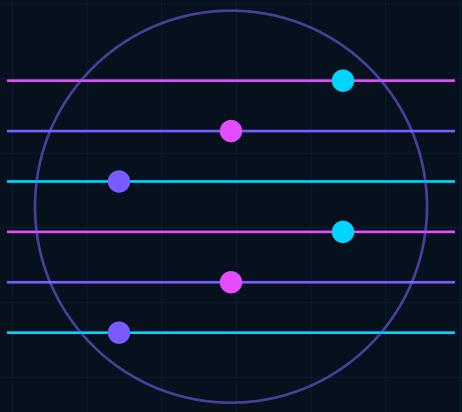
FORMAL METHODS

TLA+ Formal Verification for Distributed State Machines

Current-source review, evidence-bounded adoption decision, explicit limitations, reproducibility controls, and preserved source research note.



PERMANENT ID	EVIDENCE	ADOPTION	FRESHNESS
RES-029	A	ADOPT FOR CRITICAL STATE MA	STABLE WATCH

Freshness review: 2026-09-17 / Next scheduled review: 2027-03-16

RESEARCH CONTROL SURFACE

Required Research Fields

Each field remains independently reviewable; evidence strength does not imply production authority.

EVIDENCE GRADE — **A**

Strength of the retained source set and technical basis.

SOURCE AUTHORITY — **2 registered authorities**

TLA+ Tools and Specifications; Apalache Symbolic Model Checker

FRESHNESS — **STABLE WATCH**

Reviewed 2026-09-17; dynamic sources require event-driven revalidation.

CONFLICTS — **VISIBLE / NOT SILENT**

Differences between specification, vendor behavior, research claims, and reproduced evidence must remain explicit.

LIMITATIONS — **EXPLICIT**

No certification, procurement approval, legal conclusion, or unbounded production claim is created by this report.

REPRODUCIBILITY — **REQUIRED**

Material claims require exact versions, representative data, failure cases, artifacts, and repeatable acceptance criteria.

ADOPTION POSTURE — **ADOPT FOR CRITICAL STATE MACHINES**

Adopt TLA+ or equivalent formal specification for high-risk concurrent protocols, authorization state machines, durable workflows, consensus, and recovery logic. Map verified models to implementation tests and telemetry.

REVIEW TRIGGER — **2027-03-16**

Scheduled at 180 days; earlier on material source, vulnerability, implementation, or contradictory-evidence change.

Authority Register and Freshness Delta

Primary-source lineage is preserved; current-source changes are separated from unperformed implementation claims.

AUTHORITY 01

TLA+ Tools and Specifications

https://github.com/tlaplus/tlaplus

AUTHORITY 02

Apalache Symbolic Model Checker

https://apalache-mc.org/

2026-09-17 FRESHNESS DELTA

The TLA+ project continues to publish current-tool behavior and limitations; formal results remain model- and invariant-bounded.

Freshness policy: scheduled review + immediate review on source supersession, material release, vulnerability, retraction, or contradictory evidence.

Evidence Must Survive Contact With Reality

- Source authority is not implementation evidence. Product behavior, scale, security, interoperability, and operator fit remain environment-specific.
- Conflicting evidence is recorded rather than averaged away. Specification behavior, vendor implementation, preprint results, and production observations are separate evidence classes.
- Quantitative claims are informative until reproduced under documented hardware, software, data, topology, load, and failure conditions.
- Adoption posture is reversible. A stronger source set can coexist with a pilot-only or research-only recommendation when maturity, safety, economics, or operability remain unresolved.

ADOPTION / VALIDATION / REVIEW

Decision Gate and Validation Plan

ADOPTION POSTURE

ADOPT FOR CRITICAL STATE MACHINES

Adopt TLA+ or equivalent formal specification for high-risk concurrent protocols, authorization state machines, durable workflows, consensus, and recovery logic. Map verified models to implementation tests and telemetry.

1 / SOURCE	Pin exact versions, publication state, retrieved artifacts, and source hashes.
2 / PROTOTYPE	Demonstrate the core mechanism with representative data and instrumented execution.
3 / FAILURE	Exercise malformed input, dependency loss, stale state, overload, recovery, rollback, and misuse.
4 / BASELINE	Compare against an incumbent, classical, or simpler architecture using the same workload.
5 / ACCEPT	Record acceptance criteria, residual limitations, owner, expiration, and revalidation triggers.

