Debugging Failures as Violated Transition Invariants

A Causal Systems Perspective

Author: Amogha Rao
Undergraduate, Software Engineering
Email: amoghagrao@gmail.com

Abstract

Modern debugging practices often hide observable failures instead of addressing the underlying defects that cause them. This work suggests that ongoing production failures continue because debugging focuses on *meaning invariants*, which are limits on observed system states, rather than *transition invariants*, which are limits on allowed state changes. We present a causal model that distinguishes between human reasoning errors, latent bugs, and runtime failures. We argue that enforcing transition invariants can prevent entire groups of execution paths from producing faults.

Problem

Recurring failures continue unabated in production systems despite sustained monitoring, patching, and alerting. Because failures are detected and mitigated in these systems, only to fail again due to different workloads or interleaving, the underlying causal faults remain

Key Claim

The persistent failures arise because debugging focuses on state invariants, such as count ≥ 0 - rather than transition invariants that constrain the evolution of the state. The imposition of transition invariants eliminates spurious execution paths.

Causal Model

- **Error:** A human reasoning mistake about system behavior.
- **Bug (Fault):** A latent defect introduced into system design or code.
- **Failure:** Observable incorrect runtime behavior.

Errors create bugs; bugs manifest as failures.

Case Study

A production system was sometimes producing negative figures for the number of currently connected users. In order to workaround the evident failure, engineers clamped the value to zero. The failures resurfaced under increased concurrency. The root cause was a race condition due to concurrent, non-atomic increments on a shared counter. The implicit reasoning error was an assumption of sequential execution. The implicit invariant that was missed was:

User-count updates must be atomic with respect to execution context.

Proposed Contribution

1. Classification of meaning vs transition invariants
2. Causal debugging model
3. Methodology to derive transition invariants from design assumptions
4. Empirical validation via simulation and production-like case studies

Planned Evaluation

Metrics include:

- Failure recurrence rate
- Mean-time-to-resolution (MTTR)
- Invariant-violation vs prevention comparison

Objective

To shift debugging practice from symptom suppression to invariant-level fault prevention.