety property using the TLC model checker [103]. Rahli *et al.* [148, 149] have used EventML and Nuprl to formally verify the safety of an implementation of Multi-Paxos. Attiya *et al.* [10] provide bounds on the time to reach progress in consensus by assuming a synchronous model with known bounds on message delivery and processing time of nonfaulty processes. Keidar *et al.* [91] consider a partial synchrony model with known bounds on processing times and message delays and use it to

[9]Available at https://wcl.cs.rpi.edu/assure

guarantee progress in a consensus algorithm when the bounds hold. Malkhi *et al.* [114] introduce Stoppable Paxos, a variant of Paxos for implementing a stoppable state machine and provide an informal proof of safety and progress for Stoppable Paxos with a unique leader. McMillan *et al.* [115] machine-check and verify the proofs of safety and progress properties of Stoppable Paxos [114] using Ivy [134]. Dragoi *et al.* [47] introduce PSync, a language that allows writing, execution, and verification of high-level implementations of fault-tolerant systems, and use it to verify the safety and progress properties of *LastVoting* [32]. LastVoting is an adaptation of Paxos in the *Heard-Of model* [32] that guarantees progress under the assumption of a single leader. A machine-checked proof of safety and progress of LastVoting also appeared in [42]. Hawblitzel *et al.* [82,83] introduce a framework for designing provably correct distributed algorithms called IronFleet and use it to prove the safety and progress properties for a Multi-Paxos implementation called IronRSL by embedding TLA$^+$ specifications in Dafny [108]. Their proof of progress relies on the assumption that eventually, all messages will arrive within a maximum network delay and leader election will succeed. All of the aforementioned work has either analyzed the safety property or both the safety and progress properties of Synod-related protocols. Where progress has been verified, the authors have either assumed a unique leader, synchrony, or both.

Field and Varela [59] present *transactors*, a fault-tolerant programming model for composing loosely-coupled distributed components running in an unreliable environment like the IoP. Transactors are based on the actor model and can ensure consistency of distributed state in the presence of some specific types of node and network failures, and message loss. Bocchi *et al.* [20] present a behavioral typing system for higher-ordered timed calculus using *session types.* Their type system can check processes against protocols that are not *wait-free*, *i.e.*, the time windows for corresponding send and receive actions intersect. Charalambides *et al.* [30] introduce a novel session type system called System-A that can be used for the static verification of a large class of asynchronous distributed protocols. Duan *et al.* [48] introduce *NFVactor*, a system for *network function virtualization*, that is based on the actor model. NFVactor provides advantages such as lightweight failure resilience, high-performance flow migration, and transparent resilience. Bainomugisha *et al.* [11] propose an actor-based service partitioning model called the *resilient actor model.* The approach is based on the actor model and involves decomposing the functionality of an application into a set of resilient actors at runtime, which are then moved to devices where they can run in a distributed manner. The process is user-guided and the resulting partitioned application is retractable and resilient to network failures. Charalambides *et al.* [31] introduce a simple actor language in which programmers can specify demands for certain types of messages to be generated during execution. They regard progress as the question of whether an actor eventually receives all messages included in its *typestate* and show that their type system guarantees that in all executions of well-typed programs, all requirements generated at runtime will result in the corresponding messages being received. Musser and Varela [127] present a formalization of the actor model in Athena and show that many properties of actor computation can be expressed and proven at an abstract level, independently of the details of a particular system of actors. They develop theorems concerning the persistence of actors and messages, preservation of unique actor identifiers, monotonicity properties of actor local states, guaranteed message delivery, and general consequences of fairness.

Our work improves upon existing work by identifying fundamental asynchronous conditions under which the Synod consensus protocol can make eventual progress in the absence of a distinguished proposer. It introduces a failure-aware actor model that can be used for modeling temporary or permanent failures in aircraft and reasoning about distributed protocols like Synod in uncertain environments like the IoP. We have also presented the first machine-checked proof of eventual progress in Synod using Athena.

2.6 Chapter Summary

In this chapter, we have identified a set of sufficient conditions under which the Synod protocol can make progress, in asynchronous communication settings and in the absence of a unique leader. Leader election itself being a consensus problem, our conditions generalize Paxos' progress conditions by eliminating their cyclic reliance on consensus. Consequently, our weaker assumptions do not impose a communication bottleneck or proposal restrictions. We have introduced a failure-aware actor model (FAM), that assumes predicate fairness, to reason about communication failures in actors. Using this reasoning framework we have formally demonstrated that eventual progress can be guaranteed in Synod under the identified conditions and have used Athena to develop the first machine-checked proof of eventual progress in Synod. Additionally, we have also shown with a mechanical proof that under predicate fairness in FAM, our conditions for eventual progress will be eventually satisfied.

A potential direction of future work would be to investigate mechanically verified guarantees of timely progress by using data-driven statistical results. Such properties may be provided by using statistical observations about message transmission and processing delays, which cannot be deterministically predicted in asynchronous conditions but can be observed at run-time. Another potential direction of work would be to model message loss in FAM by introducing additional meta-level transitions. This would allow us to further weaken our current conditions for eventual progress in Synod by requiring only a subset of messages to not be lost. To avoid livelocks, it may also suffice to replace predicate fairness with a weaker assumption that *infinitely often enabled finite transition sequences must eventually occur*. Finally, another important future direction would be to investigate how progress can be guaranteed in the presence of Byzantine agents.

CHAPTER 3
A DISTRIBUTED KNOWLEDGE PROPAGATION PROTOCOL

3.1 Chapter Introduction

Knowledge is the state of an agent being aware of a fact about the world. Several states of knowledge arise when analyzing the properties of knowledge relative to a group of agents. *E.g.*, in a group of agents, there may be the following possible states of knowledge about a fact ϕ — all agents know ϕ, some agents know ϕ, or no agent knows ϕ. Two important states of knowledge are *common knowledge* and *distributed knowledge* [54]. Common knowledge implies that *everyone knows ϕ, everyone knows that everyone knows ϕ, everyone knows that everyone knows that everyone knows ϕ*, and so on. Distributed knowledge, on the other hand, implies that the knowledge of ϕ is distributed among all members in a group, *i.e.*, together the members of the group know ϕ, even though it may be the case that no individual member knows ϕ by itself. In autonomous multi-agent systems, it is often useful to analyze the knowledge state of the system in order to determine if the group of agents or each individual agent has sufficient knowledge to satisfy certain properties or meet certain goals. *E.g.*, for an aircraft to operate in an airspace with a sense of safety, it must not only have knowledge of the traffic aircraft but also know that the traffic aircraft have knowledge of it. Therefore, the study of knowledge states is important for safety-critical multi-aircraft applications like Decentralized Admission Control where the absence of a centralized coordinator warrants high situational awareness among the aircraft to ensure safety.

There are several challenges to the analysis of knowledge states for multi-aircraft applications like DAC. Firstly, different applications may require different states of knowledge. *E.g.,* a message relaying application may only require that the recipient agent learns of the message while an emergency-advisory application may require that all traffic aircraft learn about an emergency aircraft. Therefore, it is important to identify the required state of knowledge for each application before analysis. Secondly, for different applications, appropriate knowledge propagation protocols must be developed that can be used to attain

Portions of this chapter previously appeared as: Paul, S., Patterson, S., Varela, C.A.: Collaborative situational awareness for conflict-aware flight planning. In: IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 1-10 (2020). doi:10.1109/dasc50938.2020.9256620 application-specific states of knowledge. Finally, for safety-critical applications, a knowledge propagation protocol must satisfy some correctness properties. In asynchronous networks like the Internet of Planes where agents can fail, it is difficult to formally guarantee such correctness properties unless some required assumptions are made about the system.

In this chapter, we formalize the definition of a *safe state of knowledge* for DAC using *knowledge logic* [54] and present a knowledge propagation protocol called the *Two-Phase Acknowledge Protocol* that can be used to achieve this safe state with respect to the new set of owners after a candidate has been admitted. The protocol allows an aircraft to propagate a fact among a group of aircraft and learn when the safe state of knowledge has been successfully achieved. We also identify certain system conditions under which it can be guaranteed that our protocol will satisfy two important correctness properties — safety and progress. We use these conditions to provide formal proofs of the correctness properties that have been mechanically verified using the TLA$^+$ Proof System.

The rest of the chapter is structured as follows: Section 3.2 identifies a safe state of knowledge required for DAC; Section 3.3 introduces our knowledge propagation protocol TAP; Section 3.4 presents the formal verification of the correctness properties of TAP; Section 3.5 presents related work on the formal analysis of knowledge states in distributed systems; and Section 3.6 concludes the chapter with a summary and discussion on potential future directions of work.

3.2 Identifying A Safe State of Knowledge for DAC

Successful knowledge propagation in DAC must ensure a *safe state of knowledge* that will allow all aircraft to have a sense of "safety" in the absence of a centralized coordinator. In knowledge logic [54], the expression $k_i\phi$ represents that *an agent i knows the fact ϕ* and $E_G\phi$ represents that *all agents in a group G know ϕ.* The expression $E_GE_G\phi$ (or $E_G^2\phi$) represents a higher state of knowledge than $E_G\phi$ and implies that every agent in G knows two facts – (1) ϕ, and (2) $E_G\phi$. In the context of DAC, let G represent the set of new owners after a new candidate has been admitted and ϕ represent the set of approved flight plans for the aircraft in G. Since there is no centralized coordinator in DAC, it is our belief that the absence of $E_G^2\phi$ will create a scenario in which the aircraft will not be able to trust the overall safety of their decentralized operations. Fagin *et al.* [54] 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.* Therefore, we define a safe state of knowledge for DAC as the state $E_G^2\phi$.

The problem statement for knowledge propagation in DAC can be stated as — *"There is a set of aircraft $K : K \subset G$ in which all aircraft know the same value ϕ and the membership of G, and each aircraft in K must try to propagate the knowledge of ϕ to attain $E_G^2\phi$ with respect to G and at least one aircraft in K should eventually learn that $E_G^2\phi$ has been attained."*

3.3 The Two-Phase Acknowledge Protocol (TAP)

In this section, we present the Two-Phase Acknowledge Protocol, a distributed knowledge propagation protocol that can be used for propagating a value ϕ to a group of distributed nodes to eventually achieve $E_G^2\phi$.

For TAP, we assume an asynchronous, non-Byzantine system model in which agents can fail and have stable storage that can tolerate failures. Messages can be duplicated, and have arbitrary transmission times, but cannot be corrupted. We also assume that messages cannot be lost. There is a non-empty set of *propagators*, and a logically separate non-empty set of *learners*. Each propagator has knowledge of the set of all learners. A single value ϕ is known to all propagators. $E_G^2\phi$, in the context of our protocol, implies that *all learners know that all other learners know ϕ*. The goal of every propagator is to propagate ϕ and eventually learn that $E_G^2\phi$ has been achieved. Propagators and learners are logical abstractions and may be functionally implemented by the same physical node (*e.g.* – an aircraft) simultaneously.

TAP operates in the following consecutive phases:

- **Phase 1**
  (a) A propagator sends a *learn ("1a")* message with a value ϕ to all learners.
  (b) A learner learns a value ϕ if and only if it receives a *learn* message with the value ϕ from a propagator and it replies back to the propagator with a *learnt ("1b")* message if and only if it has learnt a value ϕ.
- **Phase 2**

(a) A propagator sends an *all-know ("2a")* message to each learner if and only if it has received a *learnt* message from all learners.

(b) A learner learns that all learners know the value ϕ if and only if it receives an *all-know* message from a propagator and it replies back to the propagator with an *acknowledgment ("2b")* message if and only if it has learnt that all learners know the value ϕ.

(c) A propagator learns $E_G^2\phi$ if and only if it has received *acknowledgement* message from all learners.

3.3.1 Required Correctness Properties

For use in safety-critical aerospace applications like DAC, TAP must satisfy the following correctness properties:

- Safety - This property requires that a propagator can learn that $E_G^2\phi$ has been attained if and only if all learners know $E_G\phi$. This is important because it ensures that the safe state of knowledge will be correctly attained.
- Progress - This property requires that the protocol must ensure that eventually, $E_G^2\phi$ is attained. This is important for applications where an eventual outcome is necessary.

3.3.2 Some Important Observations About TAP

The system model of TAP is not sufficient to guarantee that the required correctness properties will be satisfied – *e.g.*, if even one learner fails, the protocol will never be able to make progress. Therefore, to guarantee the correctness properties, we need to make certain additional assumptions about the system. We list some important observations that will be helpful for identifying such assumptions.

- All the propagators try to commit the same value, removing the competition for votes. Therefore, a learner can respond to *"1b"* and *"2b"* messages from multiple propagators, even if they have already committed a value.
- The non-Byzantine behavior of available agents directs that some sets of operations are atomic in nature – *e.g.*, a propagator has to send *"2a"* messages if it has received *"1b"* messages from all learners and it has to receive *"1b"* messages from all learners to send *"2a"* messages.

- If an agent receives messages that had not been sent, then its non-Byzantine behavior cannot ensure correctness – *e.g.*, if a learner receives *"2a"* messages that had not been sent, it will incorrectly learn that all learners know a value and respond with *"2b"* messages, affecting safety.

- Since successful propagation requires a value to be replicated in all learners, the protocol cannot tolerate the permanent failure of even one learner.

- Since successful propagation requires a propagator to learn about $E_G^2\phi$, at least one propagator must eventually learn about $E_G^2\phi$.

We will use the above observations to identify some assumptions that we will then use to prove the correctness guarantees for TAP.

3.3.3 Conditions for Guaranteeing Correctness

The following general condition about the availability of agents can be useful for proving the correctness of most distributed protocols under asynchronous settings:

G1 *All agents are always eventually available.*

In asynchronous communication, message transmission and processing delays can be arbitrarily long. This makes it impossible to differentiate temporary agent failures from message processing delays. As a result, stating that an agent is *"always eventually available"* is equivalent to stating that the agent is *"always available"*[10], since temporary agent failures can be subsumed under processing delays. Therefore, in our analysis of TAP, we assume a simple model of asynchronous communication where temporary agent failures cannot be detected in isolation from message delays. Under this model, we can identify the following correctness requirements:

G1a *At least one propagator is always available.*

G1b *All learners are always available.*

Following are some important observations about the identified conditions:

[10]"always" here *only* implies a period of time that is sufficient to ensure that the protocol will make progress. It does not imply that the agent must be available "endlessly", even after the protocol has terminated.

- For the protocol to make progress, there must be some propagator to send *"1a"* and *"2b"* messages and to eventually learn of $E_G^2\phi$. ***G1a*** ensures that at least one propagator is always available.
- The protocol cannot make progress in each phase unless a propagator receives *"1b"* and *"2b"* messages from all learners. ***G1b*** ensures that all learners are always available.

We can see that the conditions identified are necessary for proving correctness under our system model. It is difficult to prove that these conditions are the weakest conditions required because there may be multiple equally-weak sets of conditions under which the proofs can be completed. Proving the weakest set, therefore, requires formalizing the meaning of "weakness" in the context of these conditions. Moreover, the strong guarantees provided by mechanically-verified proofs justify the possibly-conservative conditions in the context of safety-critical applications. Therefore, for now, we are satisfied with the informal analysis of the necessity of the conditions and will consider the set $\{$***G1a***, ***G1b***$\}$ to be one of the weakest sets of conditions required for proving correctness under our system model.

TAP is a general-purpose distributed knowledge propagation protocol that can be used in asynchronous systems. In line with their definitions, propagators and learners are logical abstractions and a particular physical node may functionally implement both abstractions simultaneously. If the protocol is used in a scenario where the physical nodes exclusively implement either a propagator or a learner, but not both, then the system can withstand the failure of multiple propagators as long as ***G1a*** holds. However, in the particular use case of DAC, the set of aircraft K, which are responsible for propagating a value ϕ, may not have any knowledge about the membership of the set K, making it difficult to physically separate the set of propagators and learners. Therefore, this application necessitates the pessimistic implementation of the protocol where all aircraft in the set G functionally implement a learner and all aircraft in the set K functionally implement both a propagator and a learner. As a consequence, the set of physical nodes implementing propagators will be a subset of the physical nodes implementing learners. Under such circumstances, the condition that *all learners are always available* will imply that *all propagators are always available.* Since the set of propagators and learners are non-empty, ***G1b*** will imply ***G1a*** under such circumstances. Nevertheless, we will still use the weaker condition ***G1a*** for our proofs because this will allow the proofs to be pertinent, without loss of generality, even for applications where the implementation may not necessarily warrant ***G1b*** to imply ***G1a***.

3.4 Formal Verification of the Correctness Properties of TAP

In this section, we will formally specify some correctness properties for our protocol under the identified conditions and the assumed system model.

Our formal specification represents a distributed system as a *distributed state machine* [153] that has a current state at any given time and changes its state by performing some action. The temporal dimension is logically represented by natural numbers. Each message in the system is a tuple $\langle \rho \in \mathcal{C}$, $\delta \in \mathcal{R}$, $\nu \in \mathcal{V}$, and $\gamma \in \xi \rangle$ where $\xi = \{1a, 1b, 2a, 2b\}$, ρ is a propagator ID, δ is a learner ID, and ν is a fact. For any *state* of the system, μ represents the set of all messages in the state and τ represents the time at which the state was recorded. Assuming that the fact that needs to be propagated using the protocol is ϕ, further notations used in our formal model can be found in Table 3.1 and Table 3.2.

Table 3.1: Set symbols for the formal specification of knowledge propagation.

Symbol	Description
$\mathcal{C}$	The set of all propagators
$\mathcal{R}$	The set of all learners
$\mathcal{V}$	The set of all facts
$\mathcal{T}$	The set of all logical time steps
$\mathcal{M}$	The set of all possible messages

3.4.1 The Assumptions Required for Correctness

Given below are some assumptions that are direct logical implications of the system model and the conditions required for correctness.

A1 By ***G1a*** there is a propagator which is always available.

$\texttt{A1} \equiv \exists c \in \mathcal{C} : \forall t \in \mathcal{T} : \alpha(c, t)$

A2 By ***G1b*** all learners are always available.

$\texttt{A2} \equiv \forall r \in \mathcal{R} : \forall t \in \mathcal{T} : \alpha(r, t)$

A3 All messages which are sent must be eventually added to the channel.

$\texttt{A3} \equiv \forall t_s \in \mathcal{T} : \forall m \in \mathcal{M} : (\S(m, t_s) \implies \exists t_d \in \mathcal{T} : (t_d > t_s) \wedge \P(m, t_d))$

Table 3.2: Relation symbols for the formal specification of knowledge propagation.

Symbol	Description	Input	Output
κ	The learner knows the fact	$\mathcal{R} \times \mathcal{V}$	`Bool`
$\dot{\varepsilon}$	Learner knows that all learners know the fact	$\mathcal{R} \times \mathcal{V}$	`Bool`
$\ddot{\varepsilon}$	Propagator knows that all learners know that all learners know the fact	$\mathcal{C} \times \mathcal{V}$	`Bool`
Φ_{1a}	There is a message of type *“1a”* with the value and the propagator ID	$\mathcal{C} \times \mathcal{V}$	`Bool`
Φ_{1b}	There is a message of type *“1b”* with the value and the learner ID	$\mathcal{R} \times \mathcal{V}$	`Bool`
Φ_{2a}	There is a message of type *“2a”* with the propagator ID	$\mathcal{C}$	`Bool`
Φ_{2b}	There is a message of type *“2b”* with the learner ID and the propagator ID	$\mathcal{R} \times \mathcal{C}$	`Bool`
Δ_0	The state contains no messages	$\mathcal{T}$	`Bool`
Δ_{1a}	The system state contains *"1a"* messages from the propagator	$\mathcal{T} \times \mathcal{C}$	`Bool`
Δ_{1b}	The system state contains *"1b"* messages from the propagator	$\mathcal{T} \times \mathcal{C}$	`Bool`
Δ_{2a}	The system state contains *"2a"* messages from the propagator	$\mathcal{T} \times \mathcal{C}$	`Bool`
Δ_{2b}	The system state contains *"2b"* messages from the propagator	$\mathcal{T} \times \mathcal{C}$	`Bool`
$\S$	The message has been sent at the time	$\mathcal{M} \times \mathcal{T}$	`Bool`
$\P$	The message has been delivered at the time	$\mathcal{M} \times \mathcal{T}$	`Bool`
α	The agent is available at the time	$\mathcal{C}\|\mathcal{R} \times \mathcal{T}$	`Bool`
$\mathcal{E}$	The agent knows about ϕ at the time	$\mathcal{C}\|\mathcal{R} \times \mathcal{T}$	`Bool`
$\dot{\mathcal{E}}$	The agent knows that all agents know about ϕ at the time	$\mathcal{C}\|\mathcal{R} \times \mathcal{T}$	`Bool`
$\ddot{\mathcal{E}}$	The agent knows that all agents know that all agents know about ϕ at the time	$\mathcal{C}\|\mathcal{R} \times \mathcal{T}$	`Bool`

A4 Since messages cannot be corrupted, if at any time a message is added to the channel, there must have been a time when it was sent.

$$\texttt{A4} \equiv \forall t_d \in \mathcal{T} : \forall m \in \mathcal{M} : (\P(m, t_d) \implies \exists t_s \in \mathcal{T} : (t_d > t_s) \land \S(m, t_s))$$

We split these assumptions into two sets $\mathfrak{A}_c =$ ***{A2, A4}*** and $\mathfrak{A}_p =$ ***{A1, A2, A3}*** and use them separately for proving safety and progress respectively.

3.4.2 Some Important Correctness Lemmas

We present some important lemmas that will later be used to formally prove the required correctness properties for TAP.

Lemma 3.1 *Given $\mathfrak{A}_p$, for all propagators, if there exists a time such that the propagator c is available it has not received* "1b" *messages from all learners, then* "1a" *messages from c will eventually be delivered.*

`Lemma` $3.1 \equiv$

$$\mathfrak{A}_p \implies (\forall c \in \mathcal{C} : (\exists t \in \mathcal{T} : \tau = t \wedge \alpha(c,t) \wedge \neg(\forall r \in \mathcal{R} : \Phi_{1b}(r,c))) \\ \implies (\exists t_2 \in \mathcal{T} : \tau = t_2 \wedge (\exists v \in \mathcal{V} : \Phi_{1a}(c,v))))$$

Lemma 3.2 *Given $\mathfrak{A}_p$, for all propagators, if there exists a time such that* "1a" *messages from a propagator c have been delivered, then* "1b" *messages from all learners will eventually be delivered.*

`Lemma` $3.2 \equiv$

$$\mathfrak{A}_p \implies (\forall c \in \mathcal{C} : (\exists t \in \mathcal{T} : \tau = t \wedge (\exists v \in \mathcal{V} : \Phi_{1a}(c,v))) \\ \implies (\exists t_2 \in \mathcal{T} : \tau = t_2 \wedge (\forall r \in \mathcal{R} : \Phi_{1b}(r,c))))$$

Lemma 3.3 *Given $\mathfrak{A}_p$, for all propagators, if there exists a time such that the propagator c is available and it has received* "1b" *messages from all learners, then* "2a" *messages from c will eventually be delivered.*

`Lemma` $3.3 \equiv$

$$\mathfrak{A}_p \implies (\forall c \in \mathcal{C} : (\exists t \in \mathcal{T} : \tau = t \wedge \alpha(c,t) \wedge (\forall r \in \mathcal{R} : \Phi_{1b}(r,c))) \\ \implies (\exists t_2 \in \mathcal{T} : \tau = t_2 \wedge (\exists v \in \mathcal{V} : \Phi_{2a}(c))))$$

Lemma 3.4 *Given $\mathfrak{A}_p$, for all propagators, if there exists a time such that* "2a" *messages from a propagator c have been delivered, then* "2b" *messages from all learners will eventually be delivered.*

`Lemma` $3.4 \equiv$

$$\mathfrak{A}_p \implies (\forall c \in \mathcal{C} : (\exists t \in \mathcal{T} : \tau = t \wedge (\exists v \in \mathcal{V} : \Phi_{2a}(c))) \\ \implies (\exists t_2 \in \mathcal{T} : \tau = t_2 \wedge (\forall r \in \mathcal{R} : \Phi_{2b}(r,c))))$$

Lemma 3.5 *Given $\mathfrak{A}_p$, there will be a propagator c such that $\Delta_{1a}(c,t)$ will eventually be true .*

`Lemma` $3.5 \equiv$
$\mathfrak{A}_p \implies (\exists c \in \mathcal{C} : \exists t \in \mathcal{T} : \Delta_{1a}(c,t))$

Lemma 3.6 *Given $\mathfrak{A}_p$, there will be a propagator c such that $\Delta_{1b}(c,t)$ will eventually be true.*

`Lemma` $3.6 \equiv$
$\mathfrak{A}_p \implies (\exists c \in \mathcal{C} : \exists t \in \mathcal{T} : \Delta_{1b}(c,t))$

Lemma 3.7 *Given $\mathfrak{A}_p$, there will be a propagator c such that $\Delta_{2a}(c,t)$ will eventually be true.*

`Lemma` $3.7 \equiv$
$\mathfrak{A}_p \implies (\exists c \in \mathcal{C} : \exists t \in \mathcal{T} : \Delta_{2a}(c,t))$

Lemma 3.8 *Given $\mathfrak{A}_p$, there will be a propagator c such that $\Delta_{2b}(c,t)$ will eventually be true.*

`Lemma` $3.8 \equiv$
$\mathfrak{A}_p \implies (\exists c \in \mathcal{C} : \exists t \in \mathcal{T} : \Delta_{2b}(c,t))$

Lemma 3.9 *Given $\mathfrak{A}_c$, always, if an available propagator c knows $E_G^2\phi$, then all learners will know $E_G\phi$.*

`Lemma` $3.9 \equiv$
$\mathfrak{A}_c \implies (\forall t \in \mathcal{T} : \forall c \in \mathcal{C} : ((\tau = t \land \alpha(c,t) \land \ddot{\mathcal{E}}(c,t)) \implies (\forall r \in \mathcal{R} : \dot{\mathcal{E}}(r,t))))$

Lemma 3.10 *Given $\mathfrak{A}_c$, always, if an available learner r knows $E_G\phi$, then all learners will know ϕ.*

`Lemma` $3.10 \equiv$
$\mathfrak{A}_c \implies (\forall t \in \mathcal{T} : \forall r \in \mathcal{R} : ((\tau = t \land \alpha(r,t) \land \dot{\mathcal{E}}(r,t)) \implies (\forall r \in \mathcal{R} : \mathcal{E}(r,t))))$

3.4.3 The Correctness Properties

The safety property (Theorem 3.1) states that a propagator will know $E_G^2\phi$ if and only if all learners know both $E_G\phi$ and ϕ.

Theorem 3.1 *(Safety of TAP) Given $\mathfrak{A}_c$, if an available propagator knows about $E_G^2\phi$, then all learners know about ϕ and $E_G\phi$.*

`Theorem` $3.1 \equiv$
$\mathfrak{A}_c \implies (\forall t \in \mathcal{T} : \forall c \in \mathcal{C} : ((\tau = t \wedge \alpha(c,t) \wedge \ddot{\mathcal{E}}(c,t)) \implies (\forall r \in \mathcal{R} : \mathcal{E}(r,t) \wedge \dot{\mathcal{E}}(r,t))))$

In order to prove safety, it must be shown that a propagator can know $E_G^2\phi$ if and only if all learners know both $E_G\phi$ and ϕ. Lemma 3.9 and Lemma 3.10 can be used to prove that the safety property, represented by Theorem 3.1, is satisfied by our protocol at all times. The lemmas can be proven directly by using the proposed system model and the set of safety assumptions $\mathfrak{A}_c$.

The eventual progress property (Theorem 3.2) states that eventually $E_G^2\phi$ will be attained. Since there is no external agent that can monitor the state of the system, it will be necessary and sufficient to show that eventually, at least one propagator will learn $E_G^2\phi$.

Theorem 3.2 *(Eventual Progress in TAP) Given $\mathfrak{A}_p$, eventually, a propagator will know about $E_G^2\phi$.*

`Theorem` $3.2 \equiv$
$\mathfrak{A}_p \implies \exists t \in \mathcal{T} : \exists c \in \mathcal{C} : \ddot{\mathcal{E}}(c,t)$

In order to prove eventual progress, we use the set of assumptions $\mathfrak{A}_p$ and the system model to show that eventually, at least one propagator will learn $E_G^2\phi$. Lemma 3.1 to Lemma 3.4 state some temporal guarantees about how the protocol will proceed under some system conditions. By ***A1***, there is at least one propagator which is always available. Using ***A1***, Lemma 3.5 to Lemma 3.8 can be used to show that at least one propagator will eventually receive *"2b"* messages from all learners, thereby learning $E_G^2\phi$ by non-Byzantine behavior. Lemma 3.5 to Lemma 3.8 are proven with the help of Lemma 3.1 to Lemma 3.4. Lemma 3.8 is then used to prove Theorem 3.2.

3.4.4 The Proofs of Correctness

The detailed proof sketches for the lemmas and the theorems are given below.