EXPIRATION / REVIEW TRIGGER

Next scheduled review: 2027-03-16 (180-day cadence). Review sooner after material source revision, breaking implementation release, critical vulnerability, retraction, benchmark contradiction, changed workload, or changed legal/safety boundary.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-029 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Distributed systems engineers, AI engineers building multi-agent workflows, and platform architects designing durable execution runtimes, database consensus, and zero-trust state machines Companion Standards: MYTHOS-SWE-0007 (Formal Verification & Deterministic State Machine Standard), MYTHOS-DAT-0004 (Event Sourcing & Durable State), MYTHOS-DST-0001 (Durable Workflows & Sagas), RA-001 (Governed Multi-Agent Runtime)

The architecture of distributed systems validation has failed. Engineers attempt to verify complex, concurrent microservices and autonomous agent workflows using unit tests and integration tests. They write millions of lines of code, run them in a staging environment, and hope that the test suite covers the edge cases.

Testing is fundamentally incomplete. A unit test verifies one execution path; it cannot explore the combinatorial explosion of concurrent network delays, process crashes, and race conditions. When a distributed transaction deadlocks, or an autonomous agent enters a livelock, the engineers are mystified because "it passed all tests."

TLA+ (Temporal Logic of Actions) and its modern symbolic model checker, Apalache, shatter this empirical paradigm. TLA+ is not a programming language; it is a formal specification language based on mathematics. You write the blueprint of the state machine before writing the code.

This research note defines the architectural dynamics of formal verification. It dissects the mechanics of specifying state transitions, the physical constraints of state-space explosion, and the use of Apalache to mathematically prove the absence of deadlock and livelock in agentic workflows. Understanding these dynamics is a prerequisite for building the crash-proof, deterministic PANTHEON runtime and durable execution fabrics mandated by the Mythos Standards.

To understand formal verification, we must move from imperative code (how to do it) to declarative mathematics (what it is).

1.1 The State Machine (Variables and Init)

A TLA+ specification models a system as a state machine.

- Variables: The state of the system (e.g., `agent_state`, `pending_approvals`, `memory_graph`).
- Init: The initial predicate defining the starting state of the variables.

1.2 The Next-State Actions (The Transition)

- Next: A predicate defining how the system evolves. It is a disjunction of possible actions (e.g., `PlanAction \/ ExecuteTool \/ RequestHITL`).
- The Unchanged Macro: If an action does not modify a variable, it must explicitly state `UNCHANGED agent_state`. TLA+ forces the engineer to be hyper-aware of concurrency and side effects.

1.3 Invariants and Temporal Properties (The Proof)

- Invariants: Conditions that must be true in every reachable state (e.g., `Safety == agent_state # "CRASHED"`). If the model checker finds a single state where `agent_state == "CRASHED"`, the invariant is violated.

- Temporal Properties: Conditions about the sequence of states (e.g., `Liveness == []<>(agent_state = "IDLE")`—meaning "It is always eventually true that the agent returns to Idle"). This proves the absence of livelock.

While formal verification guarantees correctness, it introduces severe physical and cognitive constraints that make it difficult to apply naively.

2.1 The State Space Explosion

The model checker (TLC or Apalache) explores every possible interleaving of concurrent actions. If a system has 10 variables, each with 10 possible states, the state space is 10^{10}.

- The Flaw: Model checking a naive specification of a massive distributed database will run forever, exhausting the RAM of the verification server.
- The Mitigation: The engineer MUST abstract the specification. TLA+ is not for verifying the exact code; it is for verifying the algorithm. You model the essence of the consensus protocol, ignoring irrelevant details (like specific data payloads), drastically reducing the state space.

2.2 The Learning Curve (Math vs. Code)

Software engineers are trained in imperative programming (Python, Go). TLA+ is based on Zermelo-Fraenkel set theory and temporal logic.

- The Constraint: Engineers must learn to think in terms of sets, predicates, and state transitions, not `for` loops and `if/else` statements. This paradigm shift is the primary barrier to adoption.

2.3 The Specification-to-Code Gap

TLA+ proves the design is correct. It does not prove the implementation is correct.

- The Flaw: An engineer writes a perfect TLA+ spec, but then implements it in Rust and makes a pointer error. The implementation deadlocks, even though the spec is perfect.
- The Mitigation: The code MUST be a strict, traceable implementation of the TLA+ spec. The Mythos standard mandates that the state machine transitions in the code (e.g., the Temporal workflow steps) must map 1:1 to the actions in the TLA+ Next predicate.

