Innovations in Systems and Software Engineering
https://doi.org/10.1007/s11334-022-00463-5

S.I. : SELECTED EXTENDED PAPERS OF NFM 2021

Eventual consensus in Synod: verification using a failure-aware actor model

Saswata Paul[1] · **Gul Agha**[2] · **Stacy Patterson**[1] · **Carlos Varela**[1]

Received: 11 November 2021 / Accepted: 27 June 2022
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2022

Abstract
Successfully attaining consensus in the absence of a centralized coordinator is a fundamental problem in distributed multi-agent systems. We analyze progress in the Synod consensus protocol—which does not assume a unique leader—under the assumptions of asynchronous communication and potential agent failures. We identify a set of sufficient conditions under which it is possible to guarantee that a set of agents will eventually attain consensus using Synod. First, a subset of the agents must not permanently fail or exhibit Byzantine failure until consensus is reached, and second, at least one proposal must be eventually uninterrupted by higher-numbered proposals. To formally reason about agent failures, we introduce a failure-aware actor model (FAM). Using FAM, we model the identified conditions and provide a formal proof of eventual progress in Synod. Our proof has been mechanically verified using the Athena proof assistant and, to the best of our knowledge, it is the first machine-checked proof of eventual progress in Synod.

Keywords Paxos · Liveness · Fairness · Athena · Urban air mobility · Formal theorem proving

1 Introduction

Consensus, which requires a set of agents to reach an agreement, is a fundamental problem in distributed systems. Under *asynchronous* communication settings, where message transmission and processing delays are unbounded, it is impossible to guarantee consensus [1] since message delays cannot be differentiated from agent failures. Nevertheless, in *decentralized multi-agent* systems, where there is no centralized *coordinator* to manage safe operations, it is necessary for the agents to use *consensus protocols* for coordination.

An important application of multi-agent systems is in the aerospace domain where the use of *uncrewed aircraft systems* (UAS) for *Urban Air Mobility* (UAM) [2] operations will significantly increase the density of aircraft in the *National Airspace System* (NAS) [3]. Since human-operated approaches of *air-traffic management* (ATM) are not scalable to high densities and are prone to human errors [4], UAS must be capable of autonomous *UAS traffic management* (UTM) [5]. In [6], we proposed an UTM technique called *decentralized admission control* (DAC) in which aircraft coordinate over an asynchronous network called the *Internet of Planes* (IoP) [7]. In DAC, a *candidate* aircraft computes a *conflict-aware flight plan* [8] with respect to a set of *owner* aircraft of a *controlled airspace*. The candidate then requests admission into the airspace by proposing this plan to the owners. There may be multiple candidates simultaneously proposing plans that are mutually incompatible. Therefore, the owners can only admit the candidates sequentially by agreeing on one proposal at a time. Since UAM applications are time-sensitive, any consensus protocol used in DAC must guarantee an eventual agreement.

In this paper, we present an analysis of consensus that is primarily motivated by the requirements of DAC. We study the *Synod consensus protocol* [9] that does not

✉ Saswata Paul
pauls4@rpi.edu

Gul Agha
agha@illinois.edu

Stacy Patterson
sep@cs.rpi.edu

Carlos Varela
cvarela@cs.rpi.edu

[1] Department of Computer Science, Rensselaer Polytechnic Institute, 110 Eighth Street, Troy, NY 12180, USA

[2] Department of Computer Science, University of Illinois Urbana-Champaign, 506 S. Wright Street, Urbana, IL 61801, USA

Published online: 25 July 2022

restrict proposal rights to a *unique leader*, but guarantees the *safety* property that *"only one value will be agreed upon"*. To guarantee the *progress* property that, *"eventually, an agreement will happen"*, the *Paxos consensus protocol* [10], a better-known variant of Synod, allows only a unique leader to initiate proposals. For DAC, the leaderless Synod consensus protocol is more appropriate as it allows multiple candidates to propose values to the owners.

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) [1] implies that unique leadership cannot be guaranteed in asynchronous systems where agents may unpredictably fail. In some cases, network partitioning may also erroneously cause multiple leaders to be elected [11]. Third, vehicular 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*, where reconfiguration is expected to be infrequent [12], 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 paper, we focus on identifying sufficient conditions under which the fundamental Synod protocol can make eventual progress.

The failure of *safety-critical* aerospace systems can lead to the loss of human life and property (*e.g.*—the fatal crash of Air France 447 in 2009 [13,14]). To ensure correctness, formal methods can be used for the rigorous verification of such systems. The *actor model* [15,16] is a theoretical model of concurrent computation that can be used for formally reasoning about distributed systems. It assumes asynchronous communication and *fairness*, which is useful for reasoning about the progress of such systems. In airborne communication networks like the IoP, aircraft may experience temporary or permanent communication failures where they are unable to send or receive any messages. Such failures can affect progress in distributed algorithms like Synod. Therefore, to support reasoning about such communication failures, we introduce a *failure-aware actor model* (FAM) that assumes *predicate fairness* [17]. We use FAM to reason about progress in Synod under a set of purely asynchronous conditions and develop a formal proof of eventual progress. We then adopt a primarily interactive approach for mechanizing the proof using the *Athena proof assistant* [18] while using the SPASS theorem prover [19] to prove the simple steps[1].

The main contributions of this work are:

- We identify a set of conditions under which eventual progress in Synod can be guaranteed in purely asynchronous settings, without assuming a unique leader.
- We introduce a failure-aware actor model (FAM) to support formal reasoning about temporary or permanent communication failures in the IoP.
- We prove that eventual progress can be guaranteed in Synod under the identified conditions and mechanically verify our proof using Athena. To our knowledge, this is the first machine-checked proof of progress for Synod.
- We argue that in a predicate fair implementation of FAM, our conditions for progress in Synod can be guaranteed and provide a mechanically verified proof for the same[2].

The paper is structured as follows—Sect. 2 describes the Synod protocol and presents the conditions for eventual progress in Synod; Sect. 3 introduces FAM and its fairness properties; Sect. 4 presents a formal proof of eventual progress in Synod and a proof of the conditions under predicate fairness; Sect. 5 discusses the mechanization of our formal proofs in Athena; Sect. 6 discusses prior research related to our study of consensus and the actor model; Sect. 7 presents a discussion on our approach along with some potential future directions of work; and Sect. 8 concludes the paper with a summary.

2 The Synod protocol

Synod assumes an asynchronous, non-Byzantine system model in which agents operate at arbitrary speed, may fail and restart, and have stable storage. Messages can be duplicated, lost, and have arbitrary transmission time, but cannot be corrupted [10]. It 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

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

[2] This proof improves upon an earlier version of the work [20]

property of Synod requires that *any two quorums must intersect*[3]. A simple example of a quorum is a subset of the set of all acceptors which constitutes a simple majority. [21] 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*.
- *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* [10]:

- **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 [9].