Proof of Lemma 3.1 -

(1) By non-Byzantine behavior of propagators, an available propagator will eventually send *"1a"* messages.

(2) By (1) and ***A3***, eventually *"1a"* messages from the propagator will be delivered.
□

Proof of Lemma 3.2 -

(1) By ***A2***, all learners are always available.

(2) By non-Byzantine behavior of learners, an available learner will eventually send *"1b"* messages.

(3) By (1), (2), and ***A3***, eventually *"1b"* messages from all learners will be delivered.
□

Proof of Lemma 3.3 -

(1) By non-Byzantine behavior of propagators, an available propagator will eventually send *"2a"* messages.

(2) By (1) and ***A3***, eventually *"2a"* messages from the propagator will be delivered.
□

Proof of Lemma 3.4 -

(1) By ***A2***, all learners are always available.

(2) By non-Byzantine behavior of learners, an available learner will eventually send *"2b"* messages.

(3) By (1), (2), and ***A3***, eventually *"2b"* messages from all learners will be delivered.
□

Proof of Lemma 3.5 -

(1) By ***A1***, a propagator is always available.

(2) $\exists t \in \mathcal{T} : \Delta_0(t)$ since $\mu = \{\}$ in the initial state.

(3) By (1), (2), and Lemma 3.1, eventually *"1a"* messages from a propagator will be delivered.

$\square$

Proof of Lemma 3.6 -

(1) By Lemma 3.5, eventually *"1a"* messages from an available propagator will be delivered.

(2) By (1) and Lemma 3.2, eventually *"1b"* messages from all learners will be delivered.

$\square$

Proof of Lemma 3.7 -

(1) By ***A1***, a propagator is always available.

(2) By Lemma 3.6, eventually *"1b"* messages from all learners will be delivered to the propagator.

(3) By (1), (2), and Lemma 3.3, eventually *"2a"* messages from the propagator will be delivered.

$\square$

Proof of Lemma 3.8 -

(1) By Lemma 3.7, eventually *"2a"* messages from an available propagator will be delivered.

(2) By (1) and Lemma 3.4, eventually *"2b"* messages from all learners will be delivered.

$\square$

Proof of Lemma 3.9 -

(1) By non-Byzantine behavior of propagators, an available propagator can only learn $E_G^2\phi$ if it has received *"2b"* messages from all learners.

(2) By ***A4***, if *"2b"* messages from all learners have been received, they must have been sent by the learners.

(3) By non-Byzantine behavior of available learners, a learner can only send *"2b"* messages if it knows $E_G\phi$.

(4) By ***A2***, all learners are always available.

(5) By (1), (2), (3), and (4), all learners know $E_G\phi$.
□

Proof of Lemma 3.10 -

(1) By non-Byzantine behavior of learners, an available learner can only learn $E_G\phi$ if it has received *"2a"* messages from a propagator.

(2) By ***A4***, if *"2a"* messages from a propagator have been received, they must have been sent by the propagator.

(3) By non-Byzantine behavior of available propagator, a propagator can only send *"2a"* messages if it has received *"1b"* messages from all learners.

(4) By non-Byzantine behavior of available learners, a learner can only send *"1b"* messages if it knows ϕ.

(5) By ***A2***, all learners are always available.

(6) By (1), (2), (3), (4), and (5), all learners know ϕ.
□

Proof of Theorem 3.1 -

(1) Trivially by Lemma 3.9 and Lemma 3.10.
□

Proof of Theorem 3.2 -

(1) By non-Byzantine behavior of available propagator, a propagator will learn about $E_G^2\phi$ if it has received *"2b"* messages from all learners.

(2) By Lemma 3.8, an available propagator will eventually receive *"2b"* messages from all learners.

(3) By (1) and (2) an available propagator will eventually know $E^2_G\phi$.
□

We have used TLA$^+$ to specify our formalizations and TLAPS to mechanically verify all the proof sketches presented above[11]. Since TLAPS does not currently support first-order temporal logic we have incorporated time explicitly in our specification using first-order temporal logic[12]. The specification includes some additional temporal axioms required for the proofs. The proofs and specifications consist of about 1500 lines of TLA$^+$ code.

3.5 Related Work

There exists prior work in the literature on reasoning about knowledge states in distributed multi-agent systems. Halpern and Fagin [71] present a formal model to capture the interaction between knowledge and action in distributed systems by modeling the distributed system as a set of *runs*. They define a run as a function from time to global states of the system. They define *knowledge based* protocols where a process' actions depend on its knowledge and can be used to describe high-level descriptions of a process' behavior depending on its local state. Ksehmkalyani [95] examines the feasibility of achieving $E^n\phi$ for $n > 1$ and uses a restricted distributed messaging framework to explore the possibility of achieving $E^n\phi$ in asynchronous systems by using only logical clocks. They also explore the use of different types of logical clocks for attaining *concurrent common knowledge* by using their framework. Fagin and Halpern [53] provide a model for explicitly incorporating probability in knowledge logic formulas to reason about knowledge and probability. They introduce a probabilistic variant of common knowledge in multi-agent systems and provide fundamental axioms for reasoning about the same. Fagin and Vardi [56] investigate a model of distributed communication and provide a logical formalization of runs. They also analyze the logic of implicit knowledge, which is the knowledge that can be attained by combining the knowledge of a group. Their work explores the dependence of knowledge in a distributed system on the way processes communicate with one another. Guzmán *et al.* [69] introduce the theory of

[11]Complete TLAPS proofs available at https://wcl.cs.rpi.edu/assure
[12]TLAPS 1.4.5. as of June 2020 only supports propositional temporal logic

group space functions to reason about the information distributed among the members of a potentially infinite group. They develop the semantic foundations and algorithms to reason about knowledge in multi-agent systems and analyze the properties of distributed spaces for reasoning about the knowledge of such systems. Matteo [89] analyzes the use of Datalog [52] to reason about the knowledge of a group of distributed nodes and develop Knowlog, which is a variant of Datalog that can express nodes' state of knowledge using a set of epistemic modal operators. Their approach abstracts the modes of communication exchange from states of knowledge so that reasoning is simplified. They use an implementation of the *Two-Phase Commit* protocol to analyze their approach. Fagin *et al.* [55] present a formal model for the analysis of attainable knowledge states in distributed systems under certain assumptions. Their model can be used to formalize assumptions about distributed systems, such as whether they are deterministic or non-deterministic and whether the knowledge is cumulative or non-cumulative. They also provide a complete axiomatization of knowledge for some important cases of interest. Van Der Mayden [157] establishes the completeness of a logic of knowledge and time for all classes of systems that satisfy the perfect recall property which is true if, at all times, a processor's state includes a record of its previous states. They also provide an abstract characterization of systems with perfect recall that can be used to prove completeness. Kuznets *et al.* [98] propose a framework for reasoning about knowledge in multi-agent asynchronous systems with Byzantine fault failures. Their framework combines epistemic and temporal logic for specifying distributed protocols and their behaviors. The modularity of their approach allows modeling of any timing and synchrony properties of distributed systems. Knight *et al.* [94] propose a logic for reasoning about epistemic messages in asynchronous distributed systems where knowledge is true at the start of announcement and agents can predict messages that have been sent, but not yet received. They extend the Public Announcement Logic in which announcements from an external source can be used to model messages broadcast by agents within the system. Halpern [70] presents a survey of formalizations of knowledge in asynchronous and unreliable distributed systems, and provides examples to argue why this formalization of knowledge is important for analyzing distributed systems. Dwork *et al.* [50] present a general framework for formalizing and reasoning about knowledge in distributed systems. They formalize the relationship between common knowledge, global knowledge, and other states of knowledge, and show that in real-life distributed systems, common knowledge cannot be attained. They also investigate other weaker states of knowledge that are practically attainable. Halpern *et al.* [72] present a categorization of epistemic and temporal logic along two dimensions: the language used and the assumptions about the underlying distributed systems. They use these categorizations to introduce ninety-six logics, which they investigate by characterizing their complexity and their dependence on parameter combinations. However, none of this work has presented any mechanically-verified knowledge propagation protocols.

Previous work on formal verification of *atomic-commit* protocols has been limited to model checking. Popovic *et al.* [144] have presented a model checking-based approach for analysis of distributed transaction management protocols using the SMV formal verification tool and have used their approach for the verification of the Two-Phase Commit protocol. Atif [8] has presented an analysis of the Two-Phase Commit protocol and its variant, the *Three-Phase Commit* protocol, by using the process algebra mCRL2 and modal μ-calculus. They have model checked both variants of the protocol under different communication settings to analyze their behavior in practical distributed networks. None of the above work has presented any mechanically-verified theorems that can guarantee that correctness properties will hold in infinite input states.

We improve upon prior work by presenting a formally-verified distributed knowledge propagation protocol that can be used to attain a safe state of knowledge for multi-agent distributed applications like DAC and providing machine-checked formal proofs of some correctness guarantees necessary for safety-critical aerospace applications.

3.6 Chapter Summary

In this chapter, we have formalized a safe state of knowledge required for our Decentralized Admission Control approach and presented a distributed knowledge propagation protocol TAP for attaining the safe state of knowledge. We have specified two correctness properties for TAP — safety and progress, and identified a set of system conditions under which these correctness properties can be guaranteed. The set of conditions we have identified is practical and sufficient and we have informally discussed that this may be one of the weakest sets of conditions that will be required for guaranteeing the relevant correctness properties. Using TLAPS, we have developed mechanically-verified proofs to show that the properties can be guaranteed under the identified conditions, thereby making our protocol suitable for safety-critical aerospace applications.

Since aircraft have limited time to remain airborne and air-traffic data has a short useful lifetime, aircraft must be able to make operational decisions within some time bounds. Therefore, a guarantee of eventual progress *alone* is not sufficient under such circumstances. A potential future direction of work would be to investigate the development of guarantees of timely progress that can be more useful. This can be done by using statistical observations about message delays in the IoP. Our current protocol also relies on the assumption that all aircraft are non-Byzantine, requiring them to be cooperative and "well behaved". This is a strong assumption and does not consider pilot errors and other uncertainties presented by real-world applications. Therefore, another potential direction of work would be to expand our system model to accommodate Byzantine and adversarial behavior of aircraft.

CHAPTER 4
REASONING ABOUT PROBABILISTIC PROPERTIES OF DISTRIBUTED COORDINATION ALGORITHMS

4.1 Chapter Introduction

In Chapter 2 and Chapter 3, we have provided formal guarantees of eventual progress for distributed consensus and knowledge propagation algorithms useful for our Decentralized Admission Control approach. However, a guarantee of eventual progress does not provide any bound on the amount of time that may be required for a distributed algorithm to achieve its goal. As air traffic data usually has a short useful lifetime and aircraft have limited time to remain airborne, a guarantee of eventual progress *alone* is insufficient for time-critical UAM applications like DAC. To be useful, a progress guarantee must provide some time bounds that the aircraft can use to make important decisions. *E.g.*, if there is a guarantee that consensus will take at most 1 second, then a candidate can decide to only compute flight plans that start after 1 second.

The total *clock time* that is required for successful completion of a distributed algorithm is dependent on three factors — *message transmission delays*, *message processing delays*, and the number of messages involved. In asynchronous communication networks like the IoP, message delays are unbounded, making it impossible to provide deterministic bounds on the total time that may be required for a distributed algorithm to make progress. In the absence of deterministic time bounds, it is possible to provide *probabilistic bounds* for the timely progress of distributed algorithms by modeling the factors affecting total time as *stochastic processes*. This can be done by probabilistically reasoning about the message delays and developing appropriate guarantees of probabilistic properties of timely progress for distributed algorithms. *E.g.,* if a timely progress property states that *"Knowledge propagation will take at most 5 seconds with a probability 99.99999999%"*, then an aircraft can choose to only compute flight plans that start after 5 seconds to ensure that the plans don't become obsolete by the time that they are successfully propagated.

In this chapter, we introduce machine-checked guarantees of probabilistic properties

Portions of this chapter previously appeared as: Paul, S., Patterson, S., Varela, C.A.: Formal guarantees of timely progress for distributed knowledge propagation. In: Formal Methods for Autonomous Systems (FMAS), pp. 73-91 (2021). doi:10.4204/EPTCS.348.5

of timely progress for distributed algorithms (assuming eventual progress) by using theories apposite to *low-altitude platforms* (LAP) [25] of airborne communication such as VANETs. Particularly, we formalize the theory of the *Multicopy Two-Hop Relay* (MTR) protocol [112] to model message transmission delays in VANETs and the *M/M/1 queue system* [110] to model message processing delays. We then use the probabilistic bounds on these delays to provide a sufficiently high-level guarantee of timely progress for the knowledge propagation algorithm TAP (Chapter 3) that can be used for DAC. Furthermore, we showcase the development of a formal library in Athena, that is tailored towards reasoning about probabilistic properties of timely progress for distributed algorithms deployed over VANETs.

The rest of the chapter is structured as follows: Section 4.2 presents our approach of developing probabilistic properties of timely progress for distributed coordination algorithms; Section 4.3 showcases the development of a formal proof library in Athena tailored towards reasoning about probabilistic properties of distributed algorithms deployed over VANETs like the IoP; Section 4.4 presents prior work related to progress analysis and formalization of relevant mathematical theories; and Section 4.5 concludes the chapter with a summary and presents some potential future directions of work.

4.2 Probabilistic Properties of Timely Progress for Distributed Algorithms

For any distributed algorithm, if eventual progress is guaranteed, then the total time taken for successful propagation is dependent on the following:

1. message transmission delays (T_{D_m}),
2. message processing delays (T_{D_m}), and
3. the total number of messages involved (N_m).

In the absence of deterministic properties, it is possible to employ a probabilistic approach for providing formal guarantees of timely progress. This can be done by probabilistically reasoning about the message delays using either *theoretical models* or *data-driven models*. In contrast to theoretical models that are based on analytical results about system characteristics, data-driven models are usually based on real-time or historical statistical observations about a system. This chapter focuses on using theoretical models for reasoning about timely progress of distributed coordination algorithms.

For UAM applications, it is desirable to choose theoretical models that are appropriate for VANETs like the IoP. Through the IoP, aircraft should be able to not only communicate their own information but also *relay* received information from other aircraft using a *multi-hop ad-hoc* approach [162]. 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. This makes it appropriate to use *queueing theory* [110] to reason about message processing delays. The M/M/1 system is an elementary queue system that consists of a single server that processes all messages which are received, making it a reasonable choice for modeling message processing with respect to a single aircraft. The M/M/1 system also provides some useful analytical properties that make it possible to formally reason about the message processing delays. Therefore, to develop formal probabilistic properties of timely progress for distributed algorithms deployed over VANETs, we assume the following:

- all aircraft use the MTR protocol for message transmission between each other,
- each aircraft independently implements an M/M/1 queue to process the messages that it receives,
- the transmission delays of the messages are *independent and identically distributed* (IID),
- the processing delays of the messages are IID, and
- the transmission delays are independent of the processing delays.

4.2.1 Modeling the Message Transmission Delays in VANETs

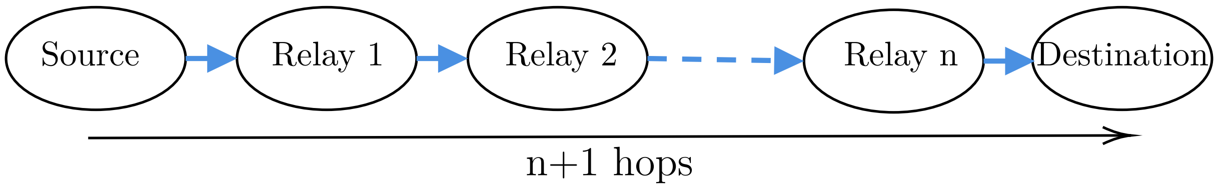


Figure 4.1: Message transmission via relaying.

Relaying (Fig. 4.1), for data transmission between a *source* and a *destination* when the two nodes are not within *transmission range*, has been proposed to be an efficient mode of communication in VANETs [68]. Relaying has also been considered to be appropriate for VANETs like the IoP where two communicating aircraft may not always be within direct *transmission range* [162]. Therefore, to ensure that our assumptions about message transmission delays in VANETs are consistent with prior work in the literature, we use the theory of the MTR protocol. For this, we partially base our specification of MTR on the description of the protocol presented by Al Hanbali *et al.* [4].

In the basic two-hop relay protocol, there are a set of $M + 1$ mobile nodes whose trajectories are IID [68]. If a *source* node wants to transfer a message to a *destination* node, it can either transmit it directly to the destination, 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.

In the MTR protocol [4], transmission delay T_{D_m} is the time taken for the destination to receive a message m or a copy of m for the first time. Two nodes *meet* when they come within the transmission range of one another and *inter-meeting time* is the time interval between two consecutive meetings of a given pair of nodes. The inter-meeting times of all pairs of nodes are IID with the common *cumulative distribution function* (CDF) $G(t)$. The source can only transmit a message to a relay that does not already hold a copy. Message transmission between two nodes within range is instantaneous. Each copy of a message has a *time-to-live* (TTL), which is the time after which a relay has to drop the copy if it has not been transmitted by the relay yet.

To derive a probabilistic bound on the time taken to transmit a message using MTR, we make the following assumptions:

- the TTLs for all messages are unrestricted, *i.e.*, a message can be held by a relay node until it can transmit it,
- the inter-meeting times of the mobile nodes are *exponentially distributed* with the *rate parameter* λ_{MTR},
- since delivery is instantaneous when the nodes meet and TTLs are unrestricted, the

inter-meeting time between the source and the destination (T_{sd}) or between a relay i and the destination (T_{r_id}) also represents the time taken by the source or the relay to directly deliver a message to the destination, and

- the source node transmits the copy of a message to all $M-1$ relay nodes. A message (or a copy) can, therefore, be delivered to the destination by the source or any of the relay nodes. Hence, the actual time taken to deliver the message will be the minimum of $\{T_{sd}, T_{r_1d}, ..., T_{r_{M-1}d}\}$ (Eq. 4.1).

$$T_{D_m} = \min(\{T_{sd}, T_{r_1d}, ..., T_{r_{M-1}d}\}) \tag{4.1}$$

By using the above assumptions and necessary theories from probability, random variables, and exponential distributions, we have formally derived the following probabilistic bound on the time taken to deliver a message m using MTR:

$$P\left(T_{D_m} \leq t\right) = 1 - e^{-\lambda_{MTR}Mt} \tag{4.2}$$

Now, the derivative of the above expression conforms to the *probability density function* (PDF) of an exponential distribution with rate parameter $\lambda_{MTR}M$. Therefore, we can conclude that T_{D_m} is exponentially distributed with a rate parameter $\lambda_{D_m} = \lambda_{MTR}M$.

4.2.2 Modeling the Message Processing Delays in VANETs

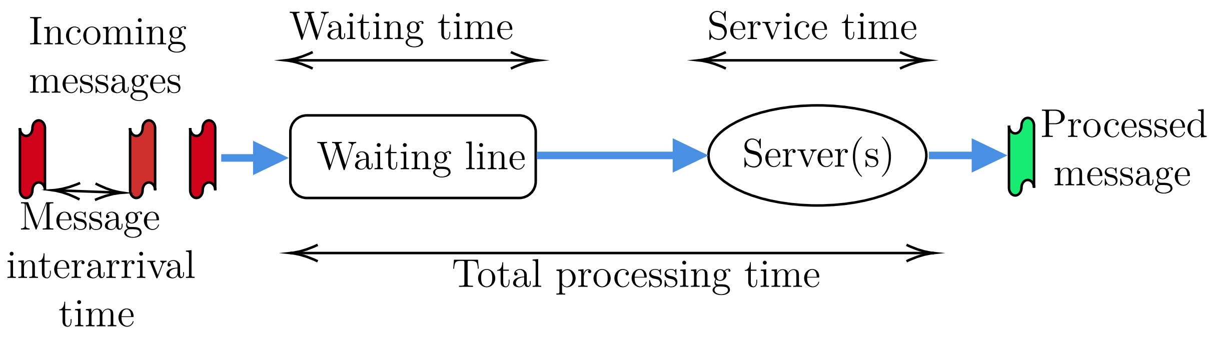


Figure 4.2: A typical queuing system.

Queueing theory (Fig. 4.2) has been extensively used in the literature for modeling throughput in VANETs [97, 161, 164, 166]. An aircraft communicating using the IoP can be considered to be a single server where messages arrive, wait some time in a queue until they are processed, and take some time to be processed. Therefore, for modeling the message processing delays in the aircraft, we use the theory of queues, particularly the *M/M/1* queue system [110]. We choose the M/M/1 queue system because it provides elegant analytical properties for computing the total time T_{P_m} required for a message m to be processed. This processing time includes the time spent in the queue and the time spent in actually processing the message. An important property of T_{P_m} in the M/M/1 queue system is that T_{P_m} is exponentially distributed [16].

The M/M/1 queue system consists of a single server where customers arrive according to a *Poisson process* [106]. It has the following characteristics [16]:

- the *interarrival times* of messages in the queue are exponentially distributed with a rate parameter λ_a,
- the *service time* of messages, which is the time spent in actually processing a message, is exponentially distributed with a mean $1/\mu_s$,
- the queue is managed by a single server, and
- the number of message arrivals in an interval of length τ follows a *Poisson distribution* [109] with a parameter $\lambda_a \tau$.

To probabilistically model T_{P_m}, we use *Little's theorem* [111], which is an important fundamental result in queueing theory. If N_q and T_p represent the *average number of messages* and the *mean processing delay* (queueing delay + service delay) respectively[13], then Little's Theorem states the useful relationship conveyed in Eq. 4.3.

$$N_q = \lambda_a T_p \tag{4.3}$$

A proof of Little's Theorem has been presented in [16], which we have formalized in Athena[14]. The proof uses $N(\tau)$, $\alpha(\tau)$, and $T(i)$ to represent the number of messages in the system at time τ, the number of messages that arrived in the interval $[0, \tau]$, and the average time spent in the system by message i respectively. The average arrival rate over the time period $[0, t]$

[13]Note that $T_p = \mathbf{E}\left[T_{P_m}\right]$ is the *mean* or the *expected value* of T_{P_m}.

[14]We present the detailed proof described in [16] to clearly highlight the deficiencies in the Athena formalization of the proof in version 0.1 of our proof library.

is then given by:

$$\lambda_t = \frac{\alpha(t)}{t} \tag{4.4}$$

Similarly, the average processing delay of the messages that arrived in the interval $[0, t]$ is given by:

$$T_t = \sum_{i=1}^{\alpha(t)} \frac{T(i)}{\alpha(t)} \tag{4.5}$$

The average number of messages in the system in the interval $[0, t]$ is given by:

$$N_t = \frac{1}{t} \int_0^t N(\tau) d\tau \tag{4.6}$$

Now, the graphical analysis of a FIFO system reveals the following [16]:

$$\frac{1}{t} \int_0^t N(\tau) d\tau = \frac{1}{t} \sum_{i=1}^{\alpha(t)} T(i) \tag{4.7}$$

Using Eq. 4.4 to Eq. 4.7, it can be shown that:

$$N_t = \lambda_t T_t \tag{4.8}$$

Assuming that the limits $N_q = \lim_{t\to\infty} N_t$, $\lambda_a = \lim_{t\to\infty} \lambda_t$, and $T_p = \lim_{t\to\infty} T_t$ exist, we can now obtain Eq. 4.3 to prove Little's Theorem.

Now, using the properties of the Poisson distribution, it has been shown in [16] that:

$$N_q = \frac{\lambda_a}{\mu_s - \lambda_a} \tag{4.9}$$

Again, from Eq. 4.9 and Eq. 4.3, we can obtain the following relationship:

$$T_p = \frac{1}{\mu_s - \lambda_a} \tag{4.10}$$

Finally, since T_{P_m} is exponentially distributed, its rate parameter λ_{P_m} can be obtained as[15]:

$$\lambda_{P_m} = \frac{1}{\mathbf{E}\left[T_{P_m}\right]} = \frac{1}{T_p} = \mu_s - \lambda_a \tag{4.11}$$

[15]For an exponentially distributed random variable, the rate parameter is the reciprocal of the mean.

4.2.3 Use Case: Timely Progress for the Two-Phase Acknowledge Protocol

We had introduced the distributed knowledge propagation protocol TAP in Chapter 3. In TAP, from the perspective of a propagator, successful propagation involves a deterministic number of messages — for each learner, 4 messages are required, so for R learners, the total number of messages required for successful propagation is $4 \times R$. If eventual progress is guaranteed, then in the worst-case, when there is no concurrency in message transmission or processing, the total time (T_S) taken for successful propagation can be calculated by using the total number of messages that are involved (N_M), and the transmission delay (T_{D_m}) and the processing delay (T_{P_m}) of each message m. The total time for the worst-case scenario can, therefore, be obtained using Eq. 4.12.

$$T_S = \sum_{m=1}^{N_M} T_{D_m} + \sum_{m=1}^{N_M} T_{P_m} \tag{4.12}$$

Using the rate parameters λ_{D_m} and λ_{P_m} for the exponentially distributed message transmission and processing delays, we can now provide a probabilistic bound on the worst-case time for progress T_S. For that, let

$$T_D = \sum_{m=1}^{N_M} T_{D_m} \qquad \& \qquad T_P = \sum_{m=1}^{N_M} T_{P_m} \tag{4.13}$$

From Eq. 4.12 and Eq. 4.13, for any two real numbers x and y, we can state:

$$((T_D \leq x) \land (T_P \leq y)) \implies (T_S \leq (x + y)) \tag{4.14}$$

Now, for two events, A and B, the following statement holds [117]:

$$(A \implies B) \implies (P(B) \geq P(A)) \tag{4.15}$$

Therefore, from Eq. 4.14 and Eq. 4.15, we get:

$$P(T_S \leq (x + y)) \geq P((T_D \leq x) \land (T_P \leq y)) \tag{4.16}$$

Since the processing delays are independent of the transmission delays, by the *product rule* of independent events [63] we have[16]:

$$P((T_D \leq x) \wedge (T_P \leq y)) = P(T_D \leq x) \times P(T_P \leq y) \tag{4.17}$$

From Eq. 4.16 and Eq. 4.17 we have:

$$P(T_S \leq (x + y)) \geq P(T_D \leq x) \times P(T_P \leq y) \tag{4.18}$$

Now, both T_D and T_P are sums of IID exponential random variables. Therefore, T_D and T_P follow two different *Erlang distributions* [109], each of which are parameterized by a *shape parameter* (which is N_M in both cases) and a rate parameter (λ_{D_m} for T_D and λ_{P_m} for T_P). The CDF of an Erlang distribution with shape parameter k and rate parameter λ is given by:

$$F_{\text{Er}}(t, k, \lambda) = 1 - \sum_{n=0}^{k-1} \frac{1}{n!} e^{-\lambda t} (\lambda t)^n \tag{4.19}$$

This can be used to compute the delay probabilities as follows:

$$P(T_D \leq x) = 1 - \sum_{n=0}^{N_M - 1} \frac{1}{n!} e^{-\lambda_{D_m} x} (\lambda_{D_m} x)^n = F_{\text{Er}}(x, N_M, \lambda_{D_m}) \tag{4.20}$$

$$P(T_P \leq y) = 1 - \sum_{n=0}^{N_M - 1} \frac{1}{n!} e^{-\lambda_{P_m} y} (\lambda_{P_m} y)^n = F_{\text{Er}}(y, N_M, \lambda_{P_m}) \tag{4.21}$$

From Eq. 4.18, Eq. 4.20, and Eq. 4.21, we can state the required bound on T_S for TAP:

$$P(T_S \leq (x + y)) \geq F_{\text{Er}}(x, N_M, \lambda_{D_m}) \times F_{\text{Er}}(y, N_M, \lambda_{P_m}) \tag{4.22}$$

where $N_M = 4 \times R$, $\lambda_{D_m} = \lambda_{MTR} M$ and $\lambda_{P_m} = \mu_s - \lambda_a$.

We have mechanically verified all the equations described in this section in Athena by using theory from limits, distributions, real numbers, and other domains. However, version 0.1 of our library lacks the theory required to reason about Poisson distributions and graphical analysis. Therefore, we have taken Eq. 4.7 and Eq. 4.9 as conjectures for our formal proofs. Both these equations are established results from queuing theory that have been borrowed from Bertsekas *et al.* [16] and we plan to prove them in future work.

[16]The product rule states that for independent events A and B, $P(A \wedge B) = P(A) \times P(B)$.

4.3 A Formal Library for Reasoning About Distributed Algorithms

Autonomous decentralized operations of mobile agents are complex in nature. Such operations involve many interlinked aspects such as the mobility of the agents, the characteristics of the communication network, the internal computational capabilities of the agents, and the interaction between the agents. Therefore, to formally reason about such operations, it is necessary to holistically consider all these important aspects. Achieving this in a machine-verifiable manner requires access to formal constructs, that are sufficiently expressive to correctly and completely specify such aspects, in some machine-readable formal language. Developing such expressive formal constructs in a machine-readable language is a challenging task since it requires domain knowledge of all aspects of the system that needs to be specified, strong knowledge of formal logic and reasoning techniques, experience and familiarity with the language in which the constructs are to be specified, and a significant amount of time and effort. In the absence of a pre-existing library that provides the necessary formal constructs to express a system, every time high-level properties of the system need to be verified, engineers have to develop the required formalizations from the ground up, often making the task of verification expensive.

