Formal Modeling and Analysis of Distributed Systems

Finding Critical Bugs Early!

Ankush Desai (ankushpd@)

Sr. Applied Scientist, Database Systems @ Amazon Web Services



ormal Methods

Its not just about the **Proofs**, its also about the **Process**!



Traditional testing approaches fail to provide the strong assurance!

Not uncommon to find bugs in production after deployment



Formal Methods!

"formal methods: any process that involves writing formal specifications and checking systems behavior against those specifications"

Thinking abstractly, formally, above coding



People confuse Programming with Coding. Coding is to Programming what Typing is to Writing

Key is to abstract away unwanted details and think of the system above coding, that improves developers ability to understand systems better.

You don't understand something until you've written it down carefully – carefully enough to explain it to somebody else (formal models). And if you haven't done that, you're just thinking you understand it,

From Talks:

The Man Who Revolutionized Computer Science With Math

How to think clearly as a programmer with mathematics and TLA+ - Leslie Lamport @ HLF 2019

Challenges with wide spread adoption of Formal Methods!

- [Steep Learning Curve] : X is not programmer friendly ...
 - Mathematical modeling language, huge overhead of maintaining the models
 - Formal design and implementation go out of sync.
 - Developers think of distributed system as communicating state machines, instead of a monolithic state transition system.
- [Scalability]: X does not scale for verification of real-world distributed systems.
 - Fails to find deep or corner case bugs.
- [Connection to Implementation]: what about the gap between design and implementation.

X is any formal methods tool.



- P is a state-machine based programming framework for modeling, specifying, and formal reasoning of distributed systems.
- P program is a collection of asynchronously communicating state machines.

Models and specifications are implemented as state machines.

• P supports a scalable backend checker.

State-of-the-art techniques to scale analysis to large distributed systems

```
machine Client
{
  var server : Server;
  var nextReqId : int;
  var lastSuccessfulRespId: int;

  start state Init {
    entry (payload : Server)
    {...}
  }

  state StartPumpingRequests {
    entry
    {...}
    on eResponse do (resp: tResponse){...}
    on eRequest goto ServiceRequests;
  }
```

P Specifications: Runtime Monitors

```
event eA: int;
event eB: int;
spec AlternatingAB observes eA, eB {
  var lastVal: int; // local state
  start state WaitForA {
    on eA goto WaitForB with (payload: int) {
      assert payload > lastVal;
     lastVal = payload;
    }
  state WaitForB {
    on eB goto WaitForA with (payload: int) {
      assert payload > lastVal;
      lastVal = payload;
```

- Specifications are runtime monitors that listen to messages, maintain local state, and assert global invariants!
- Specifications are programs!

P Tutorials and Documentation

https://p-org.github.io/P/

< Home		Q Search	P on Github 2.5k Stars · 147 Forks
Ρ		—	Table of contents
Home			Let the fun begin!
What is P?			
Getting Started	>		
Tutorials	>		
Advanced User Guide	>		
Language Manual	>		
Case Studies		Formal Modeling and Analysis of Distributed Systems	
Videos			
Publications		nuget v1.1.15 license MIT () CI on Windows passing () CI on Ubuntu passing () CI on MacOS passing	
contributing to r		Challenge: Distributed Systems are notoriously hard to get right. Programming these systems is challenging because of the need to reason about correctness in the presence of myriad possible interleaving of messages and failures. Unsurprisingly, it is common for service teams to uncover correctness bugs after deployment. Formal methods can play an important role in addressing this challenge!	
		 P Overview: P is a state machine based programming language for formally modeling and specifying complex distributed systems. P allows programmers to model their system design as a collection of communicating state machines. P supports several backend analysis engines (based on automated reasoning techniques like model checking and symbolic execution) to check that the distributed system modeled in P satisfy the desired correctness specifications. Impact: P is currently being used extensively inside Amazon (AWS) for analysis of complex distributed systems. For example, Amazon S3 used P to formally reason about the core distributed protocols involved in its strong consistency launch. Teams across AWS are now using P for thinking and reasoning about their systems formally. P is also being used for programming 	
		safe robotics systems in Academia. P was first used to implement and validate the USB device driver stack that ships with Microsoft Windows 8 and Windows Phone.	
		Experience and lessons learned : In our experience of using P inside AWS, Academia, and Microsoft. We have observed that P has helped developers in three critical ways: (1) P as a thinking tool : Writing formal specifications in P forces developers to think about their system design rigorously, and in turn helped in bridging gaps in their understanding of the system. A large fraction of the bugs can be eliminated in the process of writing specifications itself! (2) P	