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.

2.1 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*" [10]. 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 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 [10] (*livelock*).
- Progress may be affected even if one random agent fails unpredictably [1].

Paxos assumes that a distinguished proposer (leader) is elected as the only proposer that can initiate proposals [9]. Lamport states *"If the distinguished proposer can communicate successfully with a majority of acceptors, and if it*

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

uses a proposal with number greater than any already used, then it will succeed in issuing a proposal that is accepted." [10]. The FLP impossibility result [1] 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 [11]. 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.

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 [22] 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 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 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.

3 The failure-aware actor model (FAM)

Actors are self-contained, concurrently interacting entities of a computing system that communicate by message passing [23]. We use the actor model [16] to formally reason about eventual consensus in Synod since it assumes asynchronous communication and fairness, which is helpful for reasoning about progress in asynchronous networks like the Internet of Planes (IoP). Fairness in the actor model has the following consequences [24]:

- *Guaranteed message delivery*, and
- *An actor infinitely often ready to process a message will eventually process the message*.

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 reason about such actor failures, we introduce a failure-aware actor model (FAM).

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* [17], which states that a

Fig. 1 Operational semantics for the base-level transition rules

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

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

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

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

predicate that is infinitely often enabled in a given path will eventually be satisfied.

3.1 The semantics of FAM

Varela [24] presented a dialect (AM) of Agha, Mason, Smith, and Talcott's *lambda-calculus*-based actor language (AMST) [25] whose operational semantics are a set of *labeled transitions* from *actor configurations* to actor configurations[4]. 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, denoted 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 labeled 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 (AM) 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 pre-failure state. 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. Figures 1 and 2 show the operational semantics of our actor language as labeled transition rules where:

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

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

Fig. 2 Operational semantics for the meta-level transition rules

- $\rightarrow_\lambda$ 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 multi-set union,
- $\alpha|_S$ denotes restriction of mapping α to elements in set S, and
- $dom(\alpha)$ is the domain of α.

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:

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

where $fv(e)$ is the set of free variables in the expression e.

3.2 Fairness in FAM

FAM assumes a general fairness property called *predicate fairness* [17] which states that *"if a predicate is infinitely often enabled in a given path, then, eventually 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,

[4] Interested readers can refer to Sect. 4.5 of [24] for more details about the language.

Table 1 Set symbols for our formal specification

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

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 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 Sect. 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 enabled infinitely often in any path. Hence, by predicate fairness, our progress conditions will eventually be true, allowing some proposal number to eventually get accepted.

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.

Interested readers may refer to Appendix 1 for some important results about the relationship of FAM with Varela's dialect of AMST (AM).

4 Formal proof of eventual progress in Synod

In this section, we formally prove two important properties regarding eventual progress in Synod by modeling the system using FAM. The logical notations used in this section have been introduced in Tables 1 and 2.

The first property that we prove is eventual progress in Synod under the conditions (CND) identified in Sect. 2.2. To prove this property, it suffices to show that, eventually, some proposal number will receive votes from some quorum and be chosen. Theorem 1 represents this eventual progress property formally by stating that, eventually, some proposer will learn that some proposal number has been chosen.

Theorem 1 *(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.*

$$\texttt{Theorem-1} \equiv$$
$$\texttt{CND} \implies (\forall T \in \mathcal{T} : (\exists i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B} : \text{Ł}(p, b, \varsigma(T, i))))$$

By CND, eventually, some nonfaulty proposer p will become ready to propose a proposal number b, that will satisfy *P1* and *P2*, to a nonfaulty quorum Q. Our proof strategy for Theorem 1 is to show that under the fairness assumptions of FAM, the behavior of Synod actors, and some assumptions on the properties of nonfaulty actors, eventually, p will learn that b has been chosen by Q.

The next important property that we prove in this section shows that, theoretically, under the predicate fairness assumption of FAM, CND can be eventually guaranteed. For that, we specify a Synod-specific predicate ψ over actor configurations such that $\psi(\kappa)$ is `true` if and only if CND holds at configuration κ. Theorem 2 formally states the required property that, eventually, ψ will be satisfied in all fair transition paths.

Table 2 Relation symbols for our formal specification

Symbol	Description	Input	Output
ς	Get actor configuration at an indexed point in a transition path	$\mathcal{T} \times \mathbb{N}$	$\mathcal{C}$
ρ	Get indexed point in a transition path	$\mathcal{T} \times \mathbb{N}$	$\mathcal{T}$
$\top$	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`
δ	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`

Theorem 2 *(CND follows from predicate fairness) In all fair transition paths, eventually, the predicate ψ will be* `true` *at some configuration.*

$$\texttt{Theorem-2} \equiv \forall T \in \mathcal{T} : \exists \kappa_i^T \in \mathcal{C} : \psi(\kappa_i^T)$$

We prove Theorem 2 by first defining when a predicate over configurations is enabled and then arguing that under suitable conditions, predicate fairness can cause ψ to be eventually `true`.

In the rest of this section, we first formally specify some necessary fairness properties of FAM, the behavior of Synod actors, the properties of nonfaulty actors, and the conditions required for eventual progress. Then we provide the formal proofs of Theorems 1 and 2.

Given below are some important concepts useful for the formal specifications and proofs:

- A message is represented by a tuple $\langle$ `s` $\in\mathcal{A}$, `r` $\in\mathcal{A}$, `k` $\in\xi$, `b` $\in\mathcal{B}$, `v` $\in\mathcal{V}\rangle$ where `s` is the sender, `r` is the receiver, `k` is the type of message, `b` is a proposal number, `v` is a value, and $\xi = \{1a, 1b, 2a, 2b\}$. $\bar{v} \in \mathcal{V}$ is a null value constant used in $1a$ messages.
- The local state of an actor x can be extracted from a configuration κ as a tuple $\langle \eta_\kappa^x \in \mathbb{M}, \beta_\kappa^x \in \mathcal{B}, v_\kappa^x \in \mathcal{V}\rangle$ where η_κ^x is the set of messages received but not yet responded to, β_κ^x is the highest proposal number seen by an acceptor actor, and v_κ^x is the value corresponding to the highest proposal number accepted by an acceptor actor.
- *Transition paths* represent the dynamic changes to actor configurations as a result of transition steps [26] 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_i^T represents the actor configuration at an indexed point i in a path T.
- $\mathbb{P} \subset \mathcal{A}$ and $\mathbb{A} \subset \mathcal{A}$ are the sets of proposers and acceptors, respectively, and a quorum is a possibly equal, fixed, non-empty subset of $\mathbb{A}$, *i.e.*, $\forall Q \in \mathcal{Q} : Q \subseteq \mathbb{A} \wedge Q \neq \emptyset$.