For verifying the correctness of distributed algorithms deployed over VANETs, it is oftentimes necessary to prove complex high-level properties of the algorithms. To specify such high-level timely progress properties for TAP using the theory of the MTR protocol and the M/M/1 queue system, we required theories in Athena to express constructs such as networks, mobility models, communication models, inter-agent interactions, queues, and distributed algorithms. For this reason, we have created the foundation for a formal library in Athena that is tailored towards reasoning about high-level probabilistic properties of timely progress for distributed algorithms deployed over VANETs[17].

Athena provides a language for "proof programming" which relies on a fundamental technique called *method call*. Method calls in Athena represent logical *inference rules* that can accept arguments of arbitrary types and are higher-order. Successful evaluation of a method call results in a *theorem*, which is then added to the *assumption base*. The assumption base is a set of sentences that have either been proven to be correct by using inference rules

[17]Complete Athena code available at https://wcl.cs.rpi.edu/assure

or have been asserted to be correct. Athena provides the *soundness* guarantee that any proven theorem will be a direct consequence of sentences in the assumption base.

Athena allows the use of *modules* that can be used to organize, partition, and encapsulate theories into multiple logically separate *namespaces*. Modules are dynamic in the sense that they can be extended at any time and additional content can be added to them using the `extend-module` command (Fig. 4.3). This feature allows theories, which are logically separate from one another, to be categorized and structured for brevity. It also makes it easy to generalize theories for use in different contexts by importing them as required.

```
# Module Prob being extended by module PrbIndRv
load "Athena_LibDDDAS/math/Prob/Prob.ath"
extend-module Prob {
  module PrbIndRv {
    # contents of module PrbIndRv
    ...
    }
```

Figure 4.3: Extending modules in Athena.

We have used our Athena library for distributed coordination algorithms to formalize and verify all the high-level properties and theorems mentioned in Section 4.2.3. For this purpose, we have formalized theories necessary to reason about distributed coordination algorithms in VANETs by adding constructs that can be used to specify the properties of interest. *E.g.*, we have created a domain of distributed algorithms called `DistProt` that has properties which represent the characteristics of the operational environment such as `dpTotTim` (the total time for progress), `msGetTd` (the message transmission delays) and `msGetTp` (the message processing delays). The Athena code for Eq. 4.16, which uses these constructs to define the probabilistic relationship between the total processing delay, the message transmission delays, and the message processing delays, is given in Fig. 4.4.

```
# The relationship between total time for progress and message delays
define THEOREM-P-TS>=P-TD&TP :=
(forall DP r1 r2 .
    ((Prob.probE  (Prob.consE Prob.<= (dpTotTim DP) (r1 + r2)))
      >=
      (Prob.probE
        (Prob.cons2E
          (Prob.consE Prob.<= (Random.SUM (msGetTd (dpGetMsgs DP))) r1)
          (Prob.consE Prob.<= (Random.SUM (msGetTp (dpGetMsgs DP))) r2)))))
```

Figure 4.4: The representation of Eq. 4.16 in Athena.

Similarly to `DistProt`, we have specified a domain of networks called `Network`, a domain of network protocols called `PrtType` that can represent protocols like MTR, a domain of queues called `Queue` to represent queuing models like M/M/1, and other specialized theories to reason about distributed algorithms deployed over VANETs.

```
# Expected value of message processing delays in M/M/1 queue
conclude THEOREM-mean-T-value
...
let{
    ...
  N_d :=
    ( (Dist.ratePar (Random.pdf (srvcTm Q))) # mu_s
      -
      (Dist.ratePar (Random.pdf (cstArRat Q))) # 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]
    ])
```

Figure 4.5: A snippet from the proof of Eq. 4.10 in Athena.

Athena supports the use of *tactics* for interactively guiding the proof development of higher-level proof obligations from lower-level facts in the assumption base. There are built-in methods in Athena that can be used to implement tactics for both *forward* and *backward* reasoning. *E.g.,* Fig. 4.5 shows the use of the built-in `chain` method for deriving the proof of Eq. 4.10 using forward reasoning, where `T` represents T_p and `N_d` represents $\mu_s - \lambda_a$. In the future, we aim to define specialized methods that can be used as tactics for reasoning about distributed algorithms.

```
# The relationship between cdf and probability
define cdf-prob-conjecture :=
(forall x R . ((Random.cdf x R) = (probE (consE <= x R))))
```

Figure 4.6: The relationship between CDF and probability defined as a conjecture in Athena.

One of our goals while developing the library in Athena was reusability. As the proofs of different independent high-level properties are often dependent on common fundamental theories, it becomes beneficial to design the common theories in a manner that makes them reusable in different contexts. *E.g.,* the statement `cdf-prob-conjecture` (Fig. 4.6), which defines that the CDF $G_X(x)$ of a random variable X is the probability that $X \leq x$ for all reals x, has been used in various contexts throughout our formalization of the results presented in Section 4.2 (Fig. 4.7).

```
# Proof of P(min[X1,X2,...] <= y) = 1 - (1- G(X))^N
conclude THEOREM-probability-MIN-<=-IID-RVS-Gx
...
 conn-to-cdf-prob := (!uspec (!uspec cdf-prob-conjecture (Random.rvSetIdElmnt rvSet)) R)
...


# Probabilistic bound on customer delay for M/M/1 queue
conclude THEOREM-cstDly-prob
...
 conn-2-cdf-prob-conjecture := (!uspec (!uspec Prob.cdf-prob-conjecture (cstDly Q)) r)
...
```

Figure 4.7: Some instances where the `cdf-prob-conjecture` has been reused.

Version 0.1 of our Athena library for distributed algorithms consists of 71 axioms and 27 theorems, including the high-level properties of distributed algorithms and other domains described in this chapter. It is composed of theories from four primary domains — general mathematics, probability, networks, and distributed systems. Within each domain, there are theories from various related sub-domains. *E.g.,* under the domain of networks, we have theories from VANETs and queues. Fig. 4.8 depicts the hierarchy of version 0.1 of our Athena library and the dependencies between the various modules representing the different logical domains and sub-domains. As our primary goal was to develop high-level probabilistic progress properties of TAP that can be directly useful for UAM, we adopted a *top-down* approach of proof development where we only developed the lower-level theories that were required to support the higher-level properties of interest to us.

During the development of our high-level formal guarantees of timely progress for TAP in Athena, we have tried to use only fundamental facts from the various domains as axioms However, in version 0.1 of our library, there are instances where we have taken some well-established results from the domains as conjectures which we have not formally verified in Athena (*e.g.* – Eq. 4.9, `cdf-prob-conjecture`). We believe that this does not invalidate the results we have presented in the chapter or diminish their importance in any way. This is because all the conjectures we have used are well-established facts in the literature of the respective domains and have been informally proven beyond a reasonable doubt. Since our primary goal was to verify the higher-level properties of timely progress for TAP that we have presented in Section 4.2.3, in the interest of time, we decided to take some of these well-established theories as foundations for developing our higher-level guarantees because version 0.1 of our library lacks the necessary formal theory to mechanically verify them. We

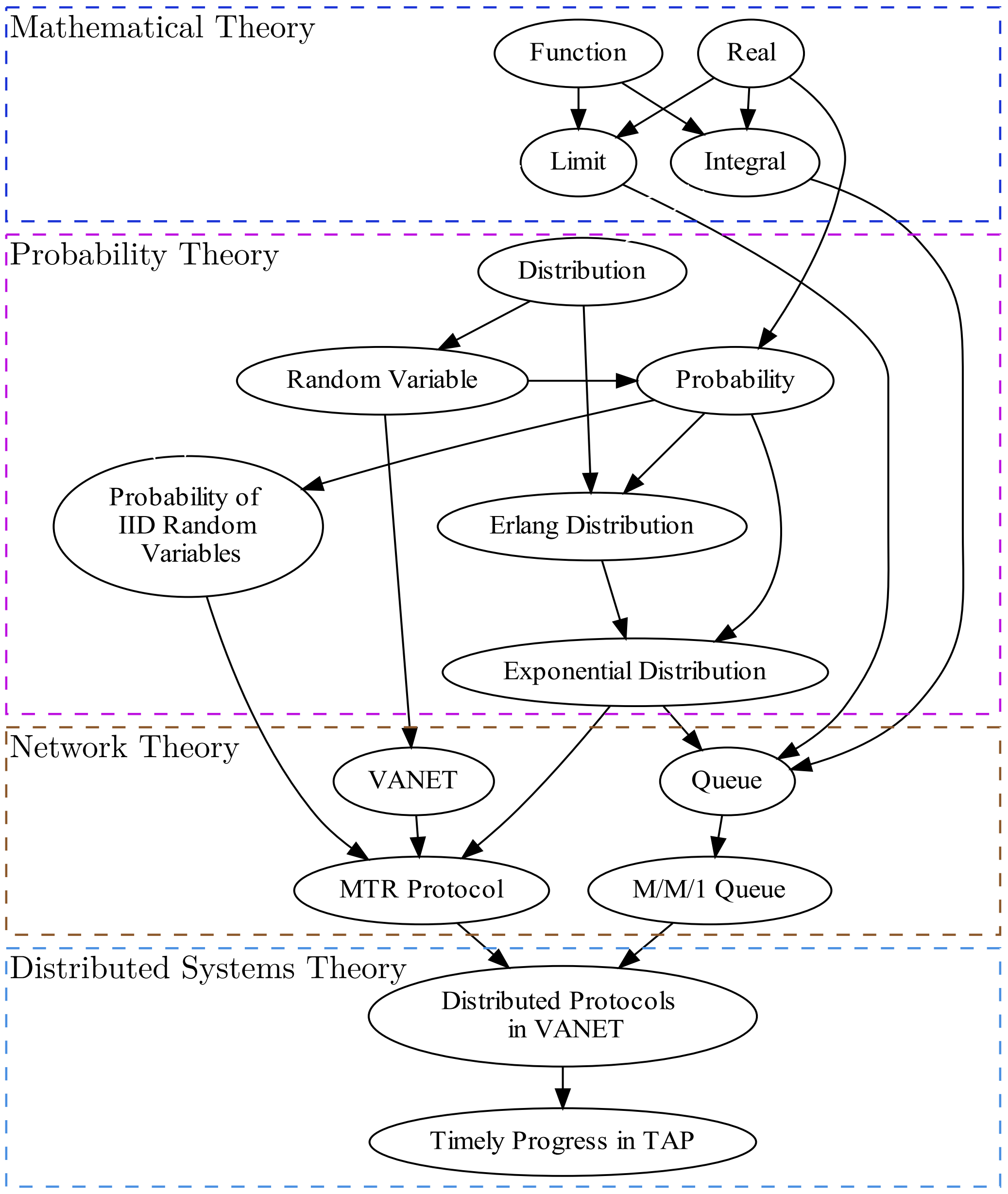


Figure 4.8: Hierarchy of version 0.1 of our Athena library for reasoning about distributed algorithms deployed over VANETs.

aim to develop the proofs of these theories in Athena from just the fundamental axioms of the corresponding domains in the future.

We have developed most of the symbols, domains, and relationships necessary to comprehensibly express all the assumptions and specifications using the capabilities of Athena's many-sorted first-order logic. We have tried to make the specifications reusable to different contexts so that further expansion of the library can be aided by using the existing specifications as building blocks rather than requiring to redevelop them for use in other contexts. 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 developed theories into appropriate categories to make it easy to import them independently in different contexts. This is important because when theories are imported during proof development, they are automatically added to the assumption base. Well-defined modules allow theories to be imported only as needed without making the assumption base unnecessarily large. This is an evolving project and we are certain that as more theory is added to the library, many of the formalizations in version 0.1 of the library will need to be redesigned in order to make them more general and better organized.

To the best of our knowledge, at the time of publication, our Athena library for distributed algorithms is the only library that provides a tailored formal foundation for reasoning about autonomous distributed coordination in VANETs based on the theory of relaying and queuing systems. Therefore, we believe that our contribution will make the task of reasoning about properties of other distributed coordination algorithms, which are implemented using the MTR protocol and the M/M/1 queue system, in Athena easier in the future as a significant amount of reusable formal constructs to express useful properties have already been developed. Nonetheless, additional theories will be required to reason about distributed algorithms involving a non-deterministic number of messages (*e.g.* – the Synod consensus protocol [136]). Our eventual goal is to expand the library by adding additional theories from distributions, queues, vehicular networks, message sequences, and other necessary domains to support reasoning about a variety of distributed algorithms that can be employed for autonomous decentralized UAM applications like DAC.

4.4 Related Work

There is existing work in the literature on the informal analysis of timely progress of consensus. Attiya *et al.* [10] provided lower-bounds on the time for progress in *round-based* [44] consensus protocols. Attiya *et al.* [9] analyzed the time complexity of solving distributed decision problems. Berman *et al.* [15] studied the number of message rounds involved in various consensus protocols.

Existing work on machine-verified guarantees has mainly analyzed eventual progress.

McMillan *et al.* [115] have proven eventual progress of *Stoppable Paxos* [114] in Ivy [134]. Dragoi *et al.* [47] and Debrat *et al.* [42] have proven eventual progress in *LastVoting* [32]. Hawblitzel *et al.* [82, 83] have proven eventual progress for a *Multi-Paxos* implementation using Dafny [108]. In [136], we have mechanically verified eventual progress in the Synod consensus protocol using Athena.

Formalization of mathematical theories also exists in the literature. Hasan [75] presented formalizations of statistical properties of discrete random variables in the *HOL Theorem Prover* [67]. Hasan *et al.* have formalized properties of the standard uniform random variable [76], discrete and continuous random variables [77, 79], tail distribution bounds [78], and conditional probability [80]. Mhamdi *et al.* [120, 121] have extended Hasan *et al.*'s work by formalizing *measure theory* and the *Lebesque integral* in HOL. HOL formalizations of the Poisson process, *continuous chain Markov process*, and M/M/1 queue have been presented in [29]. Qasim [146] has used Mhamdi *et al.*'s work to formalize the *standard normal variable.*

In contrast to the existing work, we have developed formal theories tailored towards machine-verifiable timely progress guarantees for distributed algorithms using models appropriate for VANETs and have presented the timely progress guarantee of a knowledge propagation protocol that can be used for autonomous decentralized UAM applications.

4.5 Chapter Summary

In this chapter, we have presented an approach for developing formal probabilistic properties of timely progress for distributed algorithms deployed over VANETs by using theories from the Multi-copy Two-Hop Relay protocol and the M/M/1 queue system to reason about the non-deterministic message delays. Since probabilistic progress properties provide useful probabilistic bounds on the time that may be required to make progress, they are more useful than simply guarantees of eventual progress for time-critical UAM applications like DAC. We have also showcased the development of a formal library in Athena that is tailored towards mechanically reasoning about probabilistic properties of distributed coordination algorithms deployed over VANETs. In the future, autonomous UAM applications will require participating aircraft to employ different types of distributed algorithms for achieving different operational goals such as maintaining safe separation, autonomously coordinating operations through a shared airspace, etc. Our proof library has, therefore, been tailored to be reusable for reasoning about the probabilistic properties of a variety of distributed algorithms that can be used for such autonomous UAM applications.

Version 0.1 of our Athena library has some high-level conjectures since the version lacks the theories required to formally verify them. In the future, we plan to formalize the necessary fundamental theories required to prove these conjectures as theorems. The library also does not support the complete formalization of timely progress if the number of messages is non-deterministic. Therefore, another potential direction of future work is to add additional theories to the library to allow reasoning about distributed algorithms that involve a non-deterministic number of messages, such as the Synod consensus protocol. In the future, we would also like to model other routing protocols suitable for VANETs in addition to MTR so that appropriate guarantees can be provided depending on actual implementations.

CHAPTER 5
DATA-DRIVEN RUNTIME VERIFICATION OF DISTRIBUTED COORDINATION ALGORITHMS

5.1 Chapter Introduction

Distributed algorithms, that are deployed over VANETs like the Internet of Planes for applications such as Decentralized Admission Control, are expected to operate under highly uncertain and dynamic conditions. In Chapter 4, we had shown how formal guarantees of probabilistic properties can be developed for such algorithms using theoretical models. However, during operation, the uncertain factors affecting the probabilistic properties, such as the message delays, may not conform to the initial theoretical assumptions. Under such circumstances, formal proofs of guarantees that have been developed in the pre-deployment stages may cease to be relevant after deployment as the assumptions made during proof development may no longer be true at runtime. This creates the need for developing techniques that can allow mechanically verified formal proofs of the correctness properties of distributed algorithms to be useful in the face of dynamic operating conditions at runtime.

In the presence of dynamic operating conditions that violate theoretical assumptions, data-driven statistical observations can be used to provide useful formal guarantees for distributed algorithms at runtime. This can be done by monitoring the operating conditions at runtime to learn about possible violations of existing machine-checked properties and then generating new formal properties that are true with respect to the real-time observations. However, several significant challenges must be overcome to successfully achieve such theorem proving-based runtime verification of distributed algorithms. Firstly, to detect violations of the existing formal guarantees, the operating conditions must be analyzed accurately at runtime with respect to the assumptions of the latest proofs. Secondly, it must be possible to dynamically generate new formal properties at runtime, that are relevant with respect to the real-time conditions and are machine verifiable. Addressing these challenges is a difficult task because developing machine-verifiable formal proofs of non-trivial properties takes a considerable amount of skill, expertise, and time. Therefore, it is not feasible to develop

Portions of this chapter previously appeared as: Paul, S., Kopsaftopoulos, F., Patterson, S., Varela, C.A.: Dynamic data-driven formal progress envelopes for distributed algorithms. In: Dynamic Data-Driven Application Systems (DDDAS), pp. 245-252 (2020). doi:10.1007/978-3-030-61725-7_29 them from scratch at runtime to keep up with the dynamic operating conditions.

In this chapter, we propose some novel techniques to aid in the theorem proving-based verification of dynamic distributed applications. First, we propose *formal progress envelopes* for distributed algorithms that can be used to discretize the operational state space into distinct regions where a formal proof of progress may or may not hold. Next, we introduce *parameterized proofs*, a class of formal proofs that can be instantiated at runtime with runtime-observable parameters which represent the real-time operating conditions. Finally, using progress envelopes and parameterized proofs, we propose a DDDAS-based approach that can be used to generate new formal guarantees at runtime. This approach utilizes a *formal DDDAS feedback loop* to instantiate mechanically verified parameterized proofs at runtime, thereby extending the practicality of mathematically rigorous formal theorem proving techniques beyond the pre-deployment stages of dynamic distributed applications.

The rest of the chapter is structured as follows: Section 5.2 describes formal progress envelopes; Section 5.3 introduces parameterized proofs; Section 5.4 presents our DDDAS-based approach for the runtime verification of distributed algorithms; Section 5.5 discusses related work; and Section 5.6 concludes the chapter with a discussion on future work.

5.2 Formal Progress Envelopes for Distributed Algorithms

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. 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 99.999999% 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 $[t_0, t_1]$ but follow an exponential distribution with rate parameter λ in $[t_1, t_2]$. Therefore, we can say that the first interval is outside the progress envelope while the second interval is inside the envelope (Fig. 5.1).

Since the parameters that define formal progress envelopes can be quantified and measured, the envelopes can be analyzed against real-time data. A *correctness sentinel* is a runtime-accessible program that can monitor real-time data-streams to detect if the data

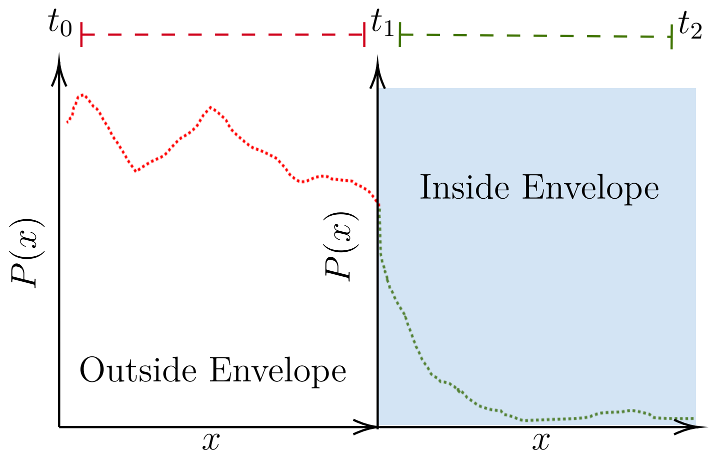


Figure 5.1: A simple binary progress envelope for message delays x.

conforms to progress envelope constraints [21]. Like any software for safety-critical systems, the sentinels must be verified to ensure that they can monitor the real-time data correctly. To ensure their correctness, sentinels can be generated directly from the formal specifications of the envelopes and some underlying physical models of uncertainty or be designed in a *domain-specific language* amenable to verification.

5.3 Parameterized Formal Proofs

Mathematically rigorous formal proofs of correctness can be developed manually in the pre-deployment stages of dynamic distributed systems. In order to make these proofs useful at runtime, it is possible to design them as functions of the progress envelope parameters that can be quantified and measured in real-time. *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 99.99999%*", 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 CDF 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 instantiated with real-time 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 real-time observations.

5.4 Runtime Verification of Distributed Algorithms Using DDDAS

The DDDAS paradigm allows a system to dynamically incorporate new data to enhance an existing model [40]. DDDAS enables an application to influence the data collection and measurement processes in real-time and give way to more effective collection and measurement of data, thus leading to a better quality of data specifically suited for the application. This paradigm involves the use of a *feedback loop* which is used to update an existing application model with better quality data in real-time to create a more accurate model. We present a DDDAS-based approach for the runtime instantiation of parameterized proofs with real-time observations to generate new formal guarantees at runtime.

5.4.1 Formal DDDAS Feedback Loop

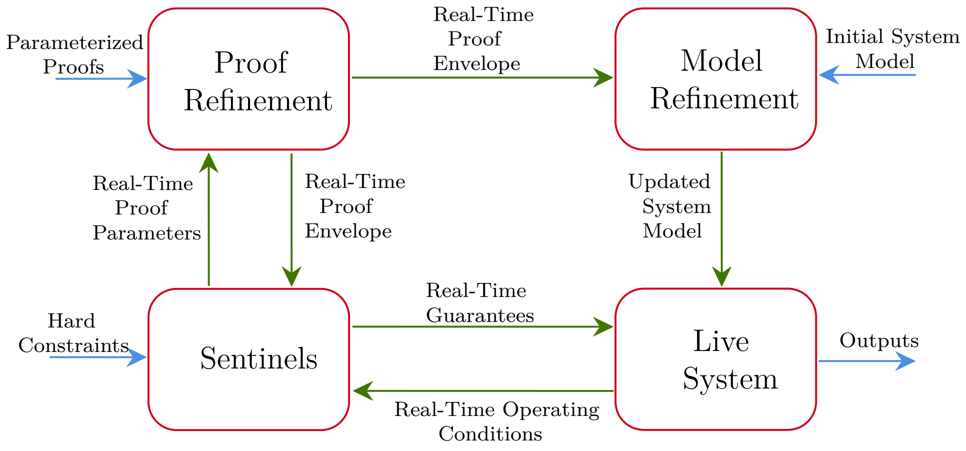


Figure 5.2: Formal DDDAS feedback loop.

Our formal DDDAS feedback loop consists of the following components (Fig. 5.2):

- The *proof refinement* component receives a parameterized proof with a set of initial parameters and real-time inputs from the sentinels. If possible, it refines the proofs with the real-time parameters and provides new envelopes.

- The *model refinement* component receives real-time 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 real-time operating conditions of the system against the latest envelopes. If the conditions do not conform to the envelopes, they send real-time parameters to the proof refinement component and inform the *live system* about guarantees that hold in real-time.

- 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 systems that are dynamic in nature, this feedback loop can be used by developing suits 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.

5.4.2 Simulation of the Feedback Loop

To illustrate the working of the formal DDDAS feedback loop, we developed a simple simulation with synthesized data[18]. Details of the simulation setup are given below:

- The proof refinement component has the following pre-proven parameterized property:

  The probability that a random variable X following a Gaussian distribution f_X will take values of at most v is $\int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$, where $z = \frac{v-\mu_X}{\sigma_X}$ is the standard score and μ_X and σ_X are the mean and standard deviation of f_X.

- The runtime-observable parameter is the one-way message transmission delay T_{P_m}.

- The live system is concerned with finding the round-trip delay $2 \times T_{P_m}$.

- The model refinement component chooses the value to use for T_{P_m} as a model for the live system's calculations.

- The sentinel monitors an incoming stream of synthesized transmission delays at runtime. The parameters monitored are the mean and standard deviation, and a check for normality. The hard constraint for the simulation is $P(T_{P_m} \leq 15 \text{ ms}) \geq 99.9\%$.

[18]A video of the simulation can be found at https://wcl.cs.rpi.edu/assure

The simulation proceeded as follows (Fig. 5.3):

1. Initially, there was an assumption that T_{P_m} follows a Gaussian distribution f_T with $\mu_T = 5$ and $\sigma_T = 1$.

    - The proof refinement component used this to calculate two initial properties:

      A $P(T_{P_m} \leq 10 \text{ ms}) = 99.99997\%$

      B $P(T_{P_m} \leq 15 \text{ ms}) = 99.99999\%$

    - As $P(T_{P_m} \leq 10 \text{ ms}) > 99.9\%$, the model refinement component selected $T_{D_m} = 10$ ms and the live system computed a round-trip delay of 20 ms.

2. The synthesized data was augmented to increase σ_T.

    - The change in σ_T was detected by the sentinel and it sent the new value of $\sigma_T \approx 3.1$ to the proof refinement component.
    - The proof refinement component recomputed the properties A and B as follows:

      A $P(T_{P_m} \leq 10 \text{ ms}) = 94.41874\%$

      B $P(T_{P_m} \leq 15 \text{ ms}) = 99.92945\%$

    - Now, the model refinement component selected $T_{D_m} = 15$ ms and the live system computed a round-trip delay of 30 ms.

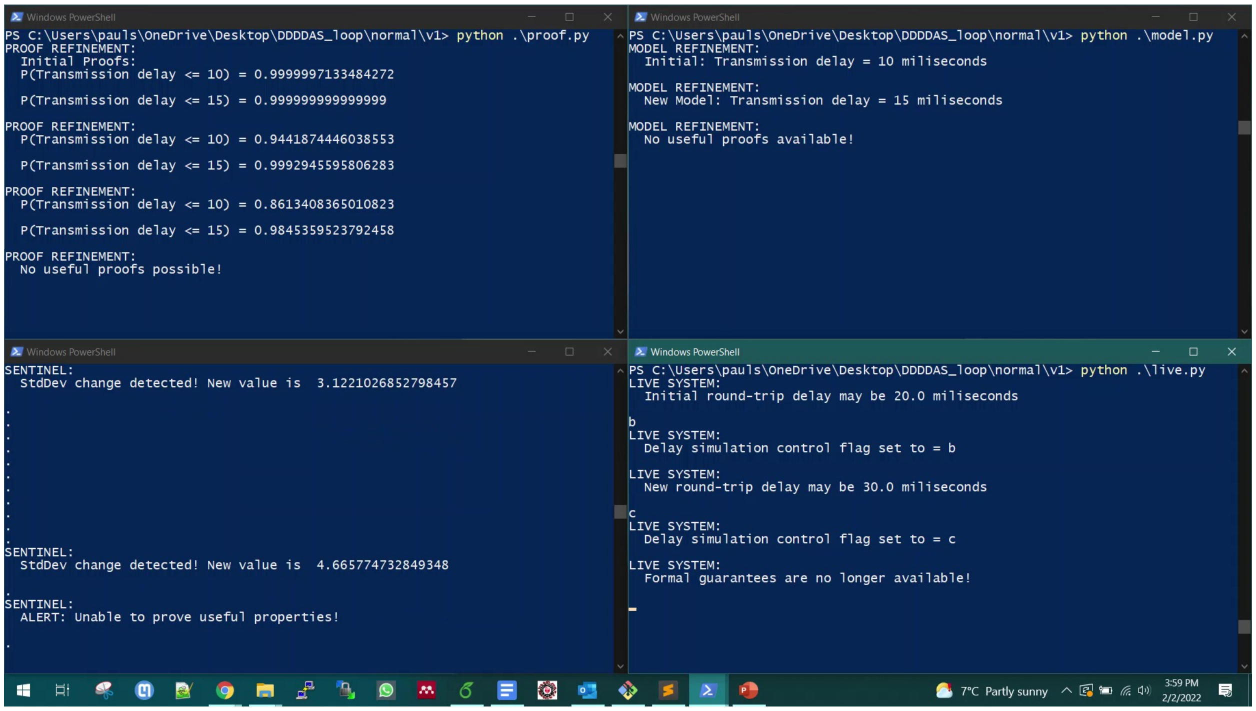
3. We further augmented the synthesized data to increase σ_T.

