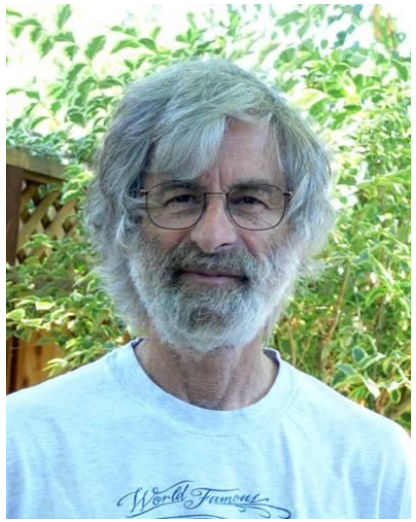
TLA+简介

谁在没有设计图的情况下盖房子?

建筑师在砌砖或钉钉子之前会制定详细的计划。但很少有程序员在开始编码之前，就画出一个关于他们的程序会做什么的草图。

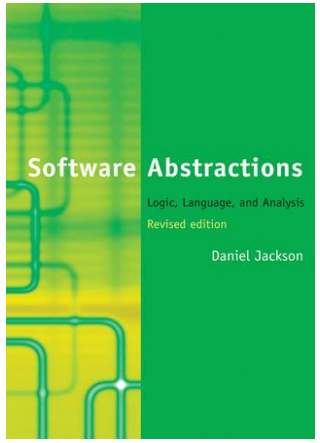


---Leslie Lamport

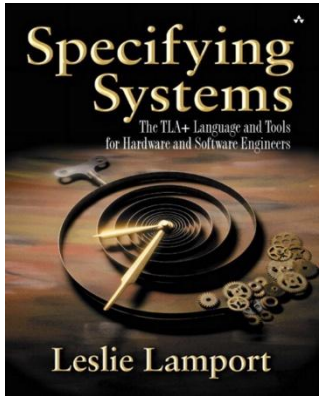
https://lamport.azurewebsites.net/tla/tla.html

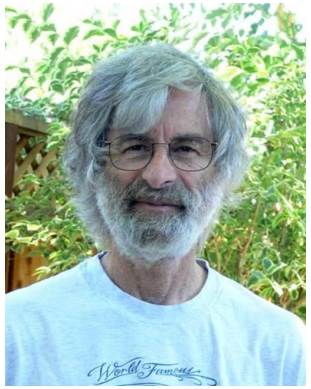
形式化的语言/工具

- Alloy



- Z-Notation
- TLA+



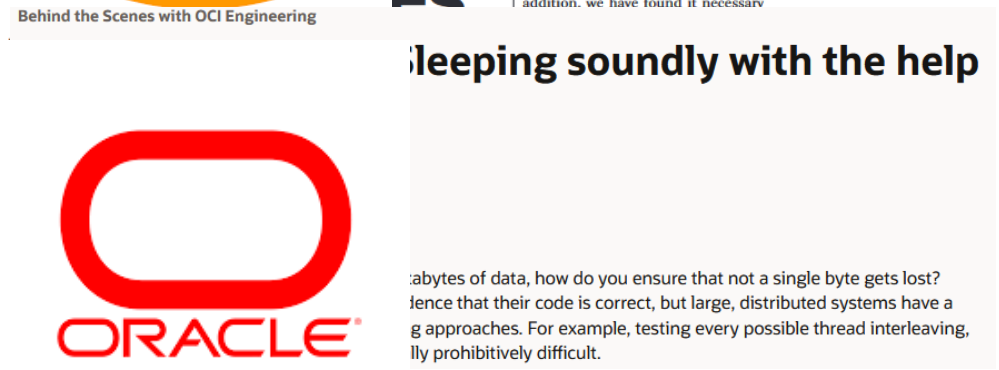


系统软件质量如何保证?