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.

$$\begin{aligned}
&\texttt{F-Snd-Axm} \equiv \\
&\forall x \in \mathcal{A}, m \in \mathcal{M}, T \in \mathcal{T}, i \in \mathbb{N} : (\Re(\rho(T,i), x, \mathfrak{s}(x,m)) \implies \\
&\quad ((\exists j \in \mathbb{N} : (j \geq i) \\
&\qquad \wedge \ \rho(T, j+1) = \top(\rho(T,j), \mathfrak{s}(x,m))) \\
&\qquad \vee \ (\exists k \in \mathbb{N} : (k > i) \\
&\quad \wedge \ (\forall j \in \mathbb{N} : (j \geq k) \implies \neg\Re(\rho(T,j), x, \mathfrak{s}(x,m))))))
\end{aligned}$$

$$
\begin{aligned}
&\texttt{F-Rcv-Axm} \;\equiv\\
&\forall x \in \mathcal{A}, m \in \mathcal{M}, T \in \mathcal{T}, i \in \mathbb{N} :\\
&((m \in \mathfrak{m}(\varsigma(T,i)) \;\wedge\; \Re(\rho(T,i), x, \mathfrak{r}(x,m))) \implies\\
&\quad ((\exists j \in \mathbb{N} : (j \geq i)\\
&\qquad \wedge\; \rho(T, j+1) = \top(\rho(T,j), \mathfrak{r}(x,m)))\\
&\qquad \vee\; (\exists k \in \mathbb{N} : (k > i)\\
&\qquad\quad \wedge\; (\forall j \in \mathbb{N} : (j \geq k) \implies\\
&\qquad \neg(m \in \mathfrak{m}(\varsigma(T,j)) \;\wedge\; \Re(\rho(T,j), x, \mathfrak{r}(x,m)))))))
\end{aligned}
$$

4.2 Rules specifying the actions of Synod actors

The Synod protocol is presented in [9] as a high-level abstraction of the behavior of the agents while leaving out the implementation details to the discretion of the system developers [27]. We specify rules over actor local states that dictate if an available Synod actor should become ready to send a $1b$, $2a$, or $2b$ message. Since Synod does not specify *when* a proposer should send a $1a$ message, we leave that behavior unspecified.

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

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

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

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.

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

4.3 Assumptions about the future behavior of nonfaulty 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 [22]. 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 $\mathring{d}$ to specify an actor as nonfaulty, such that:

- $\mathring{d}(x)$ implies that x will be eventually available if there is a message "en route" that x needs to receive.
- (x) implies that x will be eventually available if x's local state dictates that it needs to send a message.
- (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.

$$
\begin{aligned}
&\texttt{Prp-NF-Axm} \;\equiv\\
&\forall p \in \mathbb{P} : \mathring{d}(p) \implies\\
&\quad (\forall b \in \mathcal{B}, k \in \xi, v \in \mathcal{V}, T \in \mathcal{T}, i \in \mathbb{N}, a \in \mathbb{A}, Q \in \mathcal{Q} :\\
&\qquad (\phi(p, b, Q, \varsigma(T,i))\\
&\qquad \vee\; \langle a, p, k, b, v\rangle \in \mathfrak{m}(\varsigma(T,i))) \implies\\
&\qquad\qquad (\mathfrak{a}(\varsigma(T,i), p)\\
&\qquad\qquad \vee\; (\exists j \in \mathbb{N} : (j > i) \;\wedge\; \mathfrak{a}(\varsigma(T,j), p))))
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{Acc-NF-Axm} \;\equiv\\
&\forall a \in \mathbb{A} : \mathring{d}(a) \implies\\
&\quad (\forall p \in \mathbb{P}, k \in \xi, v \in \mathcal{V}, T \in \mathcal{T}, i \in \mathbb{N}, b \in \mathcal{B} :\\
&\qquad ((\langle p, a, 1a, b, v\rangle \in \eta^a_{\varsigma(T,i)} \;\wedge\; (\beta^a_{\varsigma(T,i)} < b))\\
&\qquad \vee\; (\langle p, a, 2a, b, v\rangle \in \eta^a_{\varsigma(T,i)} \;\wedge\; (\beta^a_{\varsigma(T,i)} \leq b))\\
&\qquad \vee\; (\langle p, a, k, b, v\rangle \in \mathfrak{m}(\varsigma(T,i)))) \implies\\
&\qquad\qquad (\mathfrak{a}(\varsigma(T,i), a)\\
&\qquad\qquad \vee\; (\exists j \in \mathbb{N} : (j > i) \;\wedge\; \mathfrak{a}(\varsigma(T,j), a))))
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{NF-IOE-Axm} \;\equiv\\
&\forall x \in \mathcal{A} : \mathring{d}(x) \implies\\
&\quad (\forall T \in \mathcal{T}, i \in \mathbb{N}, m \in \mathcal{M} :\\
&\qquad ((m \in \mathfrak{m}(\varsigma(T,i)) \;\wedge\; \Re(\rho(T,i), x, \mathfrak{r}(x,m))) \implies\\
&\qquad ((\exists j \in \mathbb{N} : (j \geq i)\\
&\qquad\qquad \wedge\; \rho(T, j+1) = \top(\rho(T,j), \mathfrak{r}(x,m)))\\
&\qquad\quad \vee\; (\forall k \in \mathbb{N} : (k > i) \implies\\
&\qquad\qquad (\exists j \in \mathbb{N} : (j \geq k)\\
&\qquad\qquad\quad \wedge\; (m \in \mathfrak{m}(\varsigma(T,j)) \;\wedge\; \Re(\rho(T,j), x, \mathfrak{r}(x,m)))))))\\
&\qquad \wedge\; (\Re(\rho(T,i), x, \mathfrak{s}(x,m)) \implies\\
&\qquad\qquad ((\exists j \in \mathbb{N} : (j \geq i)\\
&\qquad\qquad\qquad \wedge\; \rho(T, j+1) = \top(\rho(T,j), \mathfrak{s}(x,m)))\\
&\qquad\qquad \vee\; (\forall k \in \mathbb{N} : (k > i) \implies\\
&\qquad\qquad\quad (\exists j \in \mathbb{N} : (j \geq k) \;\wedge\; \Re(\rho(T,j), x, \mathfrak{s}(x,m))))))
\end{aligned}
$$