    - The sentinel detected the new value $\sigma_T \approx 4.7$ to the proof refinement component.
    - The proof refinement component recomputed the properties A and B as follows:

      A $P(T_{P_m} \leq 10 \text{ ms}) = 86.13408\%$

      B $P(T_{P_m} \leq 15 \text{ ms}) = 98.45359\%$

    - The sentinel detected the violation of the hard constraint and responded to the live system with a message saying that useful proofs could no longer be provided.



```
PS C:\Users\pauls\OneDrive\Desktop\DDDDAS_loop\normal\v1> python .\proof.py
PROOF REFINEMENT:
  Initial Proofs:
  P(Transmission delay <= 10) = 0.9999997133484272

  P(Transmission delay <= 15) = 0.999999999999999

PROOF REFINEMENT:
  P(Transmission delay <= 10) = 0.9441874446038553

  P(Transmission delay <= 15) = 0.9992945595806283

PROOF REFINEMENT:
  P(Transmission delay <= 10) = 0.8613408365010823

  P(Transmission delay <= 15) = 0.9845359523792458

PROOF REFINEMENT:
  No useful proofs possible!
```

```
PS C:\Users\pauls\OneDrive\Desktop\DDDDAS_loop\normal\v1> python .\model.py
MODEL REFINEMENT:
  Initial: Transmission delay = 10 miliseconds

MODEL REFINEMENT:
  New Model: Transmission delay = 15 miliseconds

MODEL REFINEMENT:
  No useful proofs available!
```

```
SENTINEL:
  StdDev change detected! New value is  3.1221026852798457

.
.
.
.
.
.
.
.
.
.
.
SENTINEL:
  StdDev change detected! New value is  4.665774732849348

.
SENTINEL:
  ALERT: Unable to prove useful properties!

.
```

```
PS C:\Users\pauls\OneDrive\Desktop\DDDDAS_loop\normal\v1> python .\live.py
LIVE SYSTEM:
  Initial round-trip delay may be 20.0 miliseconds

b
LIVE SYSTEM:
  Delay simulation control flag set to = b

LIVE SYSTEM:
  New round-trip delay may be 30.0 miliseconds

c
LIVE SYSTEM:
  Delay simulation control flag set to = c

LIVE SYSTEM:
  Formal guarantees are no longer available!
```

Figure 5.3: Screenshot from the formal DDDAS feedback loop simulation.

5.5 Related Work

Runtime monitoring of formal properties has been previously investigated. Mitsch and Platzer [122] have presented ModelPlex, a runtime validation tool that can compare the behavior of a system observed at runtime against a verified model and detect non-compliance. Pike *et al.* [142] present Copilot, a language and compiler that can be used to generate runtime monitors for real-time distributed systems. Dutle *et al.* [49] have used Copilots to generate executable runtime monitors from temporal logic specifications that have been used in NASA's ICAROUS [38] project. Bocchi *et al.* [19] propose a runtime monitoring discipline for distributed systems using *session types* that allows processes to be independently verified, either statically during deployment, or dynamically during execution. Formal safety envelopes for stochastic state identification have been proposed in [21] and [39] that can be analyzed against real-time data using *safety sentinels.*

Our work improves upon previous work by proposing techniques to extend the practicality of machine verifiable theorem proving methods beyond the pre-deployment stages of dynamic distributed applications. This allows the generation of formally rigorous guarantees of correctness at runtime when the dynamic operating conditions do not conform to the theoretical models used during proof development.

5.6 Chapter Summary

In this chapter, we have presented a DDDAS-based technique for the runtime verification of dynamic distributed applications that may not conform to theoretical models during operation. Our approach allows the instantiation of mechanically verifiable parameterized proofs at runtime using a formal DDDAS feedback loop. The new guarantees generated are, therefore, logically sound. This allows the mathematical rigor of formal theorem proving methods to be useful beyond the pre-deployment stages of dynamic distributed applications.

Real-life data is seldom perfect – *e.g.*, normality of sensor data may only be tested up to some significant level and may also be affected by pre-processing. It is also computationally expensive to effectively estimate the tail bounds of distributions in real-time. Future directions of work, therefore, include investigating the development of formally-verified runtime sentinels that can find accurate and meaningful estimates from data. Another important direction is the extensive experimental analysis of the formal DDDAS feedback loop to evaluate its efficiency with regard to real-life applications.

CHAPTER 6
A CONFLICT-AWARE FLIGHT PLANNING APPROACH FOR DECENTRALIZED ADMISSION CONTROL

6.1 Chapter Introduction

For our Decentralized Admission Control approach, the candidate aircraft must have the capability to detect potential conflicts with the flight plans of the owners and then compute proposal flight plans that are free from conflicts. Since aircraft flight plans can be composed of non-trivial flight paths, such as a collection of four-dimensional waypoints, existing technologies like the Traffic Collision Avoidance System [57], which can detect pairwise conflicts in straight line flight paths and recommend tactical avoidance maneuvers, cannot be used for DAC. Other approaches for strategic conflict management such as the distributed consensus-based merging technique for UAS presented by Balachandran *et al* [12] can be applicable for DAC, but their technique has not been formally verified. Colbert *et al.* [37] have presented *PolySafe*, a formally verified strategic algorithm that can be used to detect conflicts between polynomial trajectories, but this approach cannot be used to generate conflict-free flight plans for DAC. This creates the need for developing a formally verified strategic conflict detection and avoidance approach that can be used by the candidates in DAC to compute conflict-free proposals with respect to the owners.

In this chapter, we present a formally verified approach for strategic conflict-aware flight planning. We define a flight plan as a tuple consisting of a start time and an ordered collection of *waypoints* in three-dimensional space that an aircraft is expected to follow. Each waypoint has an associated configuration that expresses the aircraft's position, velocity, and the expected course of flight to the next waypoint. Given a set of flight plans for traffic aircraft, a set of waypoints, an initial flight plan for the *ownship*, and a set of discrete values of allowed *ground speed* for the ownship, our approach generates, when feasible, a conflict-free flight plan for the ownship with respect to the traffic flight plans. Conflict detection is based on the assumption that aircraft follow constant-velocity, straight-line segments between consecutive waypoints and that the set of flight plans for traffic aircraft is immutable.

Portions of this chapter previously appeared as: Paul, S., Patterson, S., Varela, C.A.: Conflict-aware flight planning for avoiding near mid-air collisions. In: IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 1-10 (2019). doi:10.1109/dasc43569.2019.9081658

In our approach, the 3D heading of the velocity vector between waypoints is not changed and the only changes observed in the solution flight plan are the times of arrival for the waypoints. To generate a conflict-free plan for the ownship, suitable values of ground speed are assigned to the flight segments between each consecutive pair of waypoints in the ownship's initial plan. This can be advantageous in situations where spatial deviation from the original waypoints is restricted by lateral and/or vertical constraints that may be imposed by terrain, restricted airspace, weather, or airspace capacity. This is usually true for terminal areas where aircraft are appointed *standard terminal arrival route*s (STAR) just before the final approach and urban environments that usually provide little room for lateral and vertical maneuvers. Our approach is therefore highly suitable for UAS integration into the urban air-traffic management airspace using DAC.

The rest of the chapter is structured as follows: Section 6.2 describes the problem statement for strategic conflict-aware flight planning and our algorithm for computing a conflict-free flight plan; Section 6.3 presents the formal correctness properties of our approach; Section 6.4 describes simulation-based evaluations of our approach; Section 6.5 discusses prior work on conflict detection and avoidance techniques; and Section 6.6 concludes the chapter with a summary and potential future directions of work.

6.2 Conflict-Aware Flight Planning

In this section, we present the formal foundation of our conflict-aware flight planning approach. We introduce important definitions and terms, the problem statement, and the strategy for computing valid conflict-aware flight plans. Our geometrical model of the conflict detection problem is based on a *flat-Earth assumption* with a global frame of reference. We also assume that aircraft are capable of sending and receiving multi-segment flight plans with configuration information for each waypoint.

Table 6.1: Semantic domains used in specifying the flight planning problem.

Type	**Specification**	**Set**	**Domain**
Configuration	$\langle s_x, s_y, s_z, v_g, \lambda, v_z \rangle$	$\mathbb{C}$	$\mathbb{R}^3 \times \mathbb{R} \geq 0 \times [0, 2\pi) \times \mathbb{R}$
State	$\langle s_x, s_y, s_z, v_x, v_y, v_z \rangle$	$\mathbb{S}$	$\mathbb{R}^6$
Flight plan	$\langle t, \mathbb{C}^n \rangle$	$\mathbb{F}$	$\mathbb{R} \geq 0 \times \mathbb{C}^{n \in \mathbb{N}}$

6.2.1 Definitions of Some Important Terms

- The *configuration* of an aircraft is a vector $\langle s_x, s_y, s_z, v_g, \lambda, v_z\rangle$ where s_x, s_y, and s_z are the coordinates in the x, y, and z dimensions, v_g is the ground speed, λ is the course or actual direction of motion of the aircraft with respect to the ground, and v_z is the vertical speed. The set $\mathbb{C}$ is the set of all configurations (Table. 6.1).

- The *state* of an aircraft is a vector $\langle s_x, s_y, s_z, v_x, v_y, v_z\rangle$ where s_x, s_y, and s_z are the coordinates and v_x v_y, and v_z are the components of the velocity vector in the x, y, and z dimensions. The set $\mathbb{S}$ is the set of all states (Table. 6.1).

- A *waypoint* is a position in 3D space and is represented by $\langle s_x, s_y, s_z\rangle$ where s_x, s_y, and s_z are the coordinates in the x, y, and z dimensions.

- A *segment* is the 3D straight-line flight segment between two consecutive waypoints. A segment is represented by its initial and final waypoints.

- A *flight plan* is a pair $\langle t, \mathbb{C}^n\rangle$ containing a start time t and a vector of configurations $\mathbb{C}^n$ that represent n waypoints in the flight plan. Two consecutive waypoints in a flight plan are connected by a segment. In a flight plan with n waypoints, there are always $n-1$ segments. The set $\mathbb{F}$ is the set of all flight plans (Table. 6.1).

6.2.2 Detecting Conflict Between Two Flight Plans

Given the states of two aircraft a and b at time t_0 as S_{a,t_0} and S_{b,t_0}, their 2D horizontal position vectors and 2D horizontal velocity vectors can be computed as as $\mathbf{s}_{xy,a,t_0}$ and $\mathbf{s}_{xy,b,t_0}$.

$$S_{a,t_0} = \langle s_{x,a,t_0}, s_{y,a,t_0}, s_{z,a,t_0}, v_{x,a,t_0}, v_{y,a,t_0}, v_{z,a,t_0}\rangle \tag{6.1}$$

$$S_{b,t_0} = \langle s_{x,b,t_0}, s_{y,b,t_0}, s_{z,b,t_0}, v_{x,b,t_0}, v_{y,b,t_0}, v_{z,b,t_0}\rangle \tag{6.2}$$

$$\mathbf{s}_{xy,a,t_0} = \langle s_{x,a,t_0}, s_{y,a,t_0}\rangle \tag{6.3}$$

$$\mathbf{s}_{xy,b,t_0} = \langle s_{x,b,t_0}, s_{y,b,t_0}\rangle \tag{6.4}$$

$$\mathbf{v}_{xy,a,t_0} = \langle v_{x,a,t_0}, v_{y,a,t_0}\rangle \tag{6.5}$$

$$\mathbf{v}_{xy,b,t_0} = \langle v_{x,b,t_0}, v_{y,b,t_0}\rangle \tag{6.6}$$

The relative position and relative velocity of the two aircraft in the horizontal (xy) and vertical dimensions (z) at time t_0 can now be obtained by the following equations:

$$\mathbf{s}_{xy,t_0} = \mathbf{s}_{xy,a,t_0} - \mathbf{s}_{xy,b,t_0} \tag{6.7}$$

$$\mathbf{v}_{xy,t_0} = \mathbf{v}_{xy,a,t_0} - \mathbf{v}_{xy,b,t_0} \tag{6.8}$$

$$s_{z,t_0} = s_{z,a,t_0} - s_{z,b,t_0} \tag{6.9}$$

$$v_{z,t_0} = v_{z,a,t_0} - v_{z,b,t_0} \tag{6.10}$$

Assuming that both aircraft will maintain constant velocity flights, their relative horizontal and vertical positions at any time $t \geq t_0$ can be computed as:

$$\mathbf{s}_{xy,t} = \mathbf{s}_{xy,t_0} + (t - t_0)\mathbf{v}_{xy,t_0} \tag{6.11}$$

$$s_{z,t} = s_{z,t_0} + (t - t_0)v_{z,t_0} \tag{6.12}$$

We will be using $\mathbf{s}$ and $\mathbf{v}$ to denote $\mathbf{s}_{xy}$ and $\mathbf{v}_{xy}$ in the rest of the chapter.

The *well-clear volume* (WCV) [107] is defined as a cylinder of diameter D and height H around the center of an aircraft, where D is the horizontal separation threshold and H is the vertical separation threshold (Fig. 6.1). Two aircraft a and b are said to be in a *conflict* or an NMAC at time t if their well-clear volumes intersect (Fig. 6.2), *i.e.*, their horizontal distance is less than D and vertical distance is less than H simultaneously at time t. Therefore, if there is a conflict at time t, both Eq 6.13 and Eq 6.14 are satisfied for t:

$$\|\mathbf{s_t}\| < D \tag{6.13}$$

$$|s_{z,t}| < H \tag{6.14}$$

Conflict detection by using Eq 6.13 and Eq 6.14 is applicable 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 are expected to maintain constant-velocity flights. Given two flight plans F_a and F_b, it is, therefore, necessary to discretize the temporal dimension into discrete periods of interest. This can be done by creating a non-decreasing vector T

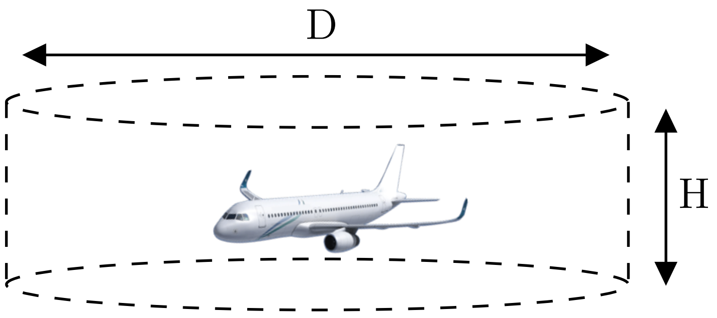


Figure 6.1: Well-clear volume around an aircraft.

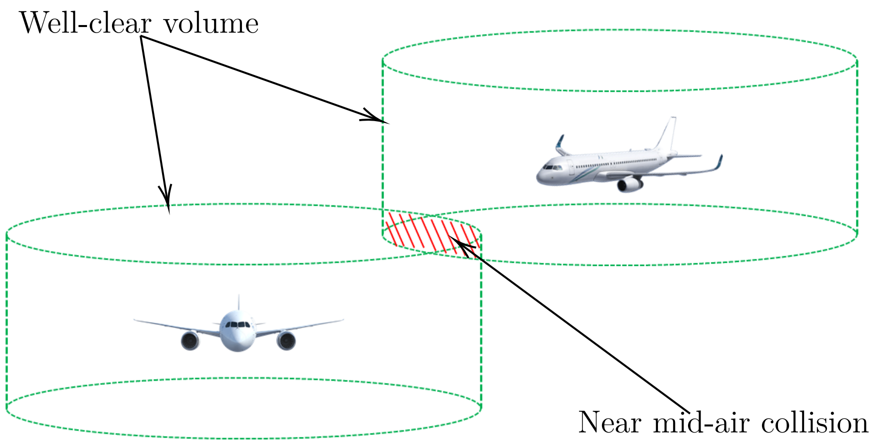


Figure 6.2: An NMAC between two aircraft.

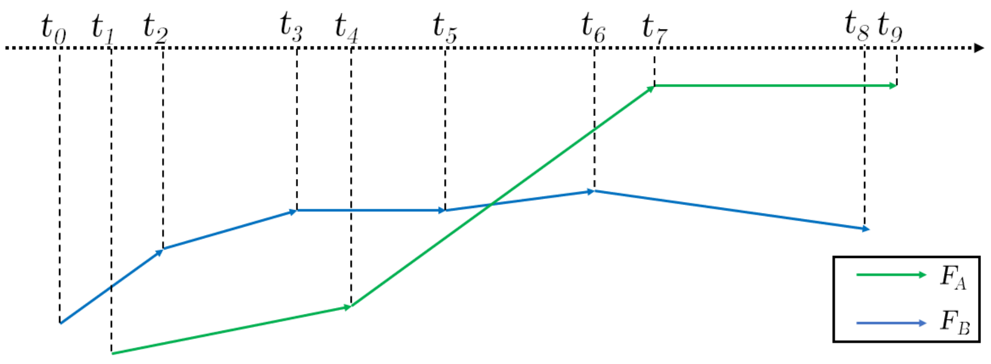


Figure 6.3: Periods of interest for two flight plans.

(which ranges over $\mathbb{R}^n$) of the times corresponding to the waypoints comprising the flight plans. The times between consecutive temporal points in T can then be used as discrete periods of interest for conflict detection. *E.g.,* in Fig. 6.3, which shows the top-view of two flight plans, the times corresponding to the waypoints in F_a are t_1, t_4, t_7, and t_9, and the times corresponding to the waypoints in F_b are t_0, t_2, t_3, t_5, t_6, and t_8. The non-decreasing vector of times can be written as $T = \langle t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9 \rangle$. Each pair of consecutive times in T has been used to discretize the temporal dimension (represented by the dotted line) into periods of interest. It should be noted that periods of interest are only valid if two flight plans have temporal overlap during that period. For example, in Fig. 6.3, the periods t_0 to t_1 and t_8 to t_9 are not valid periods of interest.

Table 6.2: Functions used in the specification of our flight planning protocol.

Function	**Input**	**Output**
`check-safety`	$\{0,1\}^{n\times m} \times \mathbb{F} \times \mathbb{F}^k \times \mathbb{R}^m \times \mathbb{R}^2$	`Bool`
`complete`	$\{0,1\}^{n\times m}$	`Bool`
`conflict`	$\mathbb{F}^2 \times \mathbb{R}^2$	`Bool`
`exists-violation`	$\mathbb{R}^2 \times \mathbb{F}^2 \times \mathbb{R}^2$	`Bool`
`get-all-times`	$\mathbb{F}^2$	$\mathbb{R}^n$
`get-config`	$\mathbb{R} \times \mathbb{F}$	$\mathbb{C}$
`get-index`	$\mathbb{R}^m \times \mathbb{R}$	$\mathbb{N}$
`plan`	$\{0,1\}^{n\times m} \times \mathbb{F} \times \mathbb{R}^m$	$\mathbb{F}$
`safe`	$\mathbb{F}^k \times \mathbb{R}^2$	`Bool`
`set`	$\{0,1\}^{n\times m} \times \mathbb{N}^2$	$\{0,1\}^{n\times m}$
`solve`	$\{0,1\}^{n\times m} \times \mathbb{N} \times \mathbb{F} \times \mathbb{F}^k \times \mathbb{R}^m \times \mathbb{R}^2$	$\mathbb{F}$
`T-violation`	$\mathbb{R}^n \times \mathbb{F}^2 \times \mathbb{R}^2$	`Bool`
`unassigned`	$\{0,1\}^{n\times m}$	$\mathbb{N}$
`valid`	$\{0,1\}^{n\times m} \times \mathbb{N} \times \mathbb{F} \times \mathbb{F}^k \times \mathbb{R}^m \times \mathbb{R}^3$	`Bool`

Given two flight plans F_a and F_b, a horizontal threshold D, and a vertical threshold H, the `conflict`(F_a, F_b, D, H) function can check if a possible conflict exists between the two flight plans (Fig. 6.4). It first uses the function `get-all-times`(F_a, F_b) to generate a non-decreasing vector of times corresponding to the waypoints in F_A and F_B as T. `confli ct` returns `True` if `T-violation`(T, F_a, F_b, D, H) returns `True`.

The function `T-violation`(T, F_a, F_b, D, H) recursively checks for possible conflicts in the period of interest between every pair of consecutive times in T and returns `True` if a conflict is detected (Fig. 6.5).

The `exists-violation`$(t_1, t_2, F_a, F_b, D, H)$ function returns `True` if there exists a time

$t : t_1 < t \leq t_2$ & $\|\mathbf{s}(t)\| < D$ & $|s_z(t)| < H$. It uses the `get-config`(t, F) function to get the configurations of aircraft a and b at time t_1 as $C_{a,t_1} = \langle s_{x,a,t_1}, s_{y,a,t_1}, s_{z,a,t_1}, v_{g,a,t_1}, \lambda_{a,t_1}, v_{z,a,t_1}\rangle$ and $C_{b,t_1} = \langle s_{x,b,t_1}, s_{y,b,t_1}, s_{z,b,t_1}, v_{g,b,t_1}, \lambda_{b,t_1}, v_{z,b,t_1}\rangle$. Then the x and y components of the velocity vectors of a and b at time t_1 are computed as:

$$v_{x,a,t_1} = v_{g,a,t_1} \times \cos\lambda_{a,t_1} \tag{6.15}$$

$$v_{y,a,t_1} = v_{g,a,t_1} \times \sin\lambda_{a,t_1} \tag{6.16}$$

$$v_{x,b,t_1} = v_{g,b,t_1} \times \cos\lambda_{b,t_1} \tag{6.17}$$

$$v_{y,b,t_1} = v_{g,b,t_1} \times \sin\lambda_{b,t_1} \tag{6.18}$$

Therefore, their initial states for the period of interest t_1 to t_2, are given by:

$$S_{a,t_1} = \langle s_{x,a,t_1}, s_{y,a,t_1}, s_{z,a,t_1}, v_{x,a,t_1}, v_{y,a,t_1}, v_{z,a,t_1}\rangle \tag{6.19}$$

$$S_{b,t_1} = \langle s_{x,b,t_1}, s_{y,b,t_1}, s_{z,b,t_1}, v_{x,b,t_1}, v_{y,b,t_1}, v_{z,b,t_1}\rangle \tag{6.20}$$

Now equations (6.13) and (6.14) can be used to determine if a violation of the well-clear volumes of the two aircraft is possible between t_1 and t_2. `exists-violation` returns `True` if a possible violation is detected (Fig. 6.6).

```
conflict($F_a, F_b, D, H$):
  Let
    $T$ = get-all-times($F_a, F_b$)
  in
    if T-violation($T, F_a, F_b, D, H$)
      return True
    else
      return False
    endif
```

Figure 6.4: Logic of the `conflict` function.

6.2.3 The Problem Statement

Given a set of flight plans Φ (which ranges over $\mathbb{F}^k, k \in \mathbb{N}$), the set is considered to be *safe* if there exists no conflict between any pair of elements in Φ. The `safe`(Φ, D, H) function returns `True` if for all $F_i, F_j \in \Phi$, `conflict`(F_i, F_j, D, H) returns `False` (Fig. 6.7). Thus, `safe`(Φ, D, H), implies that the set Φ is safe.

```
T-violation($T, F_a, F_b, D, H$):
  Let
    $T = \langle t_1, t_2, T' \rangle$
  in
    if T == Null
      return False
    else
      if exists-violation($t_1, t_2, F_a, F_b, D, H$)
        return True
      else
        return T-violation($\langle t_2, T' \rangle, F_a, F_b, D, H$)
      endif
    endif
```

Figure 6.5: Logic of the `T-violation` function.

```
exists-violation($t_1, t_2, F_a, F_b, D, H$):
  Let
    $\langle s_{x,a,t_1}, s_{y,a,t_1}, s_{z,a,t_1}, v_{g,a,t_1}, \lambda_{a,t_1}, v_{z,a,t_1} \rangle$
                                                = get-config($t_1, F_a$)
    $\langle s_{x,b,t_1}, s_{y,b,t_1}, s_{z,b,t_1}, v_{g,b,t_1}, \lambda_{b,t_1}, v_{z,b,t_1} \rangle$
                                                = get-config($t_1, F_b$)
    $v_{x,a,t_1} = v_{g,a,t_1} \times \cos \lambda_{a,t_1}$
    $v_{y,a,t_1} = v_{g,a,t_1} \times \sin \lambda_{a,t_1}$
    $v_{x,b,t_1} = v_{g,b,t_1} \times \cos \lambda_{b,t_1}$
    $v_{y,b,t_1} = v_{g,b,t_1} \times \sin \lambda_{b,t_1}$
    $S_{a,t_1} = \langle s_{x,a,t_1}, s_{y,a,t_1}, s_{z,a,t_1}, v_{x,a,t_1}, v_{y,a,t_1}, v_{z,a,t_1} \rangle$
    $S_{b,t_1} = \langle s_{x,b,t_1}, s_{y,b,t_1}, s_{z,b,t_1}, v_{x,b,t_1}, v_{y,b,t_1}, v_{z,b,t_1} \rangle$
    $\mathbf{s}_{a,t_1} = \langle s_{x,a,t_1}, s_{y,a,t_1} \rangle$
    $\mathbf{s}_{b,t_1} = \langle s_{x,b,t_1}, s_{y,b,t_1} \rangle$
    $\mathbf{v}_{a,t_1} = \langle v_{x,a,t_1}, v_{y,a,t_1} \rangle$
    $\mathbf{v}_{b,t_1} = \langle v_{x,b,t_1}, v_{y,b,t_1} \rangle$
    $\mathbf{s}_{t_1} = \mathbf{s}_{a,t_1} - \mathbf{s}_{b,t_1}$
    $\mathbf{v}_{t_1} = \mathbf{v}_{a,t_1} - \mathbf{v}_{b,t_1}$
    $\mathbf{s}_t = \mathbf{s}_{t_1} + (t - t_1)\mathbf{v}_{t_1}$
    $s_{z,t_1} = s_{z,a,t_1} - s_{z,b,t_1}$
    $v_{z,t_1} = v_{z,a,t_1} - v_{z,b,t_1}$
    $s_{z,t} = s_{z,t_1} + (t - t_1) v_{z,t_1}$
  in
    if ∃ t . $\|\mathbf{s}_t\| < D$ & $|s_{z,t}| < H$ & $t_1 < t \leq t_2$
      return True
    else
      return False
    endif
```

Figure 6.6: Logic of the `T-violation` function.

```
safe($\Phi, D, H$):
  if $\forall F_i, F_j \in \Phi$: $\neg$conflict($F_i, F_j, D, H$)
    return True
  else
    return False
  endif
```

Figure 6.7: Logic of the `safe` function.

The problem statement can now be expressed as: *Given a set of immutable traffic flight plans Φ, a horizontal threshold D, a vertical threshold H, and a proposal flight plan F_a such that* `safe`*(Φ, D, H) but $\neg$*`safe`*($\Phi \cup \{F_a\}, D, H$), the goal is to find a valid solution flight plan $\bar{F_a}$ such that* `safe`*($\Phi \cup \{\bar{F_a}\}, D, H$).*

6.2.4 The Strategy for Computing a Conflict-Free Flight Plan $\bar{F_a}$

We have shown how conflicts between two given flight plans can be detected using the `conflict` function and how the `safe` function can determine if a set of flight plans is safe. As expressed in the problem statement, the goal is to find a solution flight plan $\bar{F_a}$ given a safe set of immutable flight plans Φ and a proposal F_a. Since the flight plans in Φ are immutable, only the proposal flight plan F_a may be adjusted to find a solution. Let us consider an aircraft a that proposes F_a to be the ownship and another traffic aircraft b such that $F_b \in \Phi$. For the well-clear volumes of a and b to intersect, there needs to exist a time t such that equations (6.13) and (6.14) are simultaneously satisfied. Therefore, our strategy for avoiding conflicts between a and b is to assign suitable values of ground speed (v_g) to the different segments of F_a, such that the intersection of the well-clear volumes of a and b can be avoided. Our algorithm takes as input a vector ξ_{air} (which ranges over $\mathbb{R}^n, n \in \mathbb{N}$) of discrete values of airspeed that the ownship can fly with. Given the horizontal wind vector $\mathbf{w}$, a corresponding vector $\xi = \xi_{air}(\mathbf{w})$ (which ranges over $\mathbb{R}^n, n \in \mathbb{N}$) of values of ground speed for the ownship can be computed. The algorithm assigns a value of ground speed $v_g \in \xi$ to each segment in the ownship's flight plan to create a solution $\bar{F_a}$ such that $\neg$`conflict` $(\bar{F_a}, F_b, D, H)$. The vertical speed for each segment is then adjusted accordingly to ensure that the 3D profile of $\bar{F_a}$ is similar to that of the original flight plan F_a. We assume that aircraft can change their ground speed and vertical speed instantaneously. This pairwise strategy for conflict avoidance is extended to the set Φ so that a valid $\bar{F_a}$ will avoid conflict with every $F_b \in \Phi$, i.e., $\forall F_b \in \Phi : \neg$`conflict`$(\bar{F_a}, F_b, D, H)$. Our solution space is limited to different permutations of $v_g \in \xi$ in the segments comprising F_a. Therefore, it should be noted that this conservative approach may not always be able to compute a solution flight plan.