as a bug finder: Model checking helped find corner case bugs in system design that were missed by stress and integration testing. (3) **P helped boost developer velocity**: After the initial overhead of creating the formal models, future update,s and feature additions could be rolled out faster as these non-trivial changes are rigorously validated before implementation.

$History \ of \ P \ (Slides \ taken \ from \ my \ PhD. \ talk)$



Users in Academia

- Used P to model and verify distributed IEEE 1588 Precision time protocol [CAV 2015].
 - Used as a standard in telecommunication, industrial automation, and CERN LHC particle accelerator.
- We were able to reproduce a long debated bug of *"endlessly circulating frames"* in IEEE1588 using our verification approach.
 - The counter example was confirmed using a simulator and well received by the IEEE 1588 standards community.

Users in Microsoft

Windows	Azure (P#)	Office			
(P)		Office Client			
USB host	Node Service				
UART class extension	Learning Service				
Hid class	AZSM				
Media Agnostic USB	Blockchains				
Bluetooth					
In collabora	In collaboration with <u>UPenn</u> Grasp lab, we used P for programming safe robotics systems for Darpa demos				



11

Formal Reasoning of S3 Strong Consistency Design using P

AWS News Blog

Amazon S3 Update – Strong Read-After-Write Consistency

by Jeff Barr | on 01 DEC 2020 | in Amazon Simple Storage Services (S3), Launch, News | Permalink | 🗩 Comments | 📌 Share



Marc Brooker @MarcJBrooker

Very cool to see TLA+, P, and Dafny code on the keynote stage at AWS re:Invent.





Lessons Learned (P as a Thinking Tool)

Formal Modeling: "P as a Thinking Tool"

Design as reference executable specification, bridges the gap in understanding.



"How the system does what it is supposed to do?"

Think about **detailed design**, interaction with other components, messages exchanged, assumptions about other components.

"Does the system do what it is supposed to do?"

Safety and liveness properties. **Global invariants** both external and internal that the system must satisfy.

"What inputs and faults can the system tolerate?"

Think about the kind of inputs and faults the system must tolerate.

Lessons Learned (P as a systematic explorer or bug finder)

Model Checking: "P as a Systematic explorer or bug finder" Uncover non-trivial bugs/scenarios, gain confidence in design.

P Checker: Systematically explores nondeterministic behaviors.

- Two forms of non-determinism:
 - Data non-determinism (e.g., all possible inputs, random(), timeouts)
 - Scheduling non-determinism (e.g., interleaving of messages, failures)

Model Checking as a search problem



Scalable Exploration: Partial Order Reduction, Search Prioritization, Symbolic Execution, ..

Continued Assurance

- Leverage AWS to scale systematic exploration:
 - Explore billions of states using 100s of compute nodes every day.
 - Explore combinations of rare events (extremely low probability).
- Integrate model checking into CI/CD pipeline.

Lessons Learned (P boosts velocity of developers)

• Maintenance (quick iterations): "P boosts velocity of developers"

Rapid iterations, release features faster, eliminate bugs early instead of in production.

Service teams can try out risky optimizations or protocol changes first using P, gain confidence, and then mimic those changes in the implementation.

Higher confidence in design before moving to implementation.

What about connection to Implementation?

• We want to check specifications on the system implementation.



Systems already produce service logs!

What about connection to Implementation?

- *Runtime monitoring* is a well-known technique for checking that a system *trace* satisfies a desired *property*.
 - Trace = Service log
 - Property = P specification



• Idea: Check P Specifications on service logs!



PObserve architecture; components in orange are service-specific and are provided by developers.

Improving observability of distributed systems using formal specifications.

We define *distributed systems' observability* as the ability to check formal specifications at all the phases of development lifecycle (design, testing, and production).

ormal Methods

Its not just about the **Proofs**, its also about the **Process**!