We then introduce a predicate δ that is `true` for a proposal number if and only if it satisfies the conditions P1 and P2 described in Sect. 2.2.

$$
\begin{aligned}
&\texttt{P1-P2-Def} \;\equiv\\
&\forall b \in \mathcal{B} : \delta(b) \iff\\
&\quad (\forall p \in \mathbb{P}, T \in \mathcal{T}, i \in \mathbb{N}, v \in \mathcal{V}, a \in \mathbb{A} :\\
&\qquad ((\langle p, a, 1a, b, \bar{v}\rangle \in \eta^a_{\varsigma(T,i)} \implies \beta^a_{\varsigma(T,i)} < b)
\end{aligned}
$$

$$\wedge\ (\langle p, a, 2a, b, v\rangle \in \eta^{a}_{\varsigma(T,i)} \implies \beta^{a}_{\varsigma(T,i)} \leq b)))$$

Finally, our conditions for formally guaranteeing progress state – *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.*

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

4.4 The proof of progress

Theorem 1 (presented earlier) represents eventual progress in Synod, and Lemmas 1 and 2, which represent progress in Phase 1 and Phase 2, respectively, are useful for proving Theorem 1.

Lemma 1 *(Phase 1 progress) 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.*

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

Lemma 2 *(Phase 2 progress) 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.*

$$\begin{aligned}&\texttt{Lemma-2} \equiv\\ &\forall T \in \mathcal{T}, i \in \mathbb{N}, p \in \mathbb{P}, b \in \mathcal{B}, Q \in \mathcal{Q} :\\ &(đ(p) \ \wedge\ \delta(b) \ \wedge\ \phi(p, b, Q, \varsigma(T, i)) \ \wedge\ (\forall a \in Q : đ(a)))\\ &\qquad\implies (\exists j \in \mathbb{N} : (j \geq i) \ \wedge\ \Phi(p, b, Q, \varsigma(T, j)))\end{aligned}$$

Given below are the proof sketches of Theorem 1 and Lemmas 1 and 2:

Theorem 1 *Proof -*

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

Lemma 1 *Proof -*

(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`, and `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. □

Lemma 2 *Proof -*

(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`, and `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. □

4.5 The proof of CND under predicate fairness

In order to formalize the specification of predicate fairness in FAM, we first define what it means for a predicate P over 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[5].

$$\begin{aligned}&\texttt{Enabled-Def} \equiv\\ &\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C}, P \in \mathcal{P} : e(\kappa_i^T, P) \iff\end{aligned}$$

[5] The symbols $e(\kappa_i^T, P)$, $\lambda(\kappa_i^T, I)$, and $[\kappa_i^T \xrightarrow{I} \kappa_j^T]$ 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.

$$(\exists I \in \mathcal{I} : \lambda(\kappa_i^T, I) \implies (\exists \kappa_j^T \in \mathcal{C} : [\kappa_i^T \xrightarrow{I} \kappa_j^T] \wedge P(\kappa_j^T)))$$

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

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

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.

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

As previously mentioned in Sect. 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 highest proposal number yet proposed to a nonfaulty quorum[6], 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.

$$\begin{aligned}&\texttt{Prp-Retry} \equiv\\ &\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C} : (\exists p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_i^T, p, Q)) \implies\\ &\quad ((\exists \kappa_j^T \in \mathcal{C} : (j \geq i) \wedge \psi(\kappa_j^T))\\ &\quad \vee(\forall \kappa_k^T \in \mathcal{C} : (k > i) \implies\\ &\quad (\exists \kappa_j^T \in \mathcal{C} : (j \geq k) \wedge (\exists p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_j^T, p, Q)))))\end{aligned}$$

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`).

$$\begin{aligned}&\texttt{Wit-Seq} \equiv\\ &\forall T \in \mathcal{T}, \kappa_i^T \in \mathcal{C} :\\ &(\exists p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_i^T, p, Q)) \implies\\ &\qquad (\lambda(\kappa_i^T, \hat{I}) \implies (\exists \kappa_j^T \in \mathcal{C} : [\kappa_i^T \xrightarrow{\hat{I}} \kappa_j^T] \wedge \psi(\kappa_j^T)))\end{aligned}$$

Finally, for consensus to happen, 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 proposal yet.

$$\begin{aligned}&\texttt{Some-Prp-Ready} \equiv\\ &\forall T \in \mathcal{T} : (\exists \kappa_i^T \in \mathcal{C}, p \in \mathbb{P}, Q \in \mathcal{Q} : \varphi(\kappa_i^T, p, Q))\end{aligned}$$

Theorem 2 states that, eventually, the predicate ψ will be satisfied in all fair paths, which is, by definition, equivalent to the conditions `CND`.

Theorem 2 *Proof* -

(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`. □

5 Verification using Athena

Distributed systems have previously been verified using both *interactive theorem provers*, such as Coq [28] (*e.g.*—[29]), and *automated theorem provers* (ATP), such as Z3 [30] (*e.g.*—[31]). Automation can be helpful for proof development since ATPs can search for proofs without manual human interaction. However, our goal was to make our Athena proofs human-comprehensible in the form of clear and intuitive series of logical conclusions. Such detailed proofs can make it easy to identify the subtle ways in which the assumptions can affect the proofs, thereby facilitating future proof development under different assumptions. Interactive proof development can achieve this by requiring proofs to be conveyed as a detailed sequence of logical inference steps for mechanical verification. This is not possible when ATPs are used as black-boxes to automatically search for non-trivial proofs. Additionally, since the interactive approach requires all proof steps to be specified explicitly, intermediate results can be reused in appropriate contexts. Therefore, we adopted a primarily interactive approach of proof development aided by minimal automation.

[6] 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.

We have mechanically verified all the lemmas and theorems introduced in Sect. 4 using Athena. Although it might be possible to express our proofs in other interactive systems like Coq, our choice of Athena for this purpose has been inspired by the existing Athena formalizations of actor systems in the literature [26,32,33]. Athena is based on *many-sorted first order logic* [34] and uses a *natural deduction* [35] style of proofs, which is an intuitive way of reasoning. Logical *sentences* can either be asserted into an *assumption base* or proven as a theorem, and there is a *soundness guarantee* that any proven theorem will be a logical consequence of sentences in the assumption base. Athena performs automatic *sort-checking* to prevent *ill-sorted* expressions in specifications. It allows theorems to be introduced at an abstract level by encapsulating proofs in parameterized *methods* which can be instantiated to prove different specializations of the abstract theorems (*e.g.*—for any arbitrary p and q, the `cond-def` method can prove $\neg p \vee q$ from $p \Longrightarrow q$). Athena has an interactive proof development environment that is also integrated with ATPs like Vampire [36] and SPASS [19]. This allows the high-level skeleton of proofs to be developed interactively, while the basic inference steps can be easily delegated to ATPs via Athena commands.