Since the heading of the velocity is not changed in a segment, only two components of the velocity vector are altered – the ground speed and the vertical speed. This restricts the introduction of new waypoints in the solution flight plan. Thus, for a flight plan $F_a = \langle t, \langle C_1, C_2, ..., C_n \rangle\rangle$, where $C_i = \langle s_x, s_y, s_z, v_g, \lambda, v_z \rangle$, a valid solution $\bar{F_a} = \langle t, \langle C'_1, C'_2, ..., C'_n \rangle\rangle$ satisfies the condition that $C'_i = \langle s_x, s_y, s_z, {v_g}', \lambda, v'_z \rangle$.

There are two steps involved in the computation of a solution flight plan:

1. Finding a value of ground speed $v_g \in \xi$ for every segment $L \in F_a$.

2. Adjusting the vertical speed to maintain the 3D profile of the flight plan

In the next section, we will present the algorithm for computing a valid assignment of ground speed for the segments, that has been formally verified to result in a conflict-free flight plan if a solution is found.

When the ground speed in a flight segment between two consecutive waypoints is changed from v_g to ${v_g}'$, in order to maintain the 3D profile of the flight segment, the vertical speed v_z for that flight segment needs to be adjusted 6.8. The angle that the straight-line

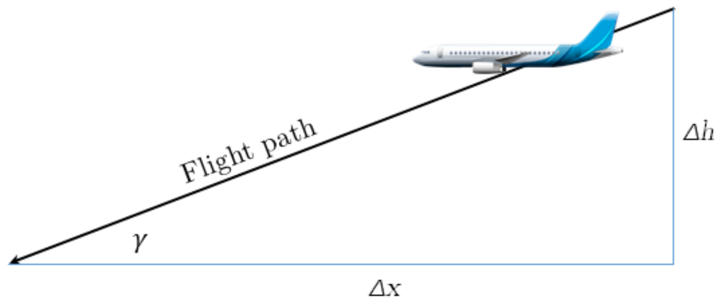


Figure 6.8: Vertical flight path angle.

flight path makes with the horizontal is given by the following equation:

$$\gamma = \tan^{-1}\frac{\Delta h}{\Delta x} = \frac{v_z}{v_g} \qquad (6.21)$$

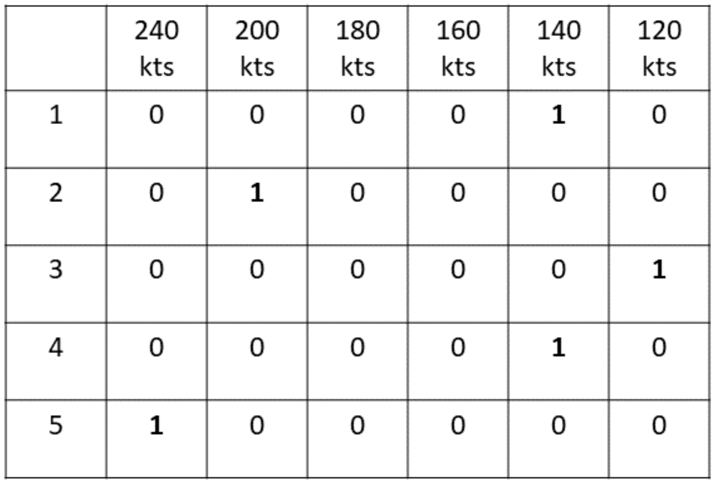
For the vertical profile to remain same, the angle γ needs to remain unchanged. Therefore, the required vertical speed $v_z^{'}$ for a segment can be computed by using equation 6.22.

$$v_z^{'} = v_g^{'} \times \frac{v_z}{v_g} \tag{6.22}$$

6.2.5 Algorithm for Computing the Ground Speed Assignments in $\bar{F_a}$

In this section, we present an algorithm for assigning a discrete value of ground speed to each segment of a flight plan F_a to create a valid solution flight plan $\bar{F_a}$. We also describe certain desirable correctness properties of the algorithm.

The assignment of ground speed to the segments can be arranged as an assignment matrix M of dimension $n \times m$ where $n \in \mathbb{N}$ is the number of segments in F_a and $m \in \mathbb{N}$ is the number of discrete values of ground speed in ξ (M ranges over $\{0,1\}^{n \times m}$). Each row r represents a segment of $\bar{F_a}$ and each column c represents a value of ground speed in ξ. Initially, all the cells of M are marked with 0's. A matrix comprised of only 0's is represented by M_0. When a value of ground speed corresponding to column c is assigned to a segment corresponding to row r, the cell $M[r,c]$ is marked with a 1. An example assignment matrix is given in Fig. 6.9. It represents a flight plan with 5 segments that have been assigned ground speeds of 140, 200, 120, 140, and 240 *kts* respectively.



	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

Figure 6.9: Example assignment matrix for a solution flight plan.

An algorithm that solves the above matrix assignment problem should satisfy the following properties:

- *Safety* - The algorithm should return an assignment matrix M if and only if adding the corresponding flight plan F to the given set of traffic flight plans Φ creates a safe set of flight plans $\Phi \cup \{F\}$.

  This property ensures that if a solution flight plan is found, the resulting set of flight plans in the airspace is safe.

- *Completeness* - If there is a value of ground speed that can be assigned to a segment such that it leads to a valid solution, then the algorithm will not return M_0.

  This property ensures that if a valid permutation of values of ground speed exists, the algorithm will find the corresponding assignment matrix and not incorrectly terminate by returning M_0.

We propose a backtracking algorithm for finding the assignment matrix M corresponding to a valid $\bar{F_a}$. It returns a solution only if it can find an assignment matrix such that the corresponding flight plan can be safely added to the set of traffic flight plans. It returns M_0 if no such solution exists.

```
complete($M$):
 if $\forall$ $row$ $\in$ $M$: assigned($row$)
  return True
 else
  return False
```

Figure 6.10: Logic of the `complete` function.

```
check-safety($M, F_a, \Phi, \xi, D, H$):
  Let
    $F$ = plan($M, F_a, \xi$)
  in
    if safe($\Phi \cup \{F\}, D, H$)
      return True
    else
      return False
    endif
```

Figure 6.11: Logic of the `check-safety` function.

Our algorithm makes use of four primary functions:

- The `complete` function takes as input an assignment matrix M and returns `True` only if all the rows of M have a ground speed assignment (Fig. 6.10). An assignment matrix M is *complete* if `complete`(M) returns `True` and *incomplete* otherwise.

```
valid($v, M, F_a, \Phi, \xi, D, H$):
 Let
  $M'$ = set($M$,unassigned($M$),getindex($\xi, v$))
  safety = check-safety($M', F_a, \Phi, \xi, D, H$)
  succeeding = solve($M', F_a, \Phi, \xi, D, H$)
 in
  if (safety & succeeding $\neq M_0$)
    return True
  else
    return False
  endif
```

Figure 6.12: Logic of the `valid` function.

```
solve($M, F_a, \Phi, \xi, D, H$):
 if complete($M$) & check-safety($M, F_a, \Phi, \xi, D, H$)
  return $M$
 endif
 if complete($M$) & $\neg$check-safety($M, F_a, \Phi, \xi, D, H$)
  return $M_0$
 endif
 if $\neg$complete($M$)&$\exists v : v \in \xi$&valid($v, M, F_a, \Phi, \xi, D, H$)
  Let
   $M'$ = set($M$,unassigned($M$),getindex($\xi, v$))
  in
   return solve($M', F_a, \Phi, \xi, D, H$)
 endif
 if $\neg$complete($M$)&$\neg(\exists v : v \in \xi$&valid($v, M, F_a, \Phi, \xi, D, H$))
  return $M_0$
 endif
```

Figure 6.13: Logic of the `solve` function.

- The `check-safety` function can determine if the flight plan corresponding to an assignment matrix M can be safely added to Φ (Fig. 6.11). It takes as input an assignment matrix M, the proposal flight plan F_a, a set of traffic flight plans Φ, a vector ξ of discrete values of ground speed for aircraft a, and the thresholds D and H. It uses the `plan` function to create a flight plan F corresponding to M and returns `True` only if the set $\Phi \cup \{F\}$ is safe.

- An assignment for a row is valid if and only if both the following conditions are satisfied:

  - The corresponding intermediate flight plan can be safely added to the set Φ.
  - The assignment allows assignments of ground speed for succeeding rows.

  The `valid` function can be used to check the validity of an assignment (Fig. 6.12). It takes as input a value of ground speed v, an assignment matrix M, the proposal flight plan F_a, a set of traffic flight plans Φ, a vector ξ of discrete values of ground speed for aircraft a, and the thresholds D and H. It returns `True` if the assignment of v to the first unassigned row satisfies the two conditions for a valid assignment and `False` otherwise.

- The `solve` function takes as input an assignment matrix M, the proposal flight plan F_a, a set of traffic flight plans Φ, a vector ξ of discrete values of ground speed for aircraft a, and the thresholds D and H (Fig. 6.13). It tries to assign a value of ground speed to the first unassigned row of M (given by `unassigned`(M)). `solve` returns the assignment matrix M if it is complete and `check-safety` returns `True` for M. If M is incomplete, then solve checks if a valid assignment of ground speed is possible for the first unassigned row. If a valid assignment of ground speed v can be found for the first unassigned row, then the current assignment matrix M is updated to a new matrix M' using the `set` function. The `set` function assigns 1 to the cell [`unassigned`(M), `getindex`(ξ, v)], where the function `getindex`(ξ, v) returns the index of v in ξ. `sol ve` then proceeds to make an assignment to the first unassigned row in the updated assignment matrix M'. If no valid assignment of ground speed can be found for the first unassigned row of M, then `solve` exits by returning the empty matrix M_0.

6.3 Formal Verification of the Flight Planning Algorithm

In this section, we present some important correctness properties of the specifications introduced in Section 6.2. We also present the formal specifications of safety and completeness properties.

Lemma 6.1 `exists-violation` *returns* `True` *if and only if a violation of minimum horizontal separation D and minimum vertical separation H between time t_1 and t_2 exists.*

`Lemma` $6.1 \equiv$
$\forall D, H, t_1, t_2 \in \mathbb{R}, F_a, F_b \in \mathbb{F}:$
$$\texttt{exists-violation}(t_1, t_2, F_a, F_b, D, H) \iff \exists t \in \mathbb{R} : \|\mathbf{s}_{xy,t}\| < D \;\;\&\;\; |s_{z,t}| < H \;\;\&\;\; t_1 < t \leq t_2$$

Lemma 6.2 `T-violation` *recursively checks for possible conflicts between pair of consecutive times in a vector of times $\langle t_1, t_2, T\rangle$ and returns* `True` *if a conflict is detected between two consecutive times t_1 and t_2.*

$\texttt{Lemma 6.2} \equiv$
$\forall D, H, t_1, t_2 \in \mathbb{R}, n \in \mathbb{N}, T \in \mathbb{R}^n, F_a, F_b \in \mathbb{F}:$
$\quad (\texttt{exists-violation}(t_1, t_2, F_a, F_b, D, H) \implies \texttt{T-violation}(\langle t_1, t_2, T\rangle, F_a, F_b, D, H))$
$\quad \& \ (\neg\texttt{exists-violation}(t_1, t_2, F_a, F_b, D, H) \implies$
$\qquad \texttt{T-violation}(\langle t_1, t_2, T\rangle, F_a, F_b, D, H) = \texttt{T-violation}(\langle t_2, T\rangle, F_a, F_b, D, H))$

Lemma 6.3 `conflict` *function returns* ***`True`*** *if and only if a conflict is detected at any time between two given flight plans.*

$\texttt{Lemma 6.3} \equiv$
$\forall D, H \in \mathbb{R}, F_a, F_b \in \mathbb{F}:$
$\quad \texttt{T-violation}(\texttt{get-all-times}(F_a, F_b), F_a, F_b, D, H) \iff \texttt{conflict}(F_a, F_b, D, H)$

Lemma 6.4 `safe` *returns* `True` *if and only if there is no conflict between any two flight plans in the given set of flight plans.*

$\texttt{Lemma 6.4} \equiv$
$\forall D, H \in \mathbb{R}, k \in \mathbb{N}, \Phi \in \mathbb{F}^k:$
$\quad \neg(\exists F_i, F_j \in \Phi : \texttt{conflict}(F_i, F_j, D, H)) \iff \texttt{safe}(\Phi, D, H)$

$\texttt{safe}(\Phi, D, H)$=`True` guarantees that the set of flight plans Φ is safe with respect to the horizontal and vertical thresholds D and H.

Lemma 6.5 `check-safety` *returns* `True` *for an assignment matrix M if and only if adding the corresponding flight plan to the set of traffic flight plans Φ results in a safe set of flight plans*

$\texttt{Lemma 6.5} \equiv$
$\forall D, H \in \mathbb{R}, n, m, k \in \mathbb{N}, M \in \{0,1\}^{n\times m}, F \in \mathbb{F}, \Phi : \mathbb{F}^k, \xi \in \mathbb{R}^m:$
$\qquad \texttt{safe}(\Phi \cup \texttt{plan}(M, F, \xi), D, H) \iff \texttt{check-safety}(M, F, \Phi, \xi, D, H)$

Lemma 6.6 `complete` *returns* `True` *if and only if every row in the matrix M has been assigned a value of ground speed.*

$\texttt{Lemma 6.6} \equiv$
$\forall n, m \in \mathbb{N}, M \in \{0,1\}^{n\times m}:$
$\quad (\forall r \in M : \texttt{assigned}(r)) \iff \texttt{complete}(M)$

Lemma 6.7 `valid` *returns* `True` *if and only if the assignment of v to the first unassigned row satisfies the two required conditions for a valid assignment.*

`Lemma` $6.7 \equiv$
$\forall D, H, v \in \mathbb{R}, n, m, k \in \mathbb{N}, M \in \{0,1\}^{n \times m}, F_a \in \mathbb{F}, \Phi \in \mathbb{F}^k, \xi \in \mathbb{R}^m :$
$\quad \texttt{valid}(v, M, F, \Phi, \xi, D, H) \iff$
$\qquad (\texttt{check-safety}(set(M, \texttt{unassigned}(M), \texttt{getindex}(\xi, v)), F, \Phi, \xi, D, H)$
$\qquad \& \neg(\texttt{solve}(M, F, \Phi, \xi, D, H) = M_0))$

6.3.1 The Safety Property

Theorem 6.1 *(`solve` is safe) 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 produces a safe set of flight plans.*

`Theorem` $6.1 \equiv$
$\forall D, H \in \mathbb{R}, n, m, k \in \mathbb{N}, M \in \{0,1\}^{n \times m}, F_a \in \mathbb{F}, \Phi \in \mathbb{F}^k, \xi \in \mathbb{R}^m :$
$\quad \texttt{complete}(M) \implies$
$\qquad ((\texttt{solve}(M, F, \Phi, \xi, D, H) = M) \iff \texttt{safe}(\Phi \cup \texttt{plan}(M, F, \xi), D, H))$

6.3.2 The Completeness Property

Theorem 6.2 *(`solve` is complete) If `solve` returns the empty assignment matrix M_0, 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 would produce a safe set of flight plans.*

`Theorem` $6.2 \equiv$
$\forall\ D, H \in \mathbb{R}, n, m, k \in \mathbb{N}, M \in \{0,1\}^{n \times m}, F_a \in \mathbb{F}, \Phi \in \mathbb{F}^k, \xi \in \mathbb{R}^m : \neg\texttt{complete}(M) \implies$
$\qquad ((\exists\ v : v \in \xi\ \&\ \texttt{valid}(v, M, F, \Phi, \xi, D, H)) \iff \neg(\texttt{solve}(M, F, \Phi, \xi, D, H) = M_0))$

All the lemmas and theorems presented in this section have been verified using the Athena proof assistant[19].

6.4 Experimental Evaluation of the Protocol

We developed a reference implementation of our approach in Python that closely follows the specifications that were used to verify the properties in Athena. We created a proposal

[19]Complete Athena code available at: https://wcl.cs.rpi.edu/assure

flight plan F_a for an ownship a and a set of traffic flight plans Φ. Initially, Φ contained only one traffic flight plan F_b that had a conflict with F_a. We then incrementally added two more traffic flight plans F_c and F_d that were designed such that a solution flight plan $\bar{F_a}$ would not exist for the proposal. All our experiments assume a Euclidean 3D coordinate system (with 100 feet = 1 unit on both axes). We used a value of 1 nautical mile for the horizontal threshold (D) and 1000 feet for the vertical threshold (H). All the flight plans were at a constant altitude of 5000 feet. The details of the flight plans are given below:

- F_a had a starting time of 0.00 seconds and was comprised of the waypoints $\langle -60, 100, 50\rangle$, $\langle -55, 65, 50\rangle$, $\langle 5, 5, 50\rangle$, and $\langle 125, 5, 50\rangle$ with a proposed ground speed of 240 *kts* in every segment.

- F_b had a starting time of 0.00 seconds and was comprised of the waypoints $\langle 125, 5, 50\rangle$, $\langle 5, 5, 50\rangle$, and $\langle -55, -55, 50\rangle$ with a ground speed of 240 *kts* in every segment.

- F_c had a starting time of 61.23 seconds and was comprised of the waypoints $\langle 69, -85, 50\rangle$, $\langle 71, -55, 50\rangle$, and $\langle 121, 5, 50\rangle$ with a ground speed of 200 *kts* in every segment.

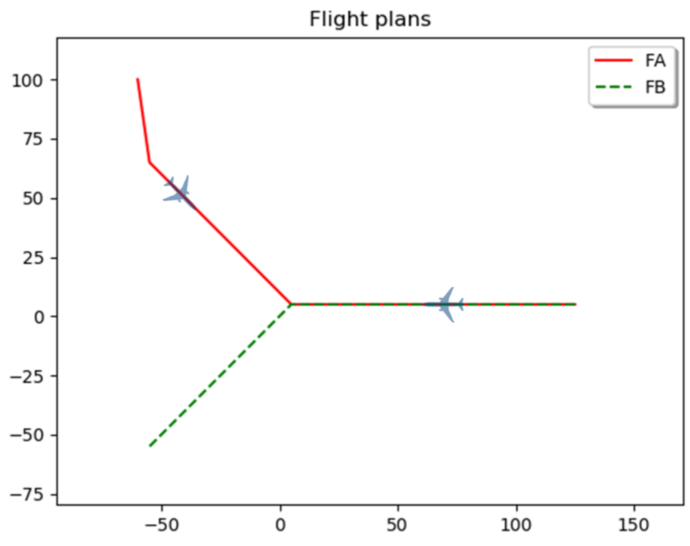
- F_d had a starting time of 63.66 seconds and was comprised of the waypoints $\langle 3, -55, 50\rangle$, $\langle 123, 65, 50\rangle$, and $\langle 160, 70, 50\rangle$ with a ground speed of 174 *kts* in every segment.

- The vector of ground speeds for aircraft a was $\xi = \langle 100, 120, 140, 160, 180, 200, 220, 240\rangle$ (all in *kts*).

Fig. 6.14 depicts the first scenario where Φ contains one traffic flight plan F_b. a has a possible conflict with aircraft b in the third segment of F_a between the waypoints $\langle 5, 5, 50\rangle$ and $\langle 125, 5, 50\rangle$. Our algorithm computed the assignment matrix given in Fig. 6.15 that corresponded to a solution flight plan $\bar{F_a}$.

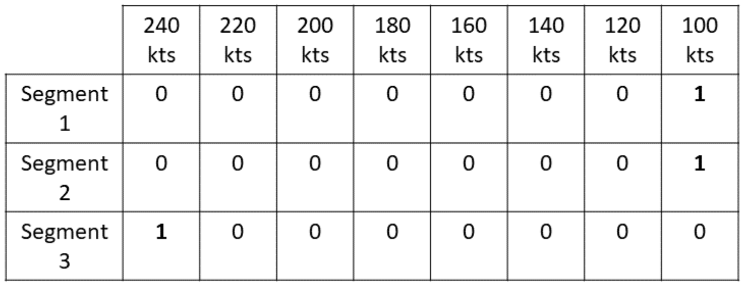
In the second scenario depicted in Fig. 6.16, we added a new traffic flight plan F_c to Φ ($\Phi' = \Phi \cup \{F_c\}$) such that aircraft a has a conflict with aircraft c if it follows the solution flight plan $\bar{F_a}$ created from the assignment matrix in Fig. 6.15. In this case, our algorithm computed the assignment matrix given in Fig. 6.17 that can be used to create a solution flight plan $\bar{F_a}'$ that maintains standard separation from both aircraft b and c.

For the third scenario, we added a fourth traffic aircraft d to the airspace (Fig. 6.18). Adding the new flight plan F_d to Φ' ($\Phi'' = \Phi \cup \{F_d\}$) created a situation where no permutation



H]

Figure 6.14: Top view of a scenario where an aircraft a is in conflict with a traffic aircraft b (aircraft markers represent the direction of flight plans and not actual positions).



	240 kts	220 kts	200 kts	180 kts	160 kts	140 kts	120 kts	100 kts
Segment 1	0	0	0	0	0	0	0	**1**
Segment 2	0	0	0	0	0	0	0	**1**
Segment 3	**1**	0	0	0	0	0	0	0

Figure 6.15: An assignment matrix for the scenario in Fig. 6.14.

of ground speed in the three segments of F_a could generate a solution that is conflict-free from all three traffic aircraft. However, on introducing an additional value of 60 *kts* ground speed in ξ, our algorithm could compute the assignment matrix for a solution flight plan $\bar{F_a}''$ as given in Fig. 6.19.

It is evident from the experiments that our approach can be effectively used for generating conflict-free flight plans. The third scenario where a flight plan was not initially possible but could be generated after a new discrete value of ground speed was allowed clearly shows that given a start time, ξ is an important variable in determining the possibility of a solution. Since ξ is a function of the allowed values of airspeed for an aircraft and the wind vector $\mathbf{w}$, it is obvious that the wind conditions can significantly impact solutions.

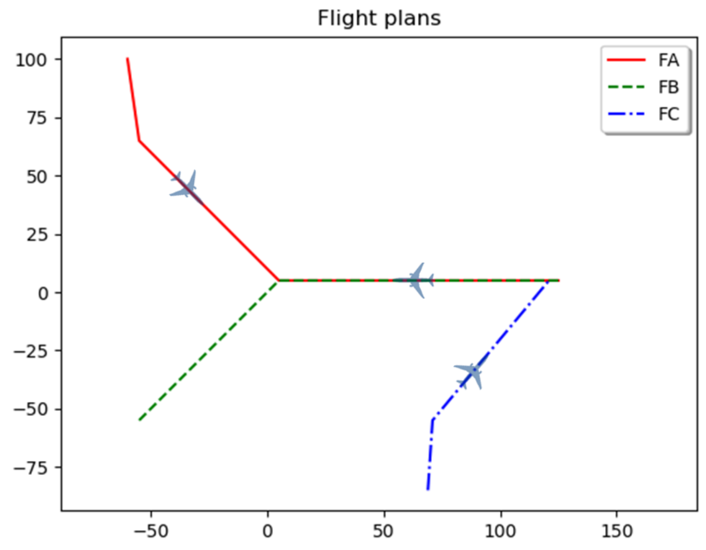
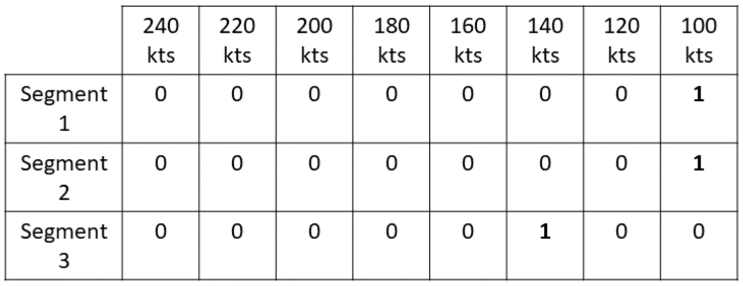


Figure 6.16: Top view of a scenario where an aircraft a is in conflict with two traffic aircraft b and c (aircraft markers represent the direction of flight plans and not actual positions).



	240 kts	220 kts	200 kts	180 kts	160 kts	140 kts	120 kts	100 kts
Segment 1	0	0	0	0	0	0	0	**1**
Segment 2	0	0	0	0	0	0	0	**1**
Segment 3	0	0	0	0	0	**1**	0	0

Figure 6.17: An assignment matrix for the scenario in Fig. 6.16.

6.5 Related Work

Pairwise conflict detection and avoidance have been previously investigated. Pritchett *et al.* [145] have presented a decentralized algorithm for aircraft conflict resolution that is based on negotiated bargaining. The resolutions proposed by them are multi-dimensional, i.e., an aircraft can either move up, down, right/left, or increase or decrease its velocity. They associate a cost to each type of resolution and find the solution that entails the least cost. They do this by using the game theory concept of bargaining. Since both aircraft maneuver in this approach, the total cost of maneuvers is divided between the ownship and the traffic aircraft, reducing the cost involved for each individual aircraft. Dowek *et al.* [46] have proposed a formally verified pairwise coordinated technique for conflict avoidance in which only one component of the velocity vector is modified. Their solutions are coordinated

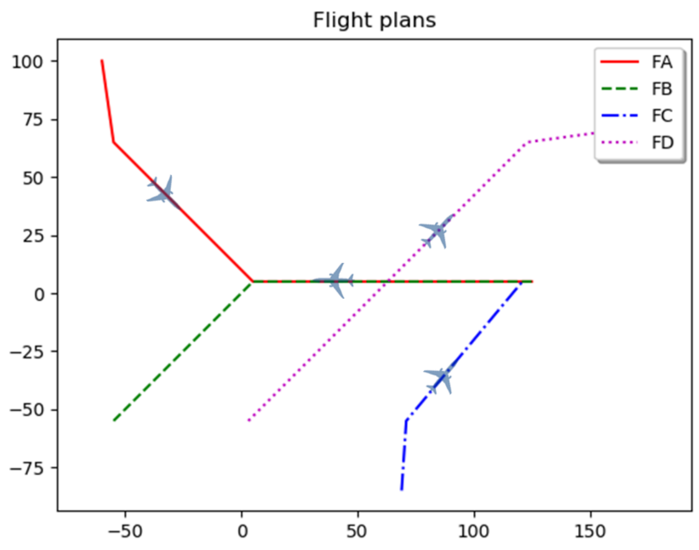
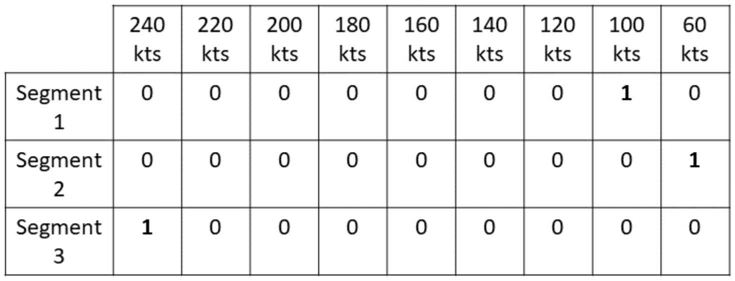


Figure 6.18: Top view of a scenario where an aircraft a is in conflict with three traffic aircraft b, c, and d (aircraft markers represent the direction of flight plans and not actual positions).



	240 kts	220 kts	200 kts	180 kts	160 kts	140 kts	120 kts	100 kts	60 kts
Segment 1	0	0	0	0	0	0	0	**1**	0
Segment 2	0	0	0	0	0	0	0	0	**1**
Segment 3	**1**	0	0	0	0	0	0	0	0

Figure 6.19: An assignment matrix for the scenario in Fig. 6.18.

such that the ownship and the traffic aircraft independently choose different directions for their conflict avoidance maneuvers. This makes sure that if either one or both the aircraft maneuver, then the potential conflict is avoided. Galdino *et al.* [62] have proposed an optimization over of Dowek *et al.*'s work by considering maneuvers in the horizontal plane that include combined modifications of the ground speed and heading of the ownship. In both these approaches, the ownship's course is modified so that it deviates from its original path. Balachandran *et al.* [12] have presented an approach for scheduling multiple *uncrewed aerial vehicles* (UAV) moving towards an intersection in an urban environment. In this approach, UAVs can adjust their time of arrival by modifying their speed and using lateral deviations. The RAFT consensus algorithm [131] is used to synchronize information across all vehicles and a *scheduling algorithm* is used to compute a compatible schedule for the UAVs. Alejo *et* *al.* [5] have proposed a technique for finding collision-free trajectories for autonomous quadrotors or helicopters whose minimum speed can be zero. Their approach involves adding new waypoints to a proposed trajectory and changing the speeds of the UAVs. They initially find a non-optimal solution and then use *particle swarm optimization* [143] to find better solutions. In contrast to the above work, the solutions generated by our approach do not include lateral or vertical deviations in order to resolve conflicts. Therefore, making it suitable in situations where additional waypoints cannot be added due to restrictions imposed by terrain, weather, or airspace capacity.