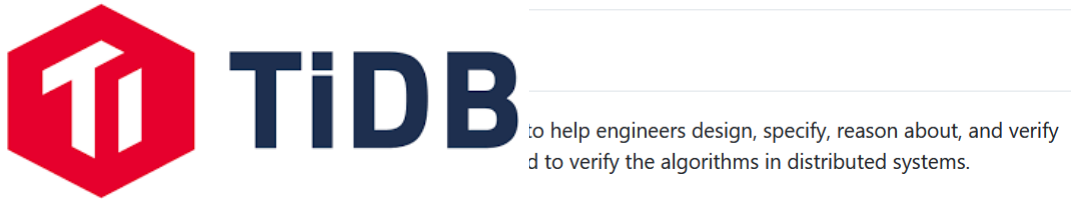
- **系统软件**
  - 正确性关键系统
  - 数据库，操作系统
  - 并发系统，分布式系统
  - 由并发、异步消息产生的非确定性非常难于测试
    - 例，并发控制、共识…
- **盖房子前，要不要先画设计图?**

业界应用形式化方法

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary



TLA+ in TiDB



o help engineers design, specify, reason about, and verify
d to verify the algorithms in distributed systems.

Microkernel Goes General: Performance and Compatibility in the HongMeng Production Microkernel

Haibo Chen[1,2], Xie Miao[1], Ning Jia[1], Nan Wang[1], Yu Li[1], Nian Liu[1], Yutao Liu[1], Fei Wang[1], Qiang
engwei Xu[1]

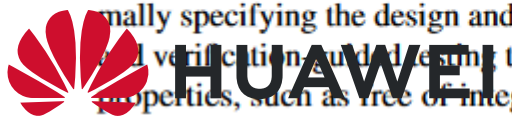
ter performance than their Linux counterparts. The critical components of *HM* are semi-formally verified [55] by formally specifying the design and using automated verification ity
verification and testing to validate the crucial security properties, such as free of integer and buffer overflow. *HM*

eXtreme Modelling in Practice

A. Jesse Jiryu Davis
MongoDB, Inc.
1633 Broadway
New York, NY 10019
e@mongodb.com

Max Hirschhorn
MongoDB, Inc.
1633 Broadway
New York, NY 10019
max.hirschhorn

Judah Schvimer
MongoDB, Inc.
1633 Broadway
New York, NY 10019
judah@mongodb.com



HarmonyOS NEXT

为什么选择TLA+

- **Why Amazon Chose TLA+**
  - 适合用于大规模分布式/并发系统
  - 丰富的表达能力
  - 工具功能完善

Amazon使用TLA+发现设计Bug

System	Components	Line count	Benefit
S3	Fault-tolerant low-level network algorithm	804 PlusCal	Found 2 design bugs. Found further design bugs in proposed optimizations.
	Background redistribution of data	645 PlusCal	Found 1 design bug, and found a bug in the first proposed fix.
DynamoDB	Replication and group-membership systems (which tightly interact)	939 TLA$^+$	Found 3 design bugs, some requiring traces of 35 steps.
EBS	Volume management	102 PlusCal	Found 3 design bugs.
EC2	Change to fault-tolerant replication, including incremental deployment to existing system, with zero downtime	250 TLA$^+$ 460 TLA$^+$ 200 TLA$^+$	Found 1 design bug.
Internal distributed lock manager	Lock-free data structure	223 PlusCal	Improved confidence. Failed to find a liveness bug as we did not check liveness.
	Fault tolerant replication and reconfiguration algorithm	318 TLA$^+$	Found 1 design bug. Verified an aggressive optimization.

TLA+ Specification

- **Variables**
  - 变量表示系统的状态
  - 变量由关键字VARIABLES修饰
- **Actions**
  - 描述系统状态之间的转移关系
    使用逻辑公式描述状态转移关系
- **Constants**
  - 常量用于定义特定的数据值
  - 由关键字CONSTANTS修饰
  - 在模型检查过程之前分配值
  - 在模型检查过程中无法更改
- **Properties**
  - 定义目标系统的行为约束
  - Safety：坏事永远不会发生
    - 如事务的原子性不会被违反
  - Liveness：正确的行为最终会发生
    - 如事务最终都会提交或回滚，不会一直停留在中间状态

CONSTANTS