We have developed custom constructs in Athena's many-sorted first order logic to specify the formal expressions presented in Sect. 4. *E.g.,* the expression $\rho(T, i)$ is represented as `(rho T i)` in our Athena formalization and a message $\langle s, r, k, b, v\rangle$ can be represented as `(consM s r k b v)` using the constructor `consM` for forming messages. Using these constructs, sentences like `F-Snd-Axm` (Fig. 3), `Rcv-Rul` (Fig. 4), and Theorem 1 (Fig. 5) can be specified in Athena using the `define` keyword, and then they can be either asserted into the assumption base or proven as theorems. We have asserted the fairness axioms, the rules defining the behavior of Synod actors, and the assumptions for progress as sentences in Athena. The proofs of the lemmas and theorems have then been mechanically verified to add the corresponding sentences to the assumption base. Proofs of generic properties, like `IOE->Fair-Snd-Theorem` (Fig. 6) (which states that an enabled **snd** transition for a nonfaulty actor will eventually happen), were developed once and then reused in multiple contexts.

Our guided proofs involved a significant number of intermediate steps that were connected by simple inference logic such as *modus ponens* and *conjunction elimination*. However, often the expressions comprised nested layers of conjunctions and disjunctions, which made the task of interactively guiding the proofs of these simple steps using Athena's proof tactics very time consuming (find an example in [37]). Therefore, we used SPASS to discharge the simple inference rules between such steps, while the logical structures of the proofs were developed manually. SPASS can be invoked within Athena by using a command of the form `(!prove p L)`

```
define F-Snd-Axm :=
(forall x m T i .
  (ready-to (rho T i) x (send x m))
    ==>
    ((exists j . (i N.<= j)
          & (= (rho T (S j)) (then (rho T j) (
     send x m))))
     (exists k . (i N.< k)
          & (forall j .
              (k N.<= j)
                ==> (~(ready-to (rho T j) x (
     send x m)))))))
```

Fig. 3 The Athena code for `F-Snd-Axm`

```
define Rcv-Rul :=
(forall T i sender recipient b typ v .
  ((inMSet (consM sender recipient typ b v) (mu
     (config (rho T i))))
  &(available (rho T i) recipient))
    ==>
      (ready-to (rho T i) recipient (receive
     recipient (consM sender recipient typ b v))
     ))
```

Fig. 4 The Athena code for `Rcv-Rul`

```
define Progress-Theorem :=
(CND
  ==>
  (forall T . (exists p i b .
       (learn p (config (rho T i)) b))))
```

Fig. 5 The Athena code for Theorem 1

```
define IOE->Fair-Snd-Theorem :=
(forall x m T i .
    (nonfaulty x)
        ==>
            (((ready-to (rho T i) x (send x m)))
                ==>
                    (exists j . (i N.<= j)
                    & (= (rho T (S j)) (then (
     rho T j) (send x m))))))
```

Fig. 6 `IOE->Fair-Snd-Theorem` in Athena

where `p` is the sentence to be proven and `L` is a list of sentences already in the assumption base. If SPASS can successfully derive `p` from `L`, then `p` is added to the assumption base as a theorem.

Using SPASS with Athena allowed us to retain the clarity provided by interactive proofs without being impeded by the drawn-out task of manually guiding the simple inference steps. As a consequence, the structure of our proofs allows us to clearly identify the subtle effects of modifying the various assumptions, at a more detailed level, than what would have been possible if SPASS was used to discharge non-trivial proof steps. *E.g.,* a failure of the proof of Theorem 1, caused by changing $j \geq i$ to $j > i$ in `F-Snd-Axm`, can be easily traced back to a basic *modus ponens* step in the proof of `IOE->Fair-Snd-Theorem`. If the complete proof of `IOE->Fair-Snd-Theorem` was

delegated to SPASS, then SPASS would have simply failed without reporting the exact cause of the failure. In our experience, such proof clarity can not only help in debugging incorrect proofs but can also promote an in-depth understanding of the system properties that are being verified.

ATPs like SPASS are like black-boxes whose internal operations cannot be easily 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 sentences are not passed to ATPs for deriving conclusions. For this reason, we have taken the following steps to obtain reasonable confidence in the proofs that we have developed:

- We have ensured that our Athena code accurately conforms to our formal proofs.
- We have only used SPASS to discharge simple steps such as *modus ponens* and *conjunction elimination*, thereby ensuring that we have a clear idea of how the proof of a step delegated to SPASS is expected to proceed.
- Before using any premise `L` to prove a step using SPASS, we have first checked that `(!prove false L)` fails in SPASS without timing out. This showed that SPASS could not derive `false` from the premise.

Using Athena, we have mechanically verified the proofs of Lemmas 1, 2, Theorems 1, and 2. The proofs consist of over 7200 lines of Athena code. However, the safety property of Synod has not been verified in this work.

6 Related work

Previously, De Prisco et al. [38] presented a rigorous handwritten proof of safety for Paxos along with an analysis of time performance and fault tolerance. Chand et al. [39] provided a proof of safety of *Multi-Paxos* using TLA$^+$ [40] and TLAPS [41]. Padon et al. [42] have verified safety of Paxos, *Vertical Paxos* [12], *Fast Paxos* [22], and *Stoppable Paxos* [43]. Küfner et al. [44] provided a machine-checkable proof for the safety property of Paxos. Schiper et al. [45] formally verified the safety property of a Paxos-based totally ordered broadcast protocol using EventML [46] and the Nuprl [47] proof assistant. Howard et al. [21] have model-checked the safety property of *Flexible Paxos* in the TLC model checker [48]. Rahli et al. [49,50] verified safety of a Multi-Paxos implementation using EventML and Nuprl. In contrast to our work, none of the above work has verified any progress property for leaderless consensus.

While we have verified eventual consensus in the absence of a unique leader, previously, progress in consensus has been studied under the assumption of a leader. A unique leader prevents livelock by design as there cannot be interrupting proposals. McMillan et al. [31] machine-checked the safety and eventual progress properties of Stoppable Paxos [43], a variant of Paxos where there is a unique leader, using Ivy [51]. Drăgoi et al. [52] verified eventual progress in *LastVoting* [53], an adaptation of Paxos in the *Heard-Of (HO) model* [53]. Eventual progress in LastVoting relies on the assumption of a unique leader and has also been verified in [54] using Isabelle [55].