Work has been done by NASA on formally verified sense and avoid systems for integration of UAS in the NAS [125, 128]. Muñoz *et al.* [124] have presented an enhancement of the TCAS conflict detection and resolution system. They provide a formally verified algorithm for predicting the issuance of TCAS resolution advisories so that pilots and UAS flight computers can take preventive actions to avoid the issuance of TCAS resolution advisories. Given a look-ahead interval, they detect resolution advisories instead of conflicts. NASA's DAIDALUS [126] suite of algorithms is aimed at increasing situational awareness of UAS pilots. DAIDALUS contains algorithms for detecting possible violations of well-clear volumes and alerting the pilots. It also includes algorithms for providing maneuvering guidance to successfully avoid conflicts. These algorithms are integral components of NASA's ICAROUS [13] and DANTi [26] systems. Colbert *et al.* [37] have presented a formally-verified algorithm called *PolySafe* that can detect conflicts between polynomial trajectories but cannot generate resolutions to avoid these conflicts.

A very similar formally-verified flight planning approach for UAS has been presented by Balachandran *et al.* [14] which accommodates real-time traffic and *geofence* constraints. Their approach is different from our approach as it assumes that all traffic aircraft travel with a constant velocity, while our approach only assumes that traffic aircraft travel with constant velocities between consecutive waypoints. Moreover, their algorithm assumes that the original flight operational constraints are defined only by a starting and an ending point which allows them to generate conflict-free trajectories by identifying possible waypoints in the 3D space as required. This is different from our approach which assumes that the original flight operational constraints are defined by a series of waypoints, spatial deviations from which may not be allowed or possible. In theory, Balachandran *et al.*'s algorithm can be used by an ownship to generate the initial input flight plan for our conflict-aware algorithm.

6.6 Chapter Summary

In this chapter, we have presented a strategic conflict-aware flight-planning algorithm that can be used by an ownship to detect and resolve conflicts with multi-segment flight plans of multiple traffic aircraft. Our algorithm has been formally verified to satisfy certain desired correctness properties, making it suitable for safety-critical aerospace applications like DAC. The conflict resolutions generated by our algorithm do not deviate an aircraft from its intended 3D spatial trajectory, making our approach suitable for situations where spatial deviations may be restricted by traffic, weather, or other factors. This protocol can be used by aircraft participating in DAC to compute conflict-free proposals.

Currently, our approach can only generate a flight plan for an ownship given a set of immutable traffic flight plans. If a solution cannot be found, no changes are made to the traffic flight plans in order to accommodate the ownship. Moreover, it assumes that the aircraft will maintain constant-velocity flight paths between a fixed set of waypoints. Therefore, interesting future directions of work would involve expanding the approach to develop a formally verified and efficient protocol that can mutate the flight plans of the traffic aircraft in case no trajectories are possible with the given set of flight plans, and to support other types of aircraft flight paths such as polynomial paths.

CHAPTER 7
DISCUSSION AND FUTURE WORK

In this thesis, we have presented a formal analysis of distributed coordination algorithms for safety-critical autonomous multi-agent systems, with a focus on aerospace applications. We have proposed novel techniques that can aid in the verification of distributed applications under uncertain and dynamic operating conditions. In this chapter, we present some relevant discussion on our methods along with some potential future directions of work.

7.1 Analysis of Decentralized Multi-Agent Coordination

Assumption of non-Byzantine behavior

In our analysis of distributed coordination algorithms, we have assumed that the agents will be non-Byzantine and *non-adversarial* in nature. In the presence of Byzantine or adversarial actions, it is challenging to provide accurate correctness guarantees as the properties that are deemed to be valid under non-Byzantine conditions may no longer be valid. In particular, consensus-based distributed algorithms are particularly susceptible to adversarial behavior, making it challenging to issue security and safety concerns in such applications. *E.g.,* if there are a majority of adversarial acceptors in Synod, these acceptors can forever refuse to accept any proposal, thereby impeding progress. Therefore, for safety-critical applications, it is important to identify the assumptions under which correctness can be guaranteed for distributed protocols in the presence of Byzantine and/or adversarial agents.

The failure-aware actor model

Predicate fairness in our failure-aware actor model is a strong fairness property that can be used for reasoning about different types of progress properties in different types of distributed protocols, depending on how the predicates are defined. In Chapter 2, we have formally shown that predicate fairness can be used to theoretically guarantee our conditions for eventual progress in Synod if the proposers keep retrying with higher proposal numbers. However, it is important to note that predicate fairness is not necessary for our conditions for progress in Synod to be satisfied as the conditions may eventually be true even in a system that is not predicate fair. In fact, it may be possible to replace predicate fairness with a weaker fairness property that can still be sufficient to claim the conditions required for progress in Synod. Additionally, the current fairness assumptions of FAM do not allow modeling message loss, even though for guaranteeing eventual progress in Synod it suffices to guarantee the eventual delivery of only a subset of the messages.

Representation of "*knowledge*" and "*belief*" in multi-agent systems

In this work, we have assumed that autonomous agents will share information with each other in order to coordinate in the absence of a decentralized coordinator. For our analysis, we have defined knowledge as the state of an agent being aware of a fact, where a fact can be any truth about the real world. However, in some situations, it may be the case that agents only have some *"beliefs"* about the world that may not be true. *E.g.,* based on erroneous calculations, an aircraft may believe that a set of flight plans is conflict-free, while in reality, that may not be the case. This can lead to situations where agents can receive inconsistent information, which may be a combination of facts and beliefs, from other agents. In such conditions, the agents may be required to *adjudicate* such inconsistent information to derive meaningful operational or ethical conclusions [22]. Our analysis does not consider such possibilities in which the information propagated for coordination may include beliefs of the individual agents instead of only consistent facts about reality.

7.2 Verification of Stochastic and Dynamic Distributed Systems

Probabilistic properties of progress for distributed algorithms

We have introduced probabilistic properties of timely progress for distributed algorithms that can be useful when deterministic guarantees of timely progress cannot be provided. To do that, we have used some probabilistic assumptions about system conditions, such as message transmission and processing delays. A question may arise that, if there is a known probabilistic bound on the message delays, then why use algorithms like Synod, that are designed for use in purely asynchronous settings? A known bound on message delays makes it possible to probabilistically detect agent failures, making it possible to use other methods for coordination, such as broadcast, that can rely on partial synchrony. We believe that both probabilistic properties of timely progress and deterministic properties of eventual progress have their own sets of benefits for safety-critical applications. Probabilistic properties of timely progress can provide useful time bounds that can be used for decision-making in practical scenarios, but they cannot guarantee progress with certainty. On the other hand, deterministic properties of eventual progress can provide theoretically elegant guarantees of progress with certainty, but do not provide any useful time bounds. Since the two types of properties are not related by logical entailment, it is our belief that for the theoretical analysis of safety-critical systems, it is important to investigate eventual progress under purely asynchronous conditions. For this reason, algorithms that do not assume any form of synchrony can be studied to identify the theoretical limitations on deterministic eventual progress. Probabilistic properties of timely progress can be used in practical implementations of systems when the systems have to make operational decisions in real-time.

The formal DDDAS feedback loop for runtime verification

Our DDDAS-based approach for the runtime verification of dynamic systems using machine-checked theorems presents a novel technique to extend the practicality of formal proofs beyond the pre-deployment stages. It allows systems to generate new properties dynamically at runtime without sacrificing the mathematical rigor of formal theorem proving principles. However, in its current form, the technique is limited by the number of parameterized proofs that can be pre-developed to effectively model possible subsets of the dynamic operating conditions. Generating accurate and correct sentinels from the formal specification of the envelopes is also a challenge. Therefore, this approach must be further investigated and the various components must be refined in order to make it robust and scalable to real-world dynamic systems.

7.3 Autonomous Multi-Agent Aerospace Operations

Decentralized Admission Control

Our decentralized admission control approach can be used for autonomous air traffic management for well-defined four-dimensional airspaces in the absence of a centralized controller. In this thesis, we have only studied a subset of the challenges involved in decentralized multi-agent coordination in DAC, but there are several open challenges to the practical and efficient implementation of DAC for UAM operations. Since aircraft may have temporal overlaps in sections of their flight plans, with respect to different sets of flight plans, the same three-dimensional airspace may have different four-dimensional instances in different temporal periods. Therefore, the discretization of four-dimensional airspaces for DAC in a meaningful way is an important challenge. Moreover, the initialization of DAC for every four-dimensional airspace is an important problem because, in the absence of any owner, some uniform approach must be identified to admit the first owner in the presence of multiple candidates. Another challenge is deciding how to implement DAC when aircraft have to pass through multiple airspaces — situations may arise when aircraft may be admitted to some airspace, but not to the succeeding airspace. In such cases, a fixed-wing aircraft may need to execute a *holding pattern* until it can get authorization to proceed to the next airspace. For this reason, autonomous ATM techniques like DAC must consider airspace operational capacities at both local and global scales. Further research is needed to address such intricate challenges involved in the practical realization of DAC for UAM operations.

Dynamic Data-Driven Aerospace Systems

DDDAS techniques integrated into aerospace systems can be known as *dynamic data-driven aerospace systems*. For aerospace applications, where the operating conditions can be highly-dynamic both at the system level and the component level, DDDAS techniques can be employed to ensure highly-adaptive systems. *E.g.,* in a distributed aerospace application like DAC, an example of system-level conditions can be the dynamic message delays, which can be used by the formal DDDAS feedback loop in an aircraft to generate new progress guarantees at runtime; and an example of component-level conditions can be the dynamic operational intents or flight plans of the traffic aircraft, which can be used by the candidates to compute their conflict-free proposals using the DDDAS loop given in Fig. 7.1.

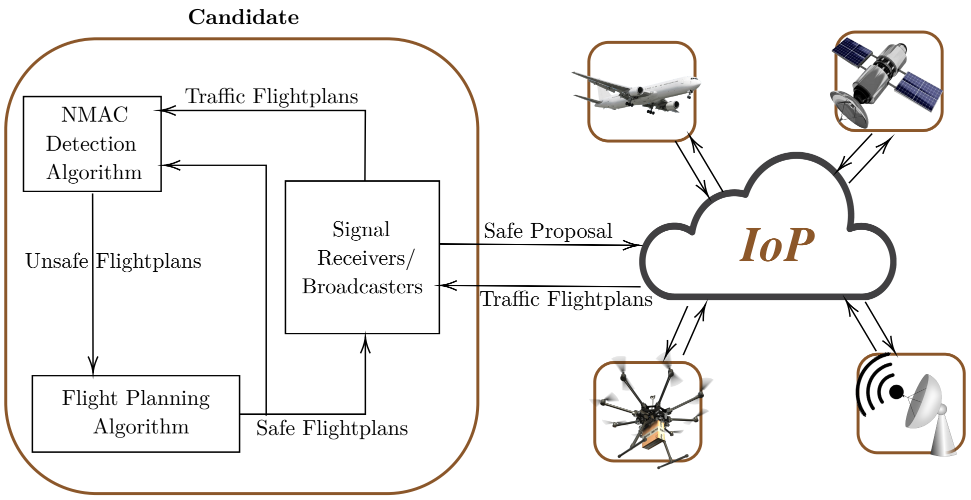


Figure 7.1: DDDAS feedback loop for high-fidelity conflict resolution in DAC.

There is significant scope for further investigation into the robust use of the DDDAS paradigm for the formal verification of stochastic aerospace systems. The DDDAS-based runtime verification approach that has been proposed in this thesis can be used for other aerospace applications apart from multi-aircraft coordination. *E.g.,* it may be possible to use the formal DDDAS feedback loop for generating formal properties at runtime with respect to data-driven stochastic state estimation applications [39].

7.4 Applications of the Proposed Techniques Beyond Aerospace

Apart from aerospace systems, multi-agent coordination can find direct applications in many other safety-critical areas such as autonomous cars [155], autonomous power grids [45], and industrial robots [87]. As an example, let us consider autonomous cars — in order to operate safely in the presence of one another, the cars have to collaboratively decide their operations. For this, they must make use of distributed coordination protocols for collaborative decision-making and operation planning. Similarly, the power loss, voltage deviation, and cost of the reactive power generation functions of distributed power generators can be optimized by using decentralized optimization techniques [92]. As all of these domains require distributed coordination among spatially separated and/or mobile agents, the techniques presented in this thesis for the verification of decentralized coordination in autonomous multi-agent systems are applicable for the verification of safety-critical properties of multi-agent coordination in all such domains.

7.5 Future Work

We discuss some potential future directions of work with regard to the verification of dynamic and stochastic systems, the verification of distributed algorithms, and the practical realization of our techniques for the verification of real-world applications.

Formal Verification of Dynamic and Stochastic Systems

First-class tool support for probabilistic properties Mechanically reasoning about probabilistic properties in proof assistants that are based on some form of predicate logic (*e.g.* – Athena) requires one to express probabilistic statements using predicate logic constructs. Rules of inference, such as *modus ponens*, are then used to construct proofs that can be mechanically checked by proof assistants to guarantee the *logical entailment* of a sentence S by a set of sentences θ. This approach of reasoning works well for situations when logical statements can have only Boolean values, but it creates an additional overhead when expressing and reasoning about probabilistic statements. This creates the need to develop proof systems that have first-class support for expressing probabilistic statements and reasoning about probabilistic properties without having to first express them in an ad-hoc manner using predicate logic constructs and then use predicate logic inference rules for proof engineering. It may be possible to create a proof system that uses a generalization of predicate logic where a sentence can have values ranging from 0 to 1 instead of Boolean values, which can be viewed as the probability of the sentence. Such a *probabilistic logic* can enable the mechanical determination of the probability of an arbitrary sentence S given a set Θ of sentences and their probabilities by *probabilistic entailment* [130]. Therefore, one important direction of future work is to investigate the development of proof systems that have first-class support for expressing and reasoning about probabilistic properties of stochastic safety-critical systems.

Learning-assisted runtime verification of dynamic systems In this thesis, we presented a formal DDDAS feedback loop that can be used by applications to update appropriate parameterized proofs at runtime, allowing the properties to retain the rigor of interactive theorem proving techniques. However, this approach is limited by the set of parameterized proofs that can be pre-developed before deployment – if the operating conditions change beyond the envelopes for the pre-developed proofs, then no guarantees can be provided. One potential approach to deal with such limitations of the framework proposed in this thesis can be to use *statistical learning* algorithms [81] that can generate parameterized properties by using runtime-observable parameters. It may be possible to verify such properties generated by learning algorithms on the basis of some fundamental axioms from the relevant domains. As ATPs can function without human interaction, they can be used to verify such generated properties at runtime in a completely autonomous manner. The approach can take the following general form — first, the sentinels observe real-time data to infer some attributes about the operating conditions, then these attributes can be fed to a learning algorithm to generate a system property. Finally, the generated properties can be fed to an ATP (or to multiple ATPs for sanity checking) along with a set of fundamental axioms to verify if the property is a logical consequence of the set of axioms. If successful, this approach will allow the generation of arbitrary properties at runtime, thereby resolving the scalability issues of the approach based on parameterized proofs. Therefore, investigating the development of such runtime verification techniques based on statistical learning algorithms and ATPs is an interesting future direction of work.

Provably Correct Distributed Multi-Agent Systems

Byzantine agents, message loss, and additional failure models The presence of Byzantine agents in a system creates significant challenges in verifying the correctness properties of distributed algorithms because the behavior of such agents cannot be modeled in advance. Moreover, if the agents have adversarial intentions, they may actively try to sabotage the correctness of distributed systems by strategically performing actions to that end. Message loss can also affect correctness since the eventual progress of most distributed algorithms relies on a certain set of messages being eventually delivered. If a system is prone to message loss, it can also affect the system's fairness conditions, making it necessary to identify some additional assumptions for proving progress. Another factor that can affect the correctness of distributed algorithms is agent failure, which can cause the topological structure of a distributed network to change, affecting the connectivity among agents. Our Failure-Aware Actor Model supports a certain type of failure where agents can temporarily or permanently get disconnected from the entire network, but not failures in which agents may only get disconnected from a subset of the network. Therefore, a potential direction of future work would be to investigate more flexible models of distributed computing that can be used for the verification of the correctness properties of distributed algorithms in the presence of Byzantine agents, message loss, and arbitrary agent failures.

Coordination in the presence of inconsistent and uncertain information In completely autonomous distributed multi-agent systems, it may be the case that during coordination, an agent receives a set of beliefs about some operational variable from other agents. Since beliefs are subjective in nature, such a set of beliefs may consist of arguments that reflect different views about the operational variable from the perspective of different agents. *E.g.,* in the case of autonomous ATM, an aircraft may receive conflicting information about the trajectory of a non-communicating aircraft, obtained through visual or radar inspection, from two other aircraft. Moreover, each of these arguments may have associated *likelihoods* with regard to some operational or ethical principles [65]. *E.g.,* if an aircraft receives inconsistent trajectory projections for a non-communicating aircraft from two other aircraft, each trajectory projection can have some associated likelihood of possible NMACs with respect to known traffic aircraft. In such situations, the recipient agent must adjudicate these arguments to derive a conclusion [64] for the purpose of coordination. One interesting direction of work is, therefore, the investigation of provably correct decentralized coordination algorithms that can be used when agents must adjudicate inconsistent and uncertain information received from other agents to derive some conclusion for safety-critical applications.

Morphable distributed computing informed by runtime verification Algorithms running over heterogenous airborne VANETs like the IoP will need to be adaptive in order to produce accurate computations from real-life data. Due to the dynamic nature of VANETs and the operational environments of mobile aircraft, it is important to judiciously determine how to efficiently run distributed algorithms over such networks. To address this, *morphable algorithms* can be designed to provide multiple computational approaches that can be used interchangeably according to the real-time operational conditions. These algorithms can be run inside of *morphable geo-spatial actors*, an abstraction to perform distributed computation over spatial regions – sections of actor-networks can be assigned specific parts of an airspace. At runtime, formal guarantees for dynamic conditions, such as probabilistic bounds on the round-trip message delays in a particular section of a network, can be used to automatically change the computation approach using morphable algorithms, thereby leading to more accurate real-time computation choices that are informed by real-time formal guarantees. As an example, let us assume that there are two algorithms available for use — C_1, which is centralized, and C_2, which is decentralized. Initially, C_2 is chosen for coordination as there is an assumption that the coordinator in C_1 will be a bottleneck. At runtime, there is a formal proof that the system conditions may not cause a bottleneck in the coordinator if C_1 is used. So, informed by this property, the system may choose to switch from C_2 to C_1 at runtime. Therefore, the investigation of how the formal runtime verification approaches can be used for morphable distributed computing is an interesting future direction of work.

Timely progress for non-deterministic distributed algorithms In Chapter 4, we have provided the theoretical foundation to reason about timely progress in distributed algorithms deployed over VANETs by stochastically modeling non-determinism in the operating conditions such as message transmission and processing delays. However, version 0.1 of our library does not have the theory to reason about non-determinism in the algorithms themselves. *E.g.,* in the Synod consensus protocol, since there is no restriction on when proposers can initiate new proposals, there may be cases where lower-numbered proposals are *interrupted* by higher-numbered proposals (see Chapter 2). Such interruptions can happen a finite number of times or can continue happening indefinitely. In the latter case, it is impossible to guarantee even eventual progress as the protocol will be stuck in a *livelock* scenario. Under the assumption of eventual progress, to probabilistically bound the time required for progress, the number of message rounds involved must be modeled. This can be done by making additional assumptions on the behavior of agents that can allow the number of message rounds to be probabilistically bounded (*e.g* – assuming that the proposers follow some retransmission algorithm like *exponential backoff* whose performance can be modeled [99]). Such assumptions may also enable the generation of more specific progress properties, such as exactly which proposer's proposal will be eventually chosen in the case of Synod. Therefore, one potential direction of future work is to investigate formal guarantees of timely progress for non-deterministic coordination algorithms and add additional theory to the Athena library to mechanically reason about such guarantees.

Bridging the gap between specifications and implementations Formal specifications of distributed algorithms developed for verification can often leave out important implementation details. *E.g.,* our proof of progress for Synod (Chapter 2) uses Lamport's high-level denotational specification of Synod to represent the behavior of the concurrent agents. The proof uses Lamport's strong assumptions about the behavior of non-faulty agents (*e.g.* – for a subset of agents, it assumes that implementations will not run into issues like infinite loops during local processing). It is a known issue that there are gaps in Lamport's specification of Synod which leaves out important implementation details. Therefore, different developers often make implementation choices for Synod that are widely different, especially when deciding when agents should retry. This results in unproven implementations that may not comply with the formally verified specification. It may be possible to create complete specifications that concretely define all implementation details in order to ensure that the implementations can follow the specifications. *E.g.,* the RAFT consensus algorithm assumes partial synchrony and specifies a complete implementation that uses *timeouts* to decide when agents should retry [131]. Therefore, one direction of future work is to investigate how specifications for distributed algorithms can be created that are both verifiable and complete, allowing the correctness properties to hold for the implementations.

From Theory to Practice

Verifiable runtime sentinels for monitoring real-life data streams Real-life data is seldom perfect – *e.g.*, the normality of sensor data may only be tested up to some significant level and may also be affected by pre-processing. Assuming normality of sensor data, it is also computationally expensive to effectively estimate the tail bounds of distributions in real-time. Therefore, it is important to investigate the development of formally-verified runtime sentinels that can find accurate and meaningful estimates from real-life data. Athena is a good candidate for this purpose because it has been demonstrated by Vargun *et al.* [159] that a *code-carrying theory* (CCT) approach can be used in Athena for generating executable code directly from a set of assertions and proofs by using a tool called CODEGEN. Therefore, once the proof refinement component generates new proofs during runtime, it may be possible to generate executable sentinel codes directly from the proofs. One potential direction of future work is, therefore, the investigation of techniques to generate executable sentinel codes directly from the specifications of high-level properties in our Athena library so that the properties can be accurately monitored against real-life data streams.

Generation of high-level assurance artifacts for certification Safety-critical software systems require rigorous certification procedures to verify their correctness and reliability. Certification agencies like the Federal Aviation Administration rely on specifications and requirements outlined in documents such as the DO-178C, Software Considerations in Airborne Systems and Equipment Certification [151], for certifying UAS operations. Formal theorem proving can be extremely effective in the rigorous verification of such systems, but typically theorem proving works at a very abstract level, and the artifacts generated by theorem proving techniques require specialization to comprehend. Therefore, it is necessary to bridge the gap between such formally rigorous techniques and requirement-level terminologies used by certification authorities. *Assurance cases* are artifacts for conveying system requirements that provide a structural argument that a system satisfies some desired safety, security, or reliability properties and are used as compelling high-level arguments for system certification [84]. It has been demonstrated that assurance cases can be designed in specialized languages that are amenable to verification [118] and can be automatically generated from the evidence produced by traditional verification tools like model checkers [119]. One potential future direction of work is to investigate approaches to automatically generate assurance case fragments for certification directly from the lower-level proof artifacts obtained from Athena's proof checking environment. This will provide the capability to utilize Athena's theorem proving techniques to provide higher-level evidence for certification that can be directly comprehended by the certification authorities.

Trusted and transparent automated theorem proving Automated theorem provers like SPASS are powerful verification tools that can be used for automatically checking if correctness properties are consequences of a given set of statements. They enable the utilization of mathematical theorem proving techniques for the analysis of real-world complex systems without having to interactively develop the proofs. Because of their automatic nature, they can be used to aid in the verification of properties that can involve a large number of intermediate proof obligations. However, one common issue with using ATPs for proof generation is that if an inconsistent set of statements is provided to the ATP as a context to prove a proof obligation, the ATP will always succeed in generating a proof, even if the proof obligation is not true. For this reason, in order to be able to trust the result of an ATP, it must be ensured that the context sent to the ATP is consistent. This creates additional overhead on the part of users since proof systems like Athena or TLAPS do not have any mechanism to automatically check for the consistency of the contexts that are sent to the back-end ATPs. In the case of significantly complex contexts, checking for consistency is impossible, which makes it challenging to use ATPs with complex contexts in a trusted manner. Moreover, in our experience, once an ATP successfully proves an obligation, Athena or TLAPS do not have any mechanism to obtain the actual logical proof of the property that was proven by the ATP. Therefore, another important direction of future work is to investigate techniques that can allow the use of ATPs in a more trusted and transparent manner for aiding in large-scale verification tasks involving theorem proving.

Experimental evaluation of theoretical results In this thesis, we have presented a theoretical analysis of decentralized coordination algorithms for vehicular systems. We have also proposed a data-driven approach for the runtime verification of dynamic distributed applications to extend the practicality of formal theorem proving techniques beyond the pre-deployment stages. Before these concepts can be practically realized in real-world systems, significant experimental evaluation of these techniques is necessary to determine their efficiency and feasibility for use in the face of real-life uncertainties. Therefore, an important direction of future work is the development of appropriate platforms for the evaluation of the decentralized coordination algorithms and the formal DDDAS feedback loop by using either software-based simulation or actual deployment of the algorithms over airborne drones.