The true power of formal verification is realized when applied to the complex, concurrent workflows of autonomous agents (like the PANTHEON runtime).

3.1 Verifying Multi-Agent Handoffs

In a multi-agent system, Agent A proposes an action, Agent B reviews it, and the Clio module checkpoints the state. Race conditions are inevitable.

- The Mechanic: A TLA+ spec models the concurrent execution of Nona (Planner), Aegis (Policy), and Morta (Executor). Apalache explores every possible interleaving of their network messages and process pauses.
- The Impact: The model checker mathematically proves that no matter how delayed the network is, or in what order the messages arrive, the system can never reach a state where an unauthorized action is executed, or the state machine deadlocks waiting for a message that will never arrive.

3.2 Proving Durable Execution Resilience

The Clio module (Temporal) must survive process crashes. If the agent crashes mid-tool-execution, the runtime must resume correctly.

- The Mechanic: The TLA+ spec includes a `Crash` action that can occur at any point. The temporal property proves that despite any number of crashes, the system eventually reaches a successfully completed or safely rolled-back state.

3.3 Apalache: Symbolic Bounded Model Checking

Traditional TLC checks states by enumerating them (explicit-state). Apalache uses symbolic execution and SMT solvers (like Z3) to check bounded traces.

- The Advantage: Apalache is vastly faster for complex data structures (like JSON payloads or memory graphs) because it reasons about the constraints mathematically rather than enumerating every possible value. It makes verifying deep, 10-step agentic workflows feasible on a laptop.

The fundamental shift of formal verification is the transition from "hope it works" to "mathematically proven."

- Mission-Critical Autonomy: An autonomous agent managing a power grid or financial portfolio cannot be tested into reliability. TLA+ provides the mathematical proof required to trust Level 5 autonomy.
- Consensus Protocol Design: Before building a custom distributed database or a DePIN smart contract, the protocol is specified in TLA+. This proves the consensus algorithm cannot fork, deadlock, or lose data under network partitions.
- Reduced Operational Fear: When a system is formally verified, the on-call engineer does not fear obscure race conditions. The system behaves exactly as the math dictates.

To deploy TLA+ in production, the Mythos Architecture enforces strict integration into the CI/CD pipeline and bounded verification contexts.

5.1 CI/CD Enforcement (Apalache in GitHub Actions)

Formal verification is useless if it is not continuously enforced.

- The Mitigation: The Mythos CI/CD pipeline MUST execute Apalache on every Pull Request that alters a state machine. The pipeline blocks the merge if the spec violates an invariant or temporal property.

5.2 Bounded Verification Contexts

You cannot formally verify the entire PANTHEON monolith.

- The Mitigation: The architecture MUST isolate the most critical, concurrent components (e.g., the Aegis policy evaluation loop, the Clio checkpointing logic) into bounded state machines. These bounded contexts are formally verified independently. The communication between them is governed by typed Protobuf contracts.

5.3 The "Spec-as-Documentation" Rule

A TLA+ spec is not a throwaway artifact.

- The Mitigation: The TLA+ specification is the authoritative documentation for the state machine. If an engineer asks, "What happens if the user cancels the workflow during HITL?", they do not read the code; they read the TLA+ spec. The code is merely an implementation detail of the math.

The strategy of relying on integration tests and staging environments to catch distributed concurrency bugs is architecturally bankrupt. The combinatorial explosion of concurrent states in modern AI agent workflows guarantees that tests will miss the exact edge case that causes a catastrophic outage. TLA+ and Apalache shift the paradigm from empirical testing to mathematical proof. By specifying the state machine before writing the code, we prove the absence of deadlock and livelock, transforming distributed systems from fragile, unpredictable webs into deterministic, mathematically sound engineering partners.

```
A unit test checks one path; a model checker proves all paths.
An integration test hopes for the best; a temporal property demands the truth.
A deadlock in production is a failure of imagination; a TLA+ violation is a failure of math.
```

```
The code is the guess; the specification is the absolute.

> Concurrency may be complex.
> Mathematical verification must be absolute.
```

End of Research Note RES-029

9. Source Register

Source	Authority and Status	Freshness Control
TLA+ Tools and Specifications https://github.com/tlaplus/tlaplus	Primary project repository Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Apalache Symbolic Model Checker https://apalache-mc.org/	Primary project documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-029_TLA_Plus_Formal_Verification_Distributed_State_Machines.md
Original source words	1526
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	180 days or event-driven trigger, whichever occurs first