In contrast to our purely asynchronous conditions for the analysis of eventual progress, progress of algorithms has also been investigated under the assumption of varying degrees of synchrony. Hawblitzel et al. [56,57] verified eventual progress in IronRSL, a Multi-Paxos implementation, using a framework called IronFleet. In addition to a unique leader, the eventual progress property of IronRSL assumes that message delays will eventually follow a known bound (*partial synchrony* [58]). Similarly, Losa & Dodds [59] presented the *Stellar Consensus Protocol* and verified its eventual progress using partial synchrony. Partial synchrony can not only allow the detection of agent failures for the execution of algorithms, but can also allow for strategic timeouts. *E.g.,* Wanner et al. verified the eventual consistency of a round-based log replication protocol in which nodes use timeouts to detect the end of rounds and in each instance, only a "primary" node (leader) is allowed to initiate proposals. Attiya et al. [60] and Keidar et al. [61] have also used partial synchrony to informally reason about progress in consensus. A more detailed review of progress in consensus algorithms can be found in [62], but they do not provide any machine-checked proofs.

Variants of the actor model, such as FAM, have also been presented in the literature. Field and Varela [63] presented *transactors*, a fault-tolerant programming model for composing loosely-coupled distributed components running in unreliable environments using actors. Transactors can ensure global consistency in application-level protocols using checkpoints. One notable difference between transactors and FAM is that, unlike FAM, transactors support the loss of messages. However, they have not mechanically verified the properties of transactors. Charalambides et al. [64] introduced a system for typing coordination constraints in *parameterized actors*. Their system can be used for the static analysis of implementations of asynchronous concurrent systems using *session types* [65]. However, they have not investigated progress properties or developed any machine-checked proofs. Charalambides et al. [66] investigated the use of type systems to capture the progress properties of actor programs but have not mechanically verified the proofs of the properties that they studied.

Mechanical verification of the properties of actor systems has been presented by Musser and Varela [26]. They created a formalization of the actor model in Athena and verified many properties of actor computation at an abstract level,

independently of the details of a particular system of actors. Their work included 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, and was later extended by Dunn [32] to include correctness properties of actor systems with First-In-First-Out communication. Their work does not model actor failures or study progress in consensus algorithms, but our formalization of FAM and Synod in Athena has been inspired by their Athena formalizations. A formalization of the safety property of transactors in Athena was also presented by Boodman [33].

7 Discussion and future work

The conditions for progress Given the FLP impossibility result, eventual consensus under asynchronous conditions can only be guaranteed by making some strong assumptions. Assumptions like a unique leader and partial synchrony [58] have been used to guarantee eventual consensus in the literature (*e.g.* – [22,57]). However, since real-life communication is asynchronous and all candidates in DAC must be allowed to initiate proposals, we have investigated eventual consensus under purely asynchronous conditions and in the absence of a leader. The conditions for eventual consensus in Synod require that some agents will be nonfaulty (to bypass the impossibility result) and that eventually, some proposal will not be interrupted by higher-numbered proposals (to avoid livelocks). Under these conditions, Lamport's proof of progress for Paxos [22] becomes a special case of our proof of progress for Synod.

Predicate fairness in FAM We have shown that predicate fairness can be used to theoretically guarantee our conditions for eventual progress if the proposers keep retrying with higher proposal numbers. Predicate fairness is stronger than the fairness property of AM and it can be used for reasoning about the progress of different distributed protocols. It is important to note that predicate fairness is not necessary for our conditions for progress to be satisfied as the conditions may eventually be true even in a system that is not predicate fair. However, we believe that our formal analysis of predicate fairness in FAM is an important theoretical contribution toward the study of progress in distributed protocols like Synod implemented using actor systems.

Importance of eventual progress A guarantee of eventual progress does not provide any bound on the time that may be required for progress. In fact, in completely asynchronous settings, it is impossible to provide a bound on the time required for progress. However, real-life communication is asynchronous and the theoretical analysis of eventual progress under purely asynchronous conditions allows for the identification of subtle details that can halt progress by causing livelock or deadlock situations. Our analysis of eventual progress in Synod formally identified sufficient conditions required to prevent the protocol from encountering such undesirable situations.

Future work Consensus is a fundamental aspect of distributed aerospace applications like DAC. We have verified eventual consensus in Synod, thereby showing that the protocol is appropriate for use in DAC. However, there are other open challenges to the practical realization of DAC, such as bootstrapping decentralized coordination and providing guarantees for coordination that can be practically useful. Since the properties of real-life communication cannot be deterministically predicted, a potential direction of future work would be to investigate probabilistic properties of timely progress for Synod. Such probabilistic properties can be developed by stochastically modeling the message delays, which cannot be deterministically predicted in asynchronous conditions [67]. Another potential direction of work would be to model message loss in FAM by introducing additional meta-level transitions. This would allow us to weaken the progress conditions further by requiring that only a subset of the messages are not 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"*. Formal reasoning about Byzantine failures is also an interesting avenue of future work.

8 Conclusion

In this paper, we have identified a set of sufficient conditions under which the Synod consensus protocol can make eventual 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. We have also 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.

Acknowledgements This research was partially supported by the National Science Foundation (NSF), Grant No.—CNS-1816307 and the Air Force Office of Scientific Research (AFOSR), DDDAS Grant No.—FA9550-19-1-0054.

Appendix A relationship of FAM to AM

We present some theorems about the relationship of FAM to Varela's dialect of AMST (AM).

Theorem 3 *(FAM subsumes AM) Any valid computation path in AM is a valid path in FAM.*

Theorem 3 *Proof* -

Let $\pi = [\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < \infty]$ be a valid computation path in AM. Let **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 4 *(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.*

Theorem 4 *Proof* -

Let $\pi = [\kappa_i \xrightarrow{l_i} \kappa_{i+1} \mid i < n]$ be a valid computation path in FAM. Let **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 [**fun** : a], [**new** : a, a'], [**snd** : a], and [**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 [**stp** : a], [**bgn** : a] in FAM. Then we can remove the transition in 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 4 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.

The mechanical verification of Theorems 3 and 4 is out of the scope of this work.