REFERENCES

[1] Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. The MIT Press, Cambridge, MA (1986). doi:10.7551/mitpress/1086.001.0001

[2] Agha, G., Mason, I.A., Smith, S., Talcott, C.: Towards a theory of actor computation. In: International Conference on Concurrency Theory. pp. 565–579 (1992). doi:10.1007/bfb0084816

[3] Agha, G.A., Mason, I.A., Smith, S.F., Talcott, C.L.: A foundation for actor computation. J. Func. Prog. **7**(1), 1–72 (1997). doi:10.1017/s095679689700261x

[4] Al Hanbali, A., Kherani, A.A., Nain, P.: Simple models for the performance evaluation of a class of two-hop relay protocols. In: International Conference on Research in Networking. pp. 191–202 (2007). doi:10.1007/978-3-540-72606-7_17

[5] Alejo, D., Cobano, J.A., Heredia, G., Ollero, A.: Collision-free 4D trajectory planning in unmanned aerial vehicles for assembly and structure construction. J. Int. Rob. Syst. **73**(1-4), 783–795 (2014). doi:10.1007/s10846-013-9948-x

[6] Alquraan, A., Takruri, H., Alfatafta, M., Al-Kiswany, S.: An analysis of network-partitioning failures in cloud systems. In: USENIX Symposium on Operating Systems Design and Implementation. pp. 51–68 (2018)

[7] Arkoudas, K., Musser, D.: Fundamental Proof Methods in Computer Science: A Computer-Based Approach. MIT Press, Cambridge, MA (2017). doi:10.1017/s1471068420000071

[8] Atif, M.: Analysis and verification of two-phase commit & three-phase commit protocols. In: International Conference on Emerging Technologies. pp. 326–331 (2009). doi:10.1109/icet.2009.5353152

[9] Attiya, H., Djerassi-Shintel, T.: Time bounds for decision problems in the presence of timing uncertainty and failures. J. Par. Dist. Comp. **61**(8), 1096–1109 (2001). doi:10.1006/jpdc.2001.1730

[10] Attiya, H., Dwork, C., Lynch, N., Stockmeyer, L.: Bounds on the time to reach agreement in the presence of timing uncertainty. J. ACM (JACM) **41**(1), 122–152 (1994). doi:10.21236/ada229766

[11] Bainomugisha, E., Vallejos, J., Tanter, E., Boix, E.G., Costanza, P., De Meuter, W., D'Hondt, T.: Resilient Actors: A runtime partitioning model for pervasive computing services. In: International Conference on Pervasive Services. p. 31–40 (2009). doi:10.1145/1568199.1568205

[12] Balachandran, S., Muñoz, C., Consiglio, M.: Distributed consensus to enable merging and spacing of UAS in an urban environment. In: International Conference on Unmanned Aircraft Systems (ICUAS). pp. 670–675 (2018). doi:10.1109/icuas.2018.8453460

[13] Balachandran, S., Muñoz, C.A., Consiglio, M.C., Feliú, M.A., Patel, A.V.: Independent configurable architecture for reliable operation of unmanned systems with distributed onboard services. In: AIAA/IEEE Digital Avionics Systems Conference (DASC). pp. 1–6 (2018). doi:10.1109/dasc.2018.8569752

[14] Balachandran, S., Narkawicz, A., Muñoz, C., Consiglio, M.: A path planning algorithm to enable well-clear low altitude UAS operation beyond visual line of sight. In: USA/Europe Air Traffic Management Research and Development Seminar. pp. 26–35 (2017)

[15] Berman, P., Garay, J.A., Perry, K.J., et al.: Towards optimal distributed consensus. In: Annual Symposium on Foundations of Computer Science. vol. 89, pp. 410–415 (1989). doi:10.1109/sfcs.1989.63511

[16] Bertsekas, D.P., Gallager, R.G., Humblet, P.: Data Networks, vol. 2. Prentice-Hall International, Hoboken, NJ (1992)

[17] Bhasin, K., Hayden, J.: Space internet architectures and technologies for NASA enterprises. In: IEEE Aerospace Conference. pp. 931–941 (2001). doi:10.1109/AERO.2001.931275

[18] Bickford, M., Constable, R.L., Rahli, V.: Logic of Events, a framework to reason about distributed systems. In: Languages for Distributed Algorithms Workshop. pp. 1–2 (2012)

[19] Bocchi, L., Chen, T.C., Demangeon, R., Honda, K., Yoshida, N.: Monitoring networks through multiparty session types. J. Theo. Comp. Sci. **669**, 33–58 (2017). doi:10.1016/j.tcs.2017.02.009

[20] Bocchi, L., Murgia, M., Vasconcelos, V.T., Yoshida, N.: Asynchronous timed session types. In: 28th European Symposium on Programming (ESOP). pp. 583–610 (2019). doi:10.1007/978-3-030-17184-1_21

[21] Breese, S., Kopsaftopoulos, F., Varela, C.: Towards proving runtime properties of data-driven systems using safety envelopes. In: 12th International Workshop on Structural Health Monitoring. pp. 1–12 (2019). doi:10.12783/shm2019/32302

[22] Bringsjord, S., Govindarajulu, N.S., Giancola, M.: Automated argument adjudication to solve ethical problems in multi-agent environments. Paladyn, J. Bhv. Rob. **12**(1), 310–335 (2021). doi:doi:10.1515/pjbr-2021-0009

[23] Brooker, P.: The Uberlingen accident: macro-level safety lessons. Sfty. Sci. **46**(10), 1483–1508 (2008). doi:10.1016/j.ssci.2007.10.001

[24] Burrows, M.: The Chubby Lock Service for loosely-coupled distributed systems. In: USENIX Symposium on Operating Systems Design and Implementation. pp. 335–350 (2006)

[25] Cao, X., Yang, P., Alzenad, M., Xi, X., Wu, D., Yanikomeroglu, H.: Airborne communication networks: a survey. J. Sel. Ars. Comm. **36**(9), 1907–1926 (2018). doi:10.1109/jsac.2018.2864423

[26] Chamberlain, J.P., Consiglio, M.C., Muñoz, C.: DANTi: detect and avoid in the cockpit. In: AIAA Aviation Technology, Integration, and Operation Conference. pp. 1–11 (2017). doi:10.2514/6.2017-4491

[27] Champion, A., Mebsout, A., Sticksel, C., Tinelli, C.: The Kind 2 model checker. In: International Conference on Computer Aided Verification. pp. 510–517 (2016). doi:10.1007/978-3-319-41540-6_29

[28] Chand, S., Liu, Y.A., Stoller, S.D.: Formal verification of Multi-Paxos for distributed consensus. In: International Symposium on Formal Methods. pp. 119–136 (2016). doi:10.1007/978-3-319-48989-6_8

[29] Chaouch, D.: Formalization of continuous time Markov Chains with applications in Queueing Theory. Master's thesis, Concordia University (2015)

[30] Charalambides, M., Dinges, P., Agha, G.: Parameterized, concurrent Session Types for asynchronous multi-actor interactions. Sci. Comp. Prog. **115-116**, 100–126 (2016). doi:10.1016/j.scico.2015.10.006

[31] Charalambides, M., Palmskog, K., Agha, G.: Types for progress in actor programs. In: Boreale, M., Corradini, F., Loreti, M., Pugliese, R. (eds.) Models, Languages, and Tools for Concurrent and Distributed Programming, pp. 315–339. Springer, New York, NY (2019). doi:10.1007/978-3-030-21485-2_18

[32] Charron-Bost, B., Schiper, A.: The Heard-Of Model: Computing in distributed systems With benign faults. Dist. Comp. **22**(1), 49–71 (2009). doi:10.1007/s00446-009-0084-6

[33] Chaudhuri, K., Doligez, D., Lamport, L., Merz, S.: Verifying safety properties with the TLA+ Proof System. In: International Joint Conference on Automated Reasoning. pp. 142–148 (2010). doi:10.1007/978-3-642-14203-1_12

[34] Chlamtac, I., Conti, M., Liu, J.J.: Mobile ad hoc networking: Imperatives and challenges. Ad Hoc Net. **1**(1), 13–64 (2003). doi:10.1016/S1570-8705(03)00013-1

[35] Chlipala, A.: Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant. MIT Press, Cambridge, MA (2013). doi:10.7551/mitpress/9153.003.0002

[36] Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., et al.: Handbook of Model Checking, vol. 10. Springer, New York, NY (2018). doi:10.1007/978-3-319-10575-8

[37] Colbert, B.K., Slagel, J.T., Crespo, L.G., Balachandran, S., Muñoz, C.: PolySafe: A formally verified algorithm for conflict detection on a polynomial airspace. IFAC-PapersOnLine **53**(2), 15615–15620 (2020). doi:10.1016/j.ifacol.2020.12.2496

[38] Consiglio, M., Muñoz, C., Hagen, G., Narkawicz, A., Balachandran, S.: ICAROUS: Integrated Configurable Algorithms for Reliable Operations of Unmanned Systems. In: IEEE/AIAA Digital Avionics Systems Conference (DASC). pp. 1–5 (2016). doi:10.1109/dasc.2016.7778033

[39] Cruz-Camacho, E., Paul, S., Kopsaftopoulos, F., Varela, C.A.: Towards provably correct probabilistic flight systems. In: Dynamic Data Driven Application Systems (DDDAS). pp. 236–244 (2020). doi:10.1007/978-3-030-61725-7_28

[40] Darema, F.: Dynamic Data-Driven Application Systems: A new paradigm for application simulations and measurements. In: International Conference on Computational Science. pp. 662–669 (2004). doi:10.1007/978-3-540-24688-6_86

[41] De Prisco, R., Lampson, B., Lynch, N.: Revisiting the Paxos algorithm. Theo. Comp. Sci. **243**(1-2), 35–91 (2000). doi:10.1016/s0304-3975(00)00042-6

[42] Debrat, H., Merz, S.: Verifying fault-tolerant distributed algorithms in the Heard-Of Model. Tech. rep., National Institute for Research in Digital Science and Technology, France (2012)

[43] DeGarmo, M., Maroney, D.: NEXTGEN and SESAR: Opportunities for UAS Integration. In: Congress of International Council of Aeronautical Sciences. pp. 1–14 (2008). doi:10.2514/6.2008-8925

[44] Delporte-Gallet, C., Fauconnier, H., Guerraoui, R., Pochon, B.: The perfectly synchronized round-based model of distributed computing. Inf. Comp. **205**(5), 783–815 (2007). doi:10.1016/j.ic.2006.11.003

[45] Dörfler, F., Bolognani, S., Simpson-Porco, J.W., Grammatico, S.: Distributed control and optimization for autonomous power grids. In: European Control Conference (ECC). pp. 2436–2453 (2019). doi:10.23919/ECC.2019.8795974

[46] Dowek, G., Muñoz, C., Carreño, V.: Provably safe coordinated strategy for distributed conflict resolution. In: AIAA Guidance, Navigation, and Control Conference. p. 6047 (2005). doi:10.2514/6.2005-6047

[47] Drăgoi, C., Henzinger, T.A., Zufferey, D.: PSync: A partially synchronous language for fault-tolerant distributed algorithms. ACM SIGPLAN Notices **51**(1), 400–415 (2016). doi:10.1145/2837614.2837650

[48] Duan, J., Yi, X., Zhao, S., Wu, C., Cui, H., Le, F.: NFVactor: A resilient NFV system using the distributed actor model. J. Sel. Ars. Comm. **37**(3), 586–599 (2019). doi:10.1109/JSAC.2019.2894287

[49] Dutle, A., Muñoz, C., Conrad, E., Goodloe, A., Perez, I., Balachandran, S., Giannakopoulou, D., Mavridou, A., Pressburger, T., et al.: From requirements to autonomous flight: An overview of the monitoring ICAROUS project. In: Formal Methods for Autonomous Systems (FMAS). pp. 73–91 (2020). doi:10.4204/EPTCS.329.3

[50] Dwork, C., Moses, Y.: Knowledge and common knowledge in a Byzantine environment I: crash failures. In: Conference on Theoretical Aspects of Reasoning about Knowledge. pp. 149–169 (1986)

[51] Eby, M.S., Kelly, W.E.: Free flight separation assurance using distributed algorithms. In: IEEE Aerospace Conference. pp. 429–441 (1999). doi:10.1109/AERO.1999.793186

[52] Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Trans. Dat. Syst. **22**(3), 364–418 (1997). doi:10.1145/261124.261126

[53] Fagin, R., Halpern, J.Y.: Reasoning about knowledge and probability. J. ACM (JACM) **41**(2), 340–367 (1994). doi:10.1145/174652.174658

[54] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press, Cambridge, MA (2004). doi:10.7551/mitpress/5803.001.0001

[55] Fagin, R., Halpern, J.Y., Vardi, M.Y.: What can machines know? On the properties of knowledge in distributed systems. J. ACM (JACM) **39**(2), 328–376 (1992). doi:10.1145/128749.150945

[56] Fagin, R., Vardi, M.Y.: Knowledge and implicit knowledge in a distributed environment: preliminary report. In: Conference on Theoretical Aspects of Reasoning About Knowledge. pp. 187–206 (1986)

[57] Federal Aviation Administration: Introduction to TCAS-II Version 7.1 (2011)

[58] Federal Aviation Administration: Concept of operations v2.0: Unmanned aircraft systems (UAS) traffic management (UTM) (2020), `https://www.faa.gov/uas/research_development/traffic_management/media/UTM_ConOps_v2.pdf`, Accessed: Feb. 2022

[59] Field, J., Varela, C.A.: Transactors: A programming model for maintaining globally consistent distributed state in unreliable environments. In: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 195–208 (2005). doi:10.1145/1040305.1040322

[60] Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM (JACM) **32**(2), 374–382 (1985). doi:10.1145/588058.588060

[61] Frew, E.W., Brown, T.X.: Airborne communication networks for small unmanned aircraft systems. Proc. IEEE **96**(12), 2008–2027 (2008). doi:10.1109/JPROC.2008.2006127

[62] Galdino, A.L., Muñoz, C., Ayala-Rincón, M.: Formal verification of an optimal air traffic conflict resolution and recovery algorithm. In: International Workshop on Logic, Language, Information, and Computation. pp. 177–188 (2007). doi:10.1007/978-3-540-73445-1_13

[63] Gallager, R.G.: Stochastic Processes: Theory for Applications. Cambridge University Press, Cambridge, UK (2013). doi:10.1017/cbo9781139626514

[64] Giancola, M., Bringsjord, S., Govindarajulu, N.S., Licato, J.: Adjudication of symbolic & connectionist arguments in autonomous driving AI. In: Global Conference on Artificial Intelligence (GCAI). pp. 28–33 (2020). doi:10.29007/k647

[65] Giancola, M., Bringsjord, S., Govindarajulu, N.S., Varela, C.: Ethical reasoning for autonomous agents under uncertainty. In: International Conference on Robot Ethics and Standards (ICRES). pp. 1–16 (2020)

[66] Gifford, J.L., Sinha, P.: Airport congestion and near-midair collisions. Trans. Res. Prt. A: General **25**(2), 91–99 (1991). doi:10.1016/0191-2607(91)90128-D

[67] Gordon, M.J.: Mechanizing programming logics in Higher Order Logic. In: Current Trends in Hardware Verification and Automated Theorem Proving, pp. 387–439. Springer, New York, NY (1989). doi:10.1007/978-1-4612-3658-0_10

[68] Grossglauser, M., Tse, D.N.: Mobility increases the capacity of ad hoc wireless networks. IEEE/ACM Trans. Net. **10**(4), 477–486 (2002). doi:10.1109/tnet.2002.801403

[69] Guzmán, M., Knight, S., Quintero, S., Ramírez, S., Rueda, C., Valencia, F.: Reasoning about distributed knowledge of groups with infinitely many agents. In: International Conference on Concurrency Theory (CONCUR) (2019). doi:10.4230/LIPIcs.CONCUR.2019.29

[70] Halpern, J.Y.: Using reasoning about knowledge to analyze distributed systems. Annual Review of Comp. Sci. **2**(1), 37–68 (1987)

[71] Halpern, J.Y., Fagin, R.: Modelling knowledge and action in distributed systems. Dist. Comp. **3**(4), 159–177 (1989). doi:10.1007/BF01784885

[72] Halpern, J.Y., Vardi, M.Y.: The complexity of reasoning about knowledge and time. I. lower bounds. J. Comp. Syst. Sci. **38**(1), 195–237 (1989). doi:10.1016/0022-0000(89)90039-1

[73] Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press, Cambridge, UK (2009)

[74] Hartenstein, H., Laberteaux, L.: A tutorial survey on vehicular ad-hoc networks. IEEE Comm. Mag. **46**(6), 164–171 (2008). doi:10.1109/MCOM.2008.4539481

[75] Hasan, O.: Probabilistic analysis using theorem proving. In: International Conference on Theorem Proving in Higher Order Logics. pp. 1–12 (2008)

[76] Hasan, O., Tahar, S.: Formalization of the standard uniform random variable. Theo. Comp. Sci. **382**, 71–83 (2007). doi:10.1016/j.tcs.2007.05.009

[77] Hasan, O., Tahar, S.: Using theorem proving to verify expectation and variance for discrete random variables. J. Auto. Reasoning **41**(3-4), 295–323 (2008). doi:10.1007/s10817-008-9113-6

[78] Hasan, O., Tahar, S.: Formal verification of tail distribution bounds in the HOL Theorem Prover. Math. Meth. App. Sci. **32**(4), 480–504 (2009). doi:10.1002/mma.1055

[79] Hasan, O., Tahar, S.: Formal probabilistic analysis: A Higher-Order Logic based approach. In: International Conference on Abstract State Machines. pp. 2–19 (2010). doi:10.1007/978-3-642-11811-1_2

[80] Hasan, O., Tahar, S.: Reasoning about conditional probabilities in a Higher-Order-Logic theorem prover. J. App. Log. **9**(1), 23–40 (2011). doi:10.1016/j.jal.2011.01.001

[81] Hastie, T., Tibshirani, R., Friedman, J.H., Friedman, J.H.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, vol. 2. Springer, New York, NY (2009)

[82] Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: IronFleet: Proving practical distributed systems correct. In: Symposium on Operating Systems Principles. pp. 1–17 (2015). doi:10.1145/2815400.2815428

[83] Hawblitzel, C., Howell, J., Kapritsos, M., Lorch, J.R., Parno, B., Roberts, M.L., Setty, S., Zill, B.: IronFleet: Proving safety and liveness of practical distributed systems. Comm. ACM **60**(7), 83–92 (2017). doi:10.1145/3068608

[84] Hawkins, R., Habli, I., Kelly, T., McDermid, J.: Assurance cases and prescriptive software safety certification: A comparative study. Sfty. Sci. **59**, 55–71 (2013). doi:10.1016/j.ssci.2013.04.007

[85] Hewitt, C.: Viewing control structures as patterns of passing messages. Art. Intll. **8**(3), 323–364 (1977). doi:10.1016/0004-3702(77)90033-9

[86] Howard, H., Malkhi, D., Spiegelman, A.: Flexible Paxos: Quorum intersection revisited. arXiv preprint (2016). doi:10.48550/arXiv.1608.06696

[87] Hryniewicz, P., Banas, W., Foit, K., Gwiazda, A., Sekala, A.: Modelling cooperation of industrial robots as multi-agent systems. In: IOP Conference Series: Materials Science and Engineering. pp. 012–061 (2017). doi:10.1088/1757-899x/227/1/012061

[88] Hurlburt, J., Wargo, C.: Development and demonstration of the NASA Small Aircraft Transportation System (SATS) Airborne Internet (AI). In: IEEE/AIAA Digital Avionics Systems Conference (DASC). pp. 10A1–10A1 (2002). doi:10.1109/DASC.2002.1052961

[89] Interlandi, M.: Reasoning about knowledge in distributed systems using Datalog. In: International Datalog 2.0 Workshop. pp. 99–110 (2012). doi:10.1007/978-3-642-32925-8_11

[90] Jackson, D.: Alloy: A lightweight object modelling notation. ACM Trans. on Sftwr. Eng. and Meth. **11**(2), 256–290 (2002). doi:10.1145/505145.505149

[91] Keidar, I., Rajsbaum, S.: Open questions on consensus performance in well-behaved runs. In: Future Directions in Distributed Computing, pp. 35–39. Springer, New York, NY (2003). doi:10.1007/3-540-37795-6_7

[92] Khan, I., Xu, Y., Sun, H., Bhattacharjee, V.: Distributed optimal reactive power control of power systems. IEEE Access **6**, 7100–7111 (2017). doi:10.1109/ACCESS.2017.2779806

[93] Kirsch, J., Amir, Y.: Paxos for system builders: An overview. In: Workshop on Large-Scale Distributed Systems and Middleware. pp. 1–6 (2008). doi:10.1145/1529974.1529979

[94] Knight, S., Maubert, B., Schwarzentruber, F.: Reasoning about knowledge and messages in asynchronous multi-agent systems. Math. Struc. Comp. Sci. **29**(1), 127–168 (2019). doi:10.1017/s0960129517000214

[95] Kshemkalyani, A.: On continuously attaining levels of concurrent knowledge without control messages. Tech. rep., University of Illinois at Chicago (1998)

[96] Küfner, P., Nestmann, U., Rickmann, C.: Formal verification of distributed algorithms. In: IFIP International Conference on Theoretical Computer Science. pp. 209–224 (2012). doi:10.1007/978-3-642-33475-7_15

[97] Kushwah, R., Tapaswi, S., Kumar, A.: Multipath delay analysis using Queuing Theory for gateway selection in hybrid MANET. Wrlss. Prsnl. Comm. **111**(1), 9–32 (2020). doi:10.1007/s11277-019-06842-9

[98] Kuznets, R., Prosperi, L., Schmid, U., Fruzsa, K.: Epistemic reasoning with Byzantine-faulty agents. In: International Symposium on Frontiers of Combining Systems. pp. 259–276 (2019). doi:10.1007/978-3-030-29007-8_15

[99] Kwak, B.J., Song, N.O., Miller, L.E.: Performance analysis of Exponential Backoff. IEEE/ACM Trans. Net. **13**(2), 343–355 (2005). doi:10.1109/TNET.2005.845533

[100] Lamport, L.: The part-time parliament. ACM Trans. Comp. Syst. **16**(2), 133–169 (1998). doi:10.1145/279227.279229

[101] Lamport, L.: Paxos made simple. ACM SIGACT News **32**(4), 18–25 (2001)

[102] Lamport, L.: Specifying systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman, Boston, MA (2002)

[103] Lamport, L.: Real-time model checking is really simple. In: Advanced Research Working Conference on Correct Hardware Design and Verification Methods. pp. 162–175 (2005). doi:10.1007/11560548_14

[104] Lamport, L.: Fast Paxos. Dist. Comp. **19**(2), 79–103 (2006). doi:10.1007/s00446-006-0005-x

[105] Lamport, L., Malkhi, D., Zhou, L.: Vertical Paxos and primary-backup replication. In: ACM Symposium on Principles of Distributed Computing. pp. 312–313 (2009). doi:10.1145/1582716.1582783

[106] Last, G., Penrose, M.: Lectures on the Poisson Process, vol. 7. Cambridge University Press, Cambridge, UK (2017). doi:10.1017/9781316104477.007

[107] Lee, S.M., Park, C., Johnson, M.A., Mueller, E.R.: Investigating effects of well clear definitions on UAS Sense-and-avoid operations in enroute and transition airspace. In: Aviation Technology, Integration, and Operations Conference. p. 4308 (2013). doi:10.2514/6.2013-4308

[108] Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: International Conference on Logic for Programming Artificial Intelligence and Reasoning. pp. 348–370 (2010). doi:10.1007/978-3-642-17511-4_20

[109] Leon-Garcia, A.: Probability and Random Processes for Electrical Engineering, 2nd edn. Addison-Wesley Longman, Boston, MA (1994)

[110] Lipsky, L.: M/M/1 Queue. In: Queueing Theory: A Linear Algebraic Approach, pp. 33–75. Springer, New York, NY (2009). doi:10.1007/978-0-387-49706-8_2

[111] Little, J.D.: A proof for the queuing formula: L= λ W. Operations Research **9**(3), 383–387 (1961). doi:10.1287/opre.9.3.383

[112] Liu, J., Jiang, X., Nishiyama, H., Kato, N.: On the delivery probability of two-hop relay MANETs with erasure coding. IEEE Trans. Comm. **61**(4), 1314–1326 (2013). doi:10.1109/tcomm.2013.020413.120198

[113] Luckner, R., Höhne, G., Fuhrmann, M.: Hazard criteria for wake vortex encounters during approach. Aero. Sci. Tech. **8**(8), 673–687 (2004). doi:10.1016/j.ast.2004.06.008

[114] Malkhi, D., Lamport, L., Zhou, L.: Stoppable Paxos. Tech. rep., Microsoft Research (2008)

[115] McMillan, K.L., Padon, O.: Deductive verification in decidable fragments with Ivy. In: International Static Analysis Symposium. pp. 43–55 (2018). doi:10.1007/978-3-319-99725-4_4

[116] Medina, D., Hoffmann, F.: The Airborne Internet. In: Plass, S. (ed.) Future Aeronautical Communications, chap. 17. IntechOpen, London, UK (2011). doi:10.5772/29680

[117] Meester, R., Slooten, K.: Some Philosophy of Probability, Statistics, and Forensic Science. Cambridge University Press, Cambridge, UK (2021). doi:10.1017/9781108596176.002

[118] Meng, B., Moitra, A., Crapo, A.W., Paul, S., Siu, K., Durling, M., Prince, D., Herencia-Zapana, H.: Towards developing formalized assurance cases. In: AIAA/IEEE Digital Avionics Systems Conference (DASC). pp. 1–9 (2020). doi:10.1109/DASC50938.2020.9256740

[119] Meng, B., Paul, S., Moitra, A., Siu, K., Durling, M.: Automating the assembly of security assurance case fragments. In: International Conference on Computer Safety, Reliability, and Security (SAFECOMP). pp. 101–114 (2021). doi:10.1007/978-3-030-83903-1_7

[120] Mhamdi, T., Hasan, O., Tahar, S.: On the formalization of the Lebesgue Integration Theory in HOL. In: International Conference on Interactive Theorem Proving. pp. 387–402 (2010). doi:10.1007/978-3-642-14052-5_27

[121] Mhamdi, T., Hasan, O., Tahar, S.: Formalization of Measure Theory and Lebesgue Integration for probabilistic analysis in HOL. ACM Trans. on Emb. Comp. Syst. **12**(1) (2013). doi:10.1145/2406336.2406349

[122] Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Formal Methods in System Design **49**(1-2), 33–74 (2016). doi:10.1007/s10703-016-0241-z

[123] Monin, J.F.: Understanding Formal Methods. Springer Science & Business Media, New York, NY (2012)

[124] Muñoz, C., Narkawicz, A., Chamberlain, J.: A TCAS-II resolution advisory detection algorithm. In: AIAA Guidance, Navigation, and Control Conference. p. 4622 (2013). doi:10.2514/6.2013-4622

[125] Muñoz, C., Narkawicz, A., Chamberlain, J., Consiglio, M.C., Upchurch, J.M.: A family of well-clear boundary models for the integration of UAS in the NAS. In: AIAA Aviation Technology, Integration, and Operation Conference. p. 2412 (2014). doi:10.2514/6.2014-2412

[126] Muñoz, C., Narkawicz, A., Hagen, G., Upchurch, J., Dutle, A., Consiglio, M., Chamberlain, J.: DAIDALUS: Detect and Avoid Alerting Logic for Unmanned Systems. In: AIAA/IEEE Digital Avionics Systems Conference (DASC). pp. 1–12 (2015). doi:10.1109/DASC.2015.7311421

[127] Musser, D.R., Varela, C.A.: Structured reasoning about actor systems. In: Workshop on Programming Based on Actors, Agents, and Decentralized Control. pp. 37–48 (2013). doi:10.1145/2541329.2541334

[128] Narkawicz, A., Muñoz, C., Dutle, A.: Coordination logic for repulsive resolution maneuvers. In: AIAA Aviation Technology, Integration, and Operation Conference. pp. 31–56 (2016). doi:10.2514/6.2016-3156

[129] Naumov, P., Stehr, M.O., Meseguer, J.: The HOL/NuPRL Proof Translator. In: International Conference on Theorem Proving in Higher Order Logics. pp. 329–345 (2001). doi:10.1007/3-540-44755-5_23

[130] Nilsson, N.J.: Probabilistic Logic. Art. Intell. **28**(1), 71–87 (1986). doi:10.1016/0004-3702(86)90031-7

[131] Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: USENIX Annual Technical Conference. pp. 305–319 (2014)

[132] Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: International Conference on Automated Deduction. pp. 748–752 (1992). doi:10.1007/3-540-55602-8_217

[133] Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: Decidable reasoning about distributed protocols. In: Object-oriented Programming, Systems, Languages, and Applications. p. 108 (2017). doi:10.1145/3140568

[134] Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. ACM SIGPLAN Notices **51**(6), 614–630 (2016). doi:10.1145/2980983.2908118

[135] Paul, S.: Emergency trajectory generation for fixed-wing aircraft. Master's thesis, Rensselaer Polytechnic Institute (Dec 2018)

[136] Paul, S., Agha, G.A., Patterson, S., Varela, C.A.: Verification of eventual consensus in Synod using a Failure-Aware Actor Model. In: NASA Formal Methods Symposium (NFM). pp. 249–267 (2021). doi:10.1007/978-3-030-76384-8_16

[137] Paul, S., Hole, F., Zytek, A., Varela, C.A.: Wind-aware trajectory planning For fixed-wing aircraft in loss of thrust emergencies. In: AIAA/IEEE Digital Avionics Systems Conference (DASC). pp. 558–567 (2018). doi:10.1109/DASC.2018.8569842

[138] Paul, S., Kopsaftopoulos, F., Patterson, S., Varela, C.A.: Dynamic data-driven formal progress envelopes for distributed algorithms. In: Dynamic Data-Driven Application Systems. pp. 245–252 (2020). doi:10.1007/978-3-030-61725-7_29

[139] Paul, S., Patterson, S., Varela, C.A.: Conflict-aware flight planning for avoiding near mid-air collisions. In: AIAA/IEEE Digital Avionics Systems Conference (DASC). pp. 1–10 (2019). doi:10.1109/dasc43569.2019.9081658

[140] Paul, S., Patterson, S., Varela, C.A.: Collaborative situational awareness for conflict-aware flight planning. In: IEEE/AIAA Digital Avionics Systems Conference (DASC). pp. 1–10 (2020). doi:10.1109/dasc50938.2020.9256620

[141] Paul, S., Patterson, S., Varela, C.A.: Formal guarantees of timely progress for distributed knowledge propagation. In: Formal Methods for Autonomous Systems (FMAS). pp. 73–91 (2021). doi:10.4204/EPTCS.348.5

[142] Pike, L., Wegmann, N., Niller, S., Goodloe, A.: Copilot: monitoring embedded systems. Innovations in Syst. and Sftwr. Eng. **9**(4), 235–255 (2013). doi:10.1007/s11334-013-0223-x

[143] Pontani, M., Conway, B.A.: Particle Swarm Optimization applied to space trajectories. J. Guid., Cont., Dyn. **33**(5), 1429–1441 (2010)

[144] Popovic, I., Vrtunski, V., Popovic, M.: Formal verification of distributed transaction management in a SOA based control system. In: IEEE International Conference and Workshops on Engineering of Computer-Based Systems. pp. 206–215 (2011). doi:10.1109/ECBS.2011.14

[145] Pritchett, A.R., Genton, A.: Negotiated decentralized aircraft conflict resolution. IEEE Trans. on Intell. Trans. Syst. **19**(1), 81–91 (2018)

[146] Qasim, M.: Formalization of normal random variables. Master's thesis, Concordia University (2016). doi:10.1016/j.entcs.2013.09.001

[147] Queille, J.P., Sifakis, J.: Fairness and related properties in transition systems — A temporal logic to deal with fairness. Acta Informatica **19**(3), 195–220 (1983). doi:10.1007/bf00265555

[148] Rahli, V., Guaspari, D., Bickford, M., Constable, R.L.: Formal specification, verification, and implementation of fault-tolerant systems Using EventML. Elctrnc. Comm. of the EASST **72**, 1–15 (2015). doi:10.14279/tuj.eceasst.72.1013

[149] Rahli, V., Guaspari, D., Bickford, M., Constable, R.L.: EventML: Specification, verification, and implementation of crash-tolerant state machine replication systems. Sci. Comp. Prog. **148**, 26–48 (2017). doi:10.1016/j.scico.2017.05.009

[150] Ren, W., Beard, R.W.: Distributed Consensus in Multi-Vehicle Cooperative Control. Springer, London, UK (2008). doi:10.1007/978-1-84800-015-5

[151] Rierson, L.: Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance. CRC Press, Boca Raton, FL (2017)

[152] Schiper, N., Rahli, V., Van Renesse, R., Bickford, M., Constable, R.L.: Developing correctly replicated databases using formal tools. In: IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 395–406 (2014). doi:10.1109/dsn.2014.45

[153] Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM Comp. Surveys (CSUR) **22**(4), 299–319 (1990). doi:10.1145/98163.98167