References

1. Fischer MJ, Lynch NA, Paterson MS (1985) Impossibility of distributed consensus with one faulty process. J ACM 32(2):374–382. https://doi.org/10.1145/588058.588060
2. Thipphavong DP, Apaza R, Barmore B, Battiste V, Burian B, Dao Q, Feary M, Go S, Goodrich KH, Homola J (2018) Urban air mobility airspace integration concepts and considerations. In: Aviation technology, integration, and operations conference, p 3676. https://doi.org/10.2514/6.2018-3676
3. National Academies of Sciences, Engineering, and Medicine (2018) Assessing the risks of integrating unmanned aircraft systems (UAS) into the national airspace system. The National Academies Press, Washington. https://doi.org/10.17226/25143
4. Hopkin VD (2017) Human factors in air traffic control. CRC Press, London. https://doi.org/10.1201/9780203751718
5. Aweiss AS, Owens BD, Rios J, Homola JR, Mohlenbrink CP (2018) Unmanned aircraft systems (UAS) traffic management (UTM) national campaign II. In: AIAA information systems-AIAA infotech@ aerospace, p 1727. https://doi.org/10.2514/6.2018-1727
6. Paul S, Patterson S, Varela CA (2020) Collaborative situational awareness for conflict-aware flight planning. In: IEEE/AIAA digital avionics systems conference, pp 1–10. https://doi.org/10.1109/dasc50938.2020.9256620
7. Paul S, Kopsaftopoulos F, Patterson S, Varela CA (2020) Dynamic data-driven formal progress envelopes for distributed algorithms. In: Dynamic data-driven application systems, pp 245–252. https://doi.org/10.1007/978-3-030-61725-7_29
8. Paul S, Patterson S, Varela CA (2019) Conflict-aware flight planning for avoiding near mid-air collisions. In: AIAA/IEEE digital avionics systems conference, San Diego. pp 1–10 . https://doi.org/10.1109/dasc43569.2019.9081658
9. Lamport L (1998) The part-time parliament. ACM Trans Comput Sys 16(2):133–169. https://doi.org/10.1145/279227.279229
10. Lamport L (2001) Paxos made simple. ACM SIGACT News 32(4):18–25
11. Alquraan A, Takruri H, Alfatafta M, Al-Kiswany S (2018) An analysis of network-partitioning failures in cloud systems. In: USENIX symposium on operating systems design and implementation, pp 51–68
12. Lamport L, Malkhi D, Zhou L (2009) Vertical Paxos and Primary-Backup Replication. In: ACM Symposium on Principles of Distributed Computing, pp. 312–313 . https://doi.org/10.1145/1582716.1582783
13. Imai S, Varela CA (2012) A programming model for spatio-temporal data streaming applications. In: Dynamic data-driven applications systems, Omaha, NE, USA, pp 1139–1148 . https://doi.org/10.1016/j.procs.2012.04.123
14. Imai S, Blasch E, Galli A, Zhu W, Lee F, Varela CA (2017) Airplane flight safety using error-tolerant data stream processing. IEEE Aerosp Electr Sys Mag 32(4):4–17. https://doi.org/10.1109/maes.2017.150242
15. Agha G (1986) Actors: a model of concurrent computation in distributed systems. The MIT Press, Cambridge. https://doi.org/10.7551/mitpress/1086.001.0001
16. Hewitt C (1977) Viewing control structures as patterns of passing messages. Artif Intell 8(3):323–364. https://doi.org/10.1016/0004-3702(77)90033-9