[154] Smullyan, R.M.: Gödel's Incompleteness Theorems. Oxford University Press, Oxford, UK (1992)

[155] Sun, L., Zhan, W., Chan, C.Y., Tomizuka, M.: Behavior planning of autonomous cars with social perception. In: IEEE Intelligent Vehicles Symposium (IV). pp. 207–213 (2019). doi:10.1109/IVS.2019.8814223

[156] Thipphavong, D.P., Apaza, R., Barmore, B., Battiste, V., Burian, B., Dao, Q., Feary, M., Go, S., Goodrich, K.H., Homola, J., et al.: Urban Air Mobility airspace integration concepts and considerations. In: Aviation Technology, Integration, and Operations Conference. p. 3676 (2018). doi:10.2514/6.2018-3676

[157] Van Der Meyden, R.: Axioms for knowledge and time in distributed systems with perfect recall. In: IEEE Symposium on Logic in Computer Science. pp. 448–457 (1994). doi:10.1109/LICS.1994.316046

[158] Varela, C.A.: Programming Distributed Computing Systems. The MIT Press, Cambridge, MA (2013)

[159] Vargun, A., Musser, D.R.: Code-carrying theory. In: ACM Symposium on Applied Computing. pp. 376–383 (2008). doi:10.1145/1363686.1363780

[160] Wang, C.X., Cheng, X., Laurenson, D.I.: Vehicle-to-vehicle channel modeling and measurements: Recent advances and future challenges. IEEE Comm. Mag. **47**(11), 96–103 (2009). doi:10.1109/MCOM.2009.5307472

[161] Wang, X., Peng, Q., Li, Y.: Cooperation achieves optimal multicast capacity-delay scaling in MANET. IEEE Trans. Comm. **60**(10), 3023–3031 (2012). doi:10.1109/tcomm.2012.081512.110535

[162] Wang, Y., Zhao, Y.J.: Fundamental issues in systematic design of airborne networks for aviation. In: IEEE Aerospace Conference. p. 8 (2006). doi:10.1109/aero.2006.1655882

[163] Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS Version 3.5. In: International Conference on Automated Deduction. pp. 140–145 (2009). doi:10.1007/978-3-642-02959-2_10

[164] Wen-jie, X., Han, Y., Feng-yu, L.: The analysis of M/M/1 Queue model with N policy for damaged nodes in MANET. In: International Conference on Computer Science and Automation Engineering. vol. 1, pp. 289–294 (2011). doi:10.1109/csae.2011.5953224

[165] Yang, S., Tan, S., Xu, J.X.: Consensus based approach for economic dispatch problem in a smart grid. IEEE Trans. Power Syst. **28**(4), 4416–4426 (2013). doi:10.1109/TPWRS.2013.2271640

[166] Yin, S., Lin, X.: MALB: MANET Adaptive Load Balancing. In: IEEE Vehicular Technology Conference, 2004. vol. 4, pp. 2843–2847 (2004). doi:10.1109/vetecf.2004.1400578

[167] Yu, Y., Manolios, P., Lamport, L.: Model checking TLA$^+$ specifications. In: Advanced Research Working Conference on Correct Hardware Design and Verification Methods. pp. 54–66 (1999). doi:10.1007/3-540-48153-2_6

[168] Zheng, Z., Xie, S., Dai, H., Chen, X., Wang, H.: An overview of blockchain technology: Architecture, consensus, and future trends. In: IEEE International Congress on Big Data. pp. 557–564 (2017). doi:10.1109/BigDataCongress.2017.85

APPENDIX A
DEVELOPING MATHEMATICAL PROOFS IN ATHENA

Athena provides a framework for writing proofs and mechanically verifying their correctness. In addition to support for basic proof methods such as *modus ponens*, *proof by induction*, *case analysis*, and *proof by contradiction*, Athena provides several syntactic constructs that can aid in the intuitive development of mathematical proofs. We present a brief overview of how a mathematical proof can be developed using Athena's constructs by illustrating our Athena proof of Eq. 4.10 (Athena identifier: `THEOREM-mean-T-value`) as an example.

```
define THEOREM-mean-T-value :=
(forall t Q .
  (
    ( ((qTyp Q) = MM1)
      & ((Dist.mean (Random.pdf (numCst Q)))
        = (Function.limit t RealExt.INFY (Function.consUF N_ t)) )
      & ((Dist.mean (Random.pdf (cstArRat Q)))
        = (Function.limit t RealExt.INFY (Function.consUF L_ t)) )
      & ((Dist.mean (Random.pdf (cstDly Q)))
        = (Function.limit t RealExt.INFY (Function.consUF T_ t)) )
    )
    ==>
        (
          (Dist.mean (Random.pdf (cstDly Q)))   # T
          = ( 1.0
            /
            ( (Dist.ratePar (Random.pdf (srvcTm Q))) # mu
              -
              (Dist.ratePar (Random.pdf (cstArRat Q))) # lambda
            )
            )
        )
  )
)

conclude THEOREM-mean-T-value
pick-any t
pick-any Q
assume
    ( ((qTyp Q) = MM1)
      & ((Dist.mean (Random.pdf (numCst Q)))
        = (Function.limit t RealExt.INFY (Function.consUF N_ t)) )
      & ((Dist.mean (Random.pdf (cstArRat Q)))
        = (Function.limit t RealExt.INFY (Function.consUF L_ t)) )
      & ((Dist.mean (Random.pdf (cstDly Q)))
      = (Function.limit t RealExt.INFY (Function.consUF T_ t)) )
  )
let{
 assum :=
   ( ((qTyp Q) = MM1)
     & ((Dist.mean (Random.pdf (numCst Q)))
       = (Function.limit t RealExt.INFY (Function.consUF N_ t)) )
     & ((Dist.mean (Random.pdf (cstArRat Q)))
       = (Function.limit t RealExt.INFY (Function.consUF L_ t)) )
     & ((Dist.mean (Random.pdf (cstDly Q)))
       = (Function.limit t RealExt.INFY (Function.consUF T_ t)) )
   );
 rhs :=
   (
     (Dist.mean (Random.pdf (cstDly Q)))   # T
     = ( 1.0
       /
       ( (Dist.ratePar (Random.pdf (srvcTm Q))) # mu
         -
         (Dist.mean (Random.pdf (cstArRat Q))) # lambda
       )
       )
   );
 N_d :=  # denominator of N from Eq 3.24
   ( (Dist.ratePar (Random.pdf (srvcTm Q))) # mu
     -
     (Dist.ratePar (Random.pdf (cstArRat Q))) # lambda
   );
 Q-is-mm1 := (!left-and assum);
 N := (Dist.mean (Random.pdf (numCst Q)));
 L := (Dist.mean (Random.pdf (cstArRat Q)));
 T := (Dist.mean (Random.pdf (cstDly Q)));
 L-Not-0 := (!left-and (!mp (!uspec MM1-denominators-not-zero Q) Q-is-mm1));
 N_d-Not-0 := (!right-and (!mp (!uspec MM1-denominators-not-zero Q) Q-is-mm1));
 L-Not-0+N_d-Not-0 := (!both L-Not-0 N_d-Not-0);
 conn-2-Little := (!uspec (!uspec LITTLES-THEOREM t) Q);
 N=LT := (!mp conn-2-Little (!right-and assum)) ;
 conn-2-commu := (!uspec (!uspec RealExt.prod-commutative-axiom T) L);
 N=TL := (!chain [ N
         = (L * T) [N=LT]
         = (T * L) [conn-2-commu]
         ]);
 TL=N := (!chain [ (T * L)
         =  N [N=TL]
         ]);
 conn-2-a-x-b=c-axiom := (!uspec (!uspec (!uspec RealExt.a-x-b=c-axiom T) L) N);
 T=N-x-inv-L := (!mp (!left-iff (!left-and conn-2-a-x-b=c-axiom)) TL=N);
 conn-2-mean-numCst-axiom := (!uspec mean-numCst-axiom Q);
 N=L-by-N_d := (!mp conn-2-mean-numCst-axiom Q-is-mm1);
```

```
85  conn-2-a-by-d-x-b-by-a-axiom := (!mp (!uspec (!uspec (!uspec RealExt.
      a-by-d-x-b-by-a-axiom L) N_d) 1.0) L-Not-0+N_d-Not-0)
86}
87(!chain [ T
88    = (N * (1.0 / L)) [T=N-x-inv-L]
89    = ((L / N_d) * (1.0 / L)) [N=L-by-N_d]
90    = (1.0 / N_d) [conn-2-a-by-d-x-b-by-a-axiom]
91    ])
```

Athena provides the top-level directive `define` that allows users to assign a unique name to an Athena phrase. With respect to formal proofs, `define` can be used to specify a formal property. *E.g.*, on line 0 in the code snippet above, the `THEOREM-mean-T-value` has been specified using the `define` directive.

The `conclude` construct in Athena allows specifying a proof of the form "p follows from D", written as `conclude` p D, where p is a conclusion and D is a deduction. *E.g.*, on line 26, the `conclude` construct has been used to initiate the proof of `THEOREM-mean-T-value`, which is the desired conclusion.

The `let` keyword allows users to create and name intermediate results that can be used further ahead in a proof. *E.g.*, a `let` expression begins on line 38 in the snippet above that contains several intermediate results like `assum`, `rhs`, and `N_d`, that have been used further down in the proof of `THEOREM-mean-T-value`. Apart from naming intermediate results, `let` expressions can also contain the proofs of those results, such as the proof of `N=TL` on line 74.

Athena provides many built-in methods such as `mp` for modus ponens (line 69), `uspec` for universal specialization (line 71), `left-and`/`right-and` for conjunction elimination (line 68 and 69), `both` for conjunction production (line 70), and `chain` for equality chaining (line 87) that can be used for applying the proof methods. Among these, the `chain` method, which allows equality chaining, is particularly helpful for constructing mathematical proofs because equality chaining is a widely-used technique in mathematical computations. This enables the development of formal proofs that closely resemble their informal counterparts, making the proofs intuitive and human-readable.

APPENDIX B
USING SPASS WITH ATHENA FOR AIDING WITH LARGE PROOFS

Athena is integrated with ATPs like Vampire and SPASS that can be invoked seamlessly like primitive procedures. For our proof of eventual progress in Synod, we have divided the proof into a series of hierarchical proof obligations that are well-connected by trivial operations like *modus ponens*, *conjunction elimination*, *case analysis*, etc. However, the complexity of the statements involved makes it cumbersome and time-consuming to write the proofs for these obvious steps using the Athena proof semantics, making the task of completing the entire proof extremely time-intensive. For this reason, we use the SPASS ATP to generate the proofs of the trivial operations that connect one proof obligation to the next obligation in the proof chain. We present a discussion on our use of SPASS with Athena.

Let us consider the example of `IOE->Fair-Rcv-Causes-Theorem`, which is an intermediate theorem that states that for a nonfaulty actor, if a receive transition is enabled for a message at a time, then the message will eventually be in the set of unresponded messages in the actor's local state.

```
define Fair-Rcv-Causes-Theorem :=
(forall x m T i .
  (nonfaulty x)
    ==>
    (
          ((inMSet m (mu (config (rho T i))))
          & (ready-to (rho T i) x (receive x m)))
              ==>
                (exists j . (i N.<= j)
                      & (inMSet m (amu (als (config (rho T j)) x))))
    )

 )
```

To prove `IOE->Fair-Rcv-Causes-Theorem`, we use four facts which have been previously added to the assumption base — `S3`, which states that every natural number is either less than or equal to its successor; `transitive`, which states the transitive rule corresponding to the *less-than or equal to* relationship of natural numbers; `IOE->Fair-Rcv-Theorem`, which states that for a nonfaulty actor, if a receive transition is enabled at some time, then it will eventually occur; and `rcv-affect`, which simply states that if a receive transition happens, then the corresponding message exists in the set of unresponded messages in the recipient's new local state.

```
define S3 := (forall n . n <= S n)

define transitive := (forall x y z . x <= y & y <= z ==> x <= z)

define rcv-affect :=
(forall x m T i j .
        (= (rho T i) (then (rho T j) (receive x m)))
      ==>
            (inMSet m (amu (als (config (rho T i)) x)))
 )

define IOE->Fair-Rcv-Theorem :=
(forall x m T i .
  (nonfaulty x)
    ==>
    (
          ((inMSet m (mu (config (rho T i))))
          & (ready-to (rho T i) x (receive x m)))
              ==>
                (exists j . (i N.<= j)
                      & (= (rho T (S j)) (then (rho T j) (receive x m))) )
    )
 )
```

We can see that to obtain `IOE->Fair-Rcv-Causes-Theorem` from the four facts, it is a simple application of `rcv-affect` to `IOE->Fair-Rcv-Theorem`, to show that the message will be received and added to the local state, followed by applications of `S3` and `transi tive`. There are many ways to develop the proof interactively in Athena, but due to the complexity of the logical expression of `IOE->Fair-Rcv-Causes-Theorem`, it becomes a very convoluted task to develop the proof interactively using Athena tactics, even though the logic is straightforward. Given below is one possible interactive Athena proof:

```
conclude
(forall x m T i .
  (nonfaulty x)
    ==>
    (
          ((inMSet m (mu (config (rho T i))))
          & (ready-to (rho T i) x (receive x m)))
              ==>
                (exists j . (i N.<= j)
                      & (inMSet m (amu (als (config (rho T j)) x))))
    )
```

```

)
pick-any x:Actor
pick-any m:Message
pick-any T:TP
pick-any i:N
assume
    (nonfaulty x)
assume
    ( (inMSet m (mu (config (rho T i))))
      & (ready-to (rho T i) x (receive x m)))
let{
  IOE->FAIR-Rcv := (!uspec (!uspec (!uspec (!uspec IOE->Fair-Rcv-Theorem x) m) T) i);
  mp1 := (!mp
            IOE->FAIR-Rcv
           (nonfaulty x)
       );
  exst-j := (!mp
               mp1
         ( (inMSet m (mu (config (rho T i))))
             & (ready-to (rho T i) x (receive x m)))
        );
  i<=Si := (!uspec N.Less=.S3 i);
  trueforSj := conclude (exists z .
                          (i N.<= z)
                        & (inMSet m (amu (als (config (rho T z)) x))))
         pick-witness j for exst-j
         let{
          exst-j-sttmnt := ( (i N.<= j)
                             & (= (rho T (S j)) (then (rho T j) (receive x m))));
          Sj := (S j);
          j<=Sj := (!uspec N.Less=.S3 j);
          i<=j := (!left-and exst-j-sttmnt);
          trnstv-sttmnt := (!uspec (!uspec (!uspec N.Less=.transitive i) j) Sj);
          i<=j&j<=Sj := (!both i<=j j<=Sj);
          rcv-efctSj := (!uspec (!uspec (!uspec (!uspec (!uspec rcv-affect x) m) T) Sj)
    j);
          i<=Sj := (!mp
                trnstv-sttmnt
                i<=j&j<=Sj
                  );
          i<=Sj&Sjthen := (!both i<=Sj (!right-and exst-j-sttmnt));
          Sjinmset := conclude (inMSet m (amu (als (config (rho T (S j))) x)))
                let{
                    implies := (!chain [ (= (rho T (S j)) (then (rho T j) (receive x m))
    )
                            ==>  (inMSet m (amu (als (config (rho T (S j))) x)))  [
    rcv-efctSj]
                                ])
                }
                (!mp implies (!right-and exst-j-sttmnt));
          i<=Sj&Sjinmset := (!both i<=Sj Sjinmset)
         }
         (!egen
                (exists j . (i N.<= j)
                      & (inMSet m (amu (als (config (rho T j)) x))))
                (S j)
          )
}
(!claim trueforSj)
```

Developing proofs in a completely interactive manner gives confidence about the logical soundness of the proof steps involved. However, in significantly large projects, like our verification of eventual progress in Synod, which involve a significant number of intermediate proof obligations, the process of interactively developing the proofs of all steps, even the ones which are trivial, can make the task of verification expensive. Under such circumstances, interactive methods can be employed in conjunction with automated methods to aid in the proof development effort. Therefore, we have used SPASS to discharge the simple propositions in our proofs while the structures of the proofs have been developed manually. For this, we have decomposed the proof of each intermediate lemma into a series of hierarchical proof obligations that are well connected by trivial operations such as *modus ponens*, *conjunction elimination* and *case analysis*. SPASS can be invoked within Athena by using a statement of the form `(!prove p L)` where `p` is the statement to be proven and `L` is a list of statements already in the assumption base that SPASS must use as a premise for proving `p`. If SPASS can successfully derive `p`, then `p` is added as a theorem into the assumption base. In many cases though, SPASS may be unable to derive the conclusion from the premises passed to it, causing it to timeout and exit. If the conclusion cannot be derived from the supplied premises, SPASS informs the same instead of timing out.

In our verification of progress in Synod, we have divided complicated proof obligations into smaller obligations that are well-connected by trivial steps and then used SPASS to prove each intermediate step. Below, we show an example of this approach by using the proof of `IOE->Fair-Rcv-Causes-Theorem` which has been broken down into three obvious steps. Each step was proven by calling SPASS with the required premises.

```
define Fair-Rcv-Causes-Theorem-step1 :=
(forall x m T i .
  (nonfaulty x)
    ==>
    (
          ((inMSet m (mu (config (rho T i))))
          & (ready-to (rho T i) x (receive x m)))
              ==>
                (exists j k . (i N.<= j)
                      & (j N.<= k)
                      & (= (rho T k) (then (rho T j) (receive x m)))
                      & (= (rho T (S j)) (then (rho T j) (receive x m))))
    )
 )

(!prove Fair-Rcv-Causes-Theorem-step1 [IOE->Fair-Rcv-Theorem N.Less=.S3])

define Fair-Rcv-Causes-Theorem-step2 :=
(forall x m T i .
  (nonfaulty x)
    ==>
    (
          ((inMSet m (mu (config (rho T i))))
          & (ready-to (rho T i) x (receive x m)))
              ==>
                (exists j k . (i N.<= j)
                      & (j N.<= k)
                      & (= (rho T k) (then (rho T j) (receive x m)))
                      & (= (rho T (S j)) (then (rho T j) (receive x m)))
                      & (inMSet m (amu (als (config (rho T k)) x))))
    )
 )

(!prove Fair-Rcv-Causes-Theorem-step2 [Fair-Rcv-Causes-Theorem-step1 rcv-affect])

# Finally, proving Fair-Rcv-Causes-Theorem
(!prove Fair-Rcv-Causes-Theorem [Fair-Rcv-Causes-Theorem-step2 N.Less=.transitive])
```

Athena is *sound*, *i.e.*, if a deduction D is evaluated with respect to an assumption base Π and produces a sentence S, then Athena provides the guarantee that S is a logical consequence of Π. Athena can also detect *ill-sorted* expressions in proofs and report them, thereby removing the possibilities of ill-formed expressions in the proofs and the assumption base. However, ATPs like SPASS are like *black-boxes* whose internal operations cannot be analyzed. If there is an inconsistency in the premise passed to an ATP, then the ATP may derive `false` from the premise, enabling it to prove any desired conclusion. Therefore, one must make sure that inconsistent statements are not passed to ATPs for deriving conclusions. According to *Gödel's Incompleteness Theorems*, it is impossible to prove that a set of axioms is consistent [154]. Therefore, we have taken the following steps in order to obtain confidence in the correctness of our proofs:

- We have checked that the facts in our assumption base are not inconsistent by making sure that they conform to the informal proofs that we have developed.
- We have checked that SPASS cannot generate `false` from the statements that are in our assumption base.
- We have kept the intermediate obligations in our proof hierarchy simple enough so that they can be obtained from their preceding steps by basic operations like *modus ponens*, *conjunction elimination*, etc. In this way, we have ensured that we have a clear understanding of how the proof of a step is expected to proceed before passing the parameters to SPASS.
- Every time we have used SPASS to prove some conclusion from a premise `L`, we have first tried to prove `false` from `L` by executing the command `(!prove false L)` in Athena. In all such cases, when we have tried to prove `false` from a premise we wanted to use, SPASS has failed by finding a completion without timing out. This showed that SPASS could not have used `false` to derive the proof of the conclusion.

Given the soundness guarantees of Athena, its built-in sort-detection feature, and the steps we have taken to avoid inconsistencies in our proofs, we have established a reasonable degree of confidence in the soundness of our approach.

APPENDIX C
VERSION 0.1 OF OUR ATHENA LIBRARY FOR DISTRIBUTED ALGORITHMS

The logical hierarchy of version 0.1 of our Athena library for stochastic reasoning about distributed coordination protocols can be divided into four main levels:

- **General mathematics:**

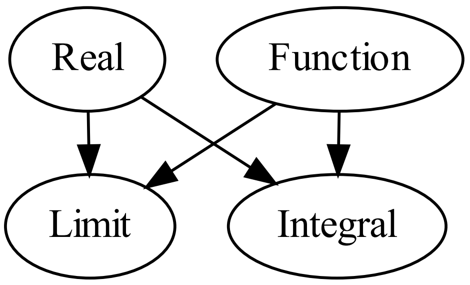


Figure C.1: The general mathematics level of the library.

At this level of the library, there are Athena constructs that can be used to express concepts of real numbers, function expressions, limits, and integrals (Fig. C.1). An example of such a construct is the `Func` datatype that has a `(consUF FunSym Real)` constructor which can be used to specify a unary function with a given function symbol and a real number.

This level is very rudimentary in version 0.1 and contains a few well-known conjectures of real numbers and does not contain any proofs or theorems.

- **Probability:**

This level of the library contains Athena constructs required to express concepts from distributions, random variables, IID random variables, probability, Erlang distributions, and exponential distributions (Fig. C.2). An example of such a construct is the

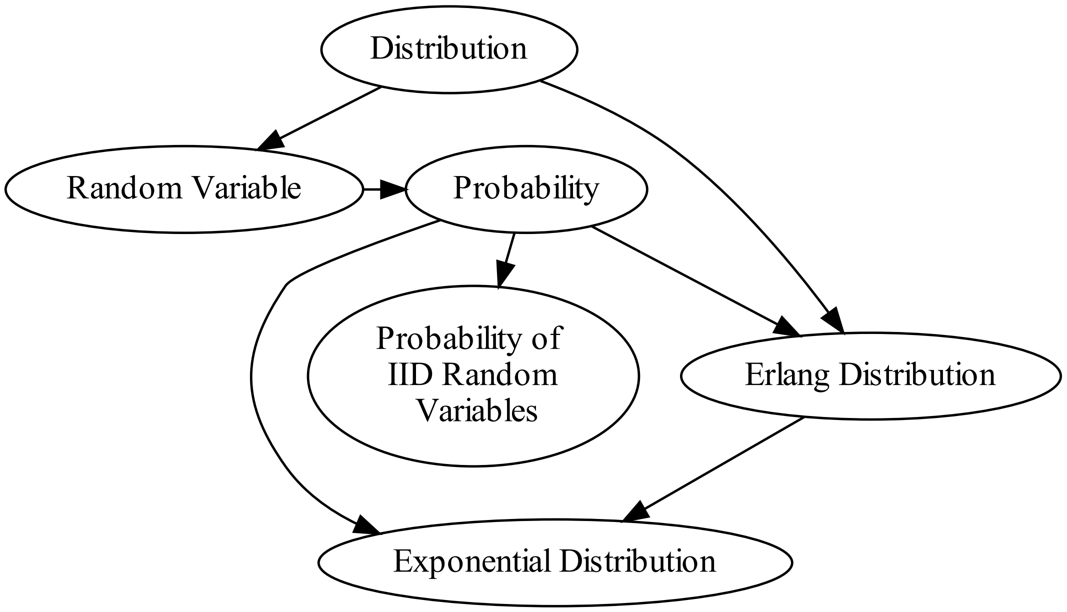


Figure C.2: The probability level of the library.

datatype `Event` that has a `(consE rel:Rel rv:Random.RandVar rl:Real)` constructor. This constructor can be used to define an event of the form "a random variable X takes a value `<symbol>` a real x" where `<symbol>` can be any relational operator in the set $\{<, \leq, >, \geq, =\}$. Probability of an event is expressed by the `probE` operator which takes an `Event` as input and returns a `Real`.

The modules in this level contain several important properties of random variables, probability, and Erlang and exponential distributions that can be used for reasoning about higher-level properties. There are proofs of important high-level properties such as Eq. C.1 and Eq. C.2[20].

$$P\left(\min\left(X_1 \ldots X_n\right) > y\right) = P\left(X_1 > y, X_2 > y, ..., X_n > y\right) \tag{C.1}$$

$$P\left(X_1 \leq x_1, X_2 \leq x_2, ..., X_n \leq x_n\right) = \prod_{i=1}^{n} P(X_i \leq x_i) \tag{C.2}$$

[20]For independent random variables.

- **Networks:**

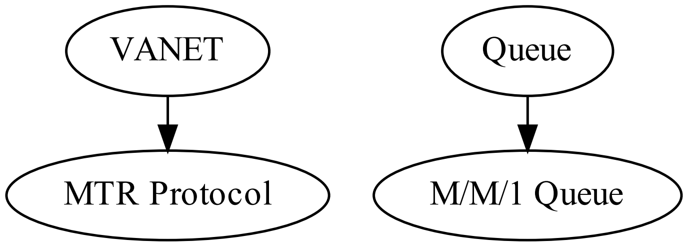


Figure C.3: The network level of the library.

This level of the library contains Athena constructs that can be used to express concepts from VANETs, the MTR protocol, queues, and the M/M/1 queue (Fig. C.3). *E.g.,* the `netTd` and the `netNodeSet` operators defined over the domain of networks `Network` return the transmission delay in a network and the set of all mobile nodes in a network respectively. Similarly, the `cstArRat` and the `cstDly` operators defined over the domain of all queues `Queue` represent the customer arrival rate and the customer processing delay of a queue respectively.

This level contains some general properties of networks and queues such as Little's theorem and also the proofs of high-level probabilistic properties regarding message transmission and processing delays from Chapter 4 such as Eq. 4.2 and Eq. 4.11.

- **Distributed systems:**

At this level of the library, there are Athena constructs to express concepts relevant to timely progress properties of distributed algorithms (Fig. C.4). Examples of such constructs are the operators `msGetTd`, `msGetTp`, and `dpTotTim` defined over the domain of distributed protocols `DisProt` that return the mesage transmission and processing delays and the total time required for progress for a protocol.

This level contains all the high-level properties of timely progress described in Chapter 4. It also contains the proofs of these properties and the derivation of the proof of timely progress for TAP from Eq. 4.12 to Eq. 4.22.

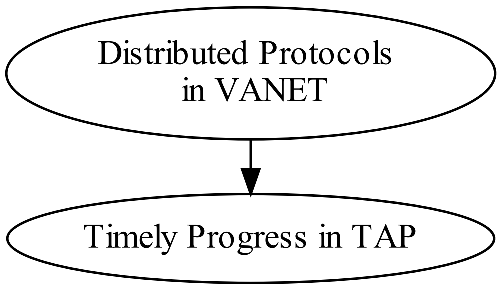


Figure C.4: The distributed system level of the library.

Overall, version 0.1 of our Athena library for probabilistic properties of distributed protocols consists of 71 axioms and 27 theorems that are spread across the different levels described above.

APPENDIX D
PERMISSIONS

Permissions for Chapter 1

The following files contain license details and terms and conditions for the reproduction of materials used in Chapter 1 of the dissertation:

- - **File Name:** EPTCS_FMAS21.pdf
  - **File Type:** Portable Document Format (PDF)
  - **File Size:** 60 kb
  - **Required Software:** Any standard PDF viewer
  - **Special Harware Requirements:** None
- - **File Name:** SPRINGER_NFM21.pdf
  - **File Type:** Portable Document Format (PDF)
  - **File Size:** 197 kb
  - **Required Software:** Any standard PDF viewer
  - **Special Harware Requirements:** None
- - **File Name:** SPRINGER_DDDAS20.pdf
  - **File Type:** Portable Document Format (PDF)
  - **File Size:** 197 kb
  - **Required Software:** Any standard PDF viewer
  - **Special Harware Requirements:** None

Permissions for Chapter 2

The following files contain license details and terms and conditions for the reproduction of materials used in Chapter 2 of the dissertation:

- - **File Name:** SPRINGER_NFM21.pdf
  - **File Type:** Portable Document Format (PDF)

  - **File Size:** 197 kb
  - **Required Software:** Any standard PDF viewer
  - **Special Harware Requirements:** None

Permissions for Chapter 4

The following files contain license details and terms and conditions for the reproduction of materials used in Chapter 4 of the dissertation:

- - **File Name:** EPTCS_FMAS21.pdf
  - **File Type:** Portable Document Format (PDF)
  - **File Size:** 60 kb
  - **Required Software:** Any standard PDF viewer
  - **Special Harware Requirements:** None

Permissions for Chapter 5

The following files contain license details and terms and conditions for the reproduction of materials used in Chapter 5 of the dissertation:

- - **File Name:** SPRINGER_DDDAS20.pdf
  - **File Type:** Portable Document Format (PDF)
  - **File Size:** 197 kb
  - **Required Software:** Any standard PDF viewer
  - **Special Harware Requirements:** None