17. Queille J-P, Sifakis J (1983) Fairness and related properties in transition systems-A temporal logic to deal with fairness. Acta Informatica 19(3):195–220. https://doi.org/10.1007/bf00265555
18. Arkoudas K, Musser D (2017) Fundamental proof methods in computer science: a computer-based approach. MIT Press, Cambridge. https://doi.org/10.1017/s1471068420000071
19. Weidenbach C, Dimova D, Fietzke A, Kumar R, Suda M, Wischnewski P (2009) SPASS version 3.5. In: International conference on automated deduction. Springer, pp 140–145. https://doi.org/10.1007/978-3-642-02959-2_10
20. Paul S, Agha GA, Patterson S, Varela CA (2021) Verification of eventual consensus in synod using a failure-aware actor model. In: NASA formal methods symposium (NFM). Springer, Cham , pp 249–267. https://doi.org/10.1007/978-3-030-76384-8_16
21. Howard H, Malkhi D, Spiegelman A (2016) Flexible Paxos: quorum intersection revisited. arXiv preprint. https://doi.org/10.48550/arXiv.1608.06696
22. Lamport L (2006) Fast Paxos. Distrib Comput 19(2):79–103. https://doi.org/10.1007/s00446-006-0005-x
23. Agha G, Mason IA, Smith S, Talcott C (1992) Towards a theory of actor computation. In: International conference on concurrency theory. Springer, pp 565–579 . https://doi.org/10.1007/bfb0084816
24. Varela CA (2013) Programming distributed computing systems. The MIT Press, Cambridge, MA
25. Agha GA, Mason IA, Smith SF, Talcott CL (1997) A foundation for actor computation. J Funct Progr 7(1):1–72. https://doi.org/10.1017/s095679689700261x
26. Musser DR, Varela CA (2013) Structured reasoning about actor systems. In: Workshop on programming based on actors, agents, and decentralized control. AGERE!. ACM, New York, NY, USA, pp 37–48. https://doi.org/10.1145/2541329.2541334
27. Ongaro D, Ousterhout J (2014) In search of an understandable consensus algorithm. In: 2014 USENIX annual technical conference, pp 305–319
28. Chlipala A (2013) Certified programming with dependent types: a pragmatic introduction to the coq proof assistant. MIT Press, Cambridge. https://doi.org/10.7551/mitpress/9153.003.0002
29. Wilcox JR, Woos D, Panchekha P, Tatlock Z, Wang X, Ernst MD, Anderson TE (2015) Verdi: a framework for implementing and formally verifying distributed systems. In: The 36th ACM SIGPLAN conference on programming language design and implementation, pp 357–368. https://doi.org/10.1145/2737924.2737958
30. De Moura L, Bjørner N (2008) Z3: An efficient smt solver. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, pp 337–340. https://doi.org/10.1007/978-3-540-78800-3_24
31. McMillan KL, Padon O (2018) Deductive verification in decidable fragments with Ivy. In: Static analysis. Springer, pp 43–55. https://doi.org/10.1007/978-3-319-99725-4_4
32. Dunn IW (2014) Proving correctness of actor systems using FIFO communication. In: Master's thesis, Rensselaer Polytechnic Institute (May 2014)
33. Boodman B (2008) Implementing and verifying the safety of the transactor model. In: Master's thesis, Rensselaer Polytechnic Institute (May 2008)
34. Manzano M (1996) Extensions of first-order logic, vol 19. Cambridge University Press, Cambridge, UK
35. Arkoudas K (2005) Simplifying proofs in fitch-style natural deduction systems. J Autom Reason 34(3):239–294. https://doi.org/10.1007/s10817-005-9000-3
36. Riazanov A, Voronkov A (2002) The design and implementation of VAMPIRE. AI Commun 15(2):91–110
37. Paul S, Agha GA, Patterson S, Varela CA (2021) Verification of eventual consensus in synod using a failure-aware actor model. In: Technical report, Rensselaer Polytechnic Institute, Department of Computer Science (March 2021). https://doi.org/10.48550/arXiv.2103.14576
38. De Prisco R, Lampson B, Lynch N (2000) Revisiting the Paxos algorithm. Theor Comput Sci 243(1–2):35–91. https://doi.org/10.1016/s0304-3975(00)00042-6
39. Chand S, Liu YA, Stoller SD (2016) Formal verification of multi-Paxos for distributed consensus. In: International symposium on formal methods. Springer, pp 119–136. https://doi.org/10.1007/978-3-319-48989-6_8
40. Lamport L (2002) Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co. Inc, Boston
41. Chaudhuri K, Doligez D, Lamport L, Merz S (2010) Verifying safety properties with the TLA+ proof system. In: International joint conference on automated reasoning. Springer, pp 142–148. https://doi.org/10.1007/978-3-642-14203-1_12
42. Padon O, Losa G, Sagiv M, Shoham S (2017) Paxos made EPR: Decidable reasoning about distributed protocols. In: Proceedings of the ACM on programming languages 1(Oopsla), p 108. https://doi.org/10.1145/3140568
43. Malkhi D, Lamport L, Zhou L (2008) Stoppable Paxos. Technical report, Microsoft research
44. Küfner P, Nestmann U, Rickmann C (2012) Formal verification of distributed algorithms. In: IFIP international conference on theoretical computer science. Springer, pp 209–224 . https://doi.org/10.1007/978-3-642-33475-7_15
45. Schiper N, Rahli V, Van Renesse R, Bickford M, Constable RL (2014) Developing correctly replicated databases using formal tools. In: IEEE/IFIP international conference on dependable systems and networks. IEEE, pp 395–406. https://doi.org/10.1109/dsn.2014.45
46. Bickford M, Constable RL, Rahli V (2012) Logic of events, a framework to reason about distributed systems. In: Languages for distributed algorithms workshop
47. Naumov P, Stehr MO, Meseguer J (2001) The HOL/NuPRL proof translator. In: International conference on theorem proving in higher order logics. Springer, pp 329–345. https://doi.org/10.1007/3-540-44755-5_23
48. Lamport L (2005) Real-time model checking is really simple. In: Advanced research working conference on correct hardware design and verification methods. Springer, pp 162–175. https://doi.org/10.1007/11560548_14
49. Rahli V, Guaspari D, Bickford M, Constable RL (2015) Formal specification, verification and implementation of fault-tolerant systems using EventML. Electr Commun EASST 72:1–15
50. Rahli V, Guaspari D, Bickford M, Constable RL (2017) EventML: specification, verification, and implementation of crash-tolerant state machine replication systems. Sci Comput Prog 148:26–48. https://doi.org/10.1016/j.scico.2017.05.009
51. Padon O, McMillan KL, Panda A, Sagiv M, Shoham S (2016) Ivy: safety verification by interactive generalization. ACM SIGPLAN Not 51(6):614–630. https://doi.org/10.1145/2980983.2908118
52. Drăgoi C, Henzinger TA, Zufferey D (2016) PSync: a partially synchronous language for fault-tolerant distributed algorithms. ACM SIGPLAN Not 51(1):400–415. https://doi.org/10.1145/2837614.2837650. (**ACM**)
53. Charron-Bost B, Schiper A (2009) The heard-of model: computing in distributed systems with benign faults. Distr Comput 22(1):49–71. https://doi.org/10.1007/s00446-009-0084-6
54. Debrat H, Merz S (2012) Verifying fault-tolerant distributed algorithms in the heard-of model. Arch Form Proofs 2012
55. Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL - a proof assistant for higher-order logic. Springer, Switzerland. https://doi.org/10.1007/3-540-45949-9
56. Hawblitzel C, Howell J, Kapritsos M, Lorch JR, Parno B, Roberts ML, Setty S, Zill B (2015) IronFleet: proving practical distributed

systems correct. In: Symposium on operating systems principles. ACM, pp 1–17. https://doi.org/10.1145/2815400.2815428
57. Hawblitzel C, Howell J, Kapritsos M, Lorch JR, Parno B, Roberts ML, Setty S, Zill B (2017) IronFleet: proving safety and liveness of practical distributed systems. Commun ACM 60(7):83–92. https://doi.org/10.1145/3068608
58. Dwork C, Lynch N, Stockmeyer L (1988) Consensus in the presence of partial synchrony. J ACM 35(2):288–323. https://doi.org/10.1145/42282.42283
59. Losa G, Dodds M (2020) On the formal verification of the stellar consensus protocol. In: Bernardo, B., Marmsoler, D. (eds) 2nd workshop on formal methods for blockchains, vol 84, California, LA, pp 1–9. https://doi.org/10.4230/OASIcs.FMBC.2020.9
60. Attiya H, Dwork C, Lynch N, Stockmeyer L (1994) Bounds on the time to reach agreement in the presence of timing uncertainty. J ACM 41(1):122–152
61. Keidar I, Rajsbaum S (2003) Open questions on consensus performance in well-behaved runs. In: Future directions in distributed computing. Springer, London, pp 35–39. https://doi.org/10.1007/3-540-37795-6_7
62. Chand S, Liu YA (2021) Brief Announcement: What's Live? Understanding distributed consensus. In: Proceedings of the 2021 ACM symposium on principles of distributed computing. Association for computing machinery, New York, NY, USA. pp 565–568. https://doi.org/10.1145/3465084.3467947
63. Field J, Varela CA (2005) Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on principles of programming languages, pp 195–208. https://doi.org/10.1145/1040305.1040322
64. Charalambides M, Dinges P, Agha G (2016) Parameterized, concurrent session types for asynchronous multi-actor interactions. Sci Comput Prog 115–116:100–126. https://doi.org/10.1016/j.scico.2015.10.006
65. Bocchi L, Murgia M, Vasconcelos VT, Yoshida N (2019) Asynchronous timed session types. In: Programming languages and systems. Springer, Cham, pp 583–610. https://doi.org/10.1007/978-3-030-17184-1_21
66. Charalambides M, Palmskog K, Agha G (2019) Types for progress in actor programs. In: Models, languages, and tools for concurrent and distributed programming, pp 315–339. https://doi.org/10.1007/978-3-030-21485-2_18
67. Paul S, Patterson S, Varela CA (2021) Formal guarantees of timely progress for distributed knowledge propagation. In: Formal methods for autonomous systems (FMAS). Electronic proceedings in theoretical computer science. vol 348, Open Publishing Association, The Hague, Netherlands, pp 73–91. https://doi.org/10.4204/EPTCS.348.5

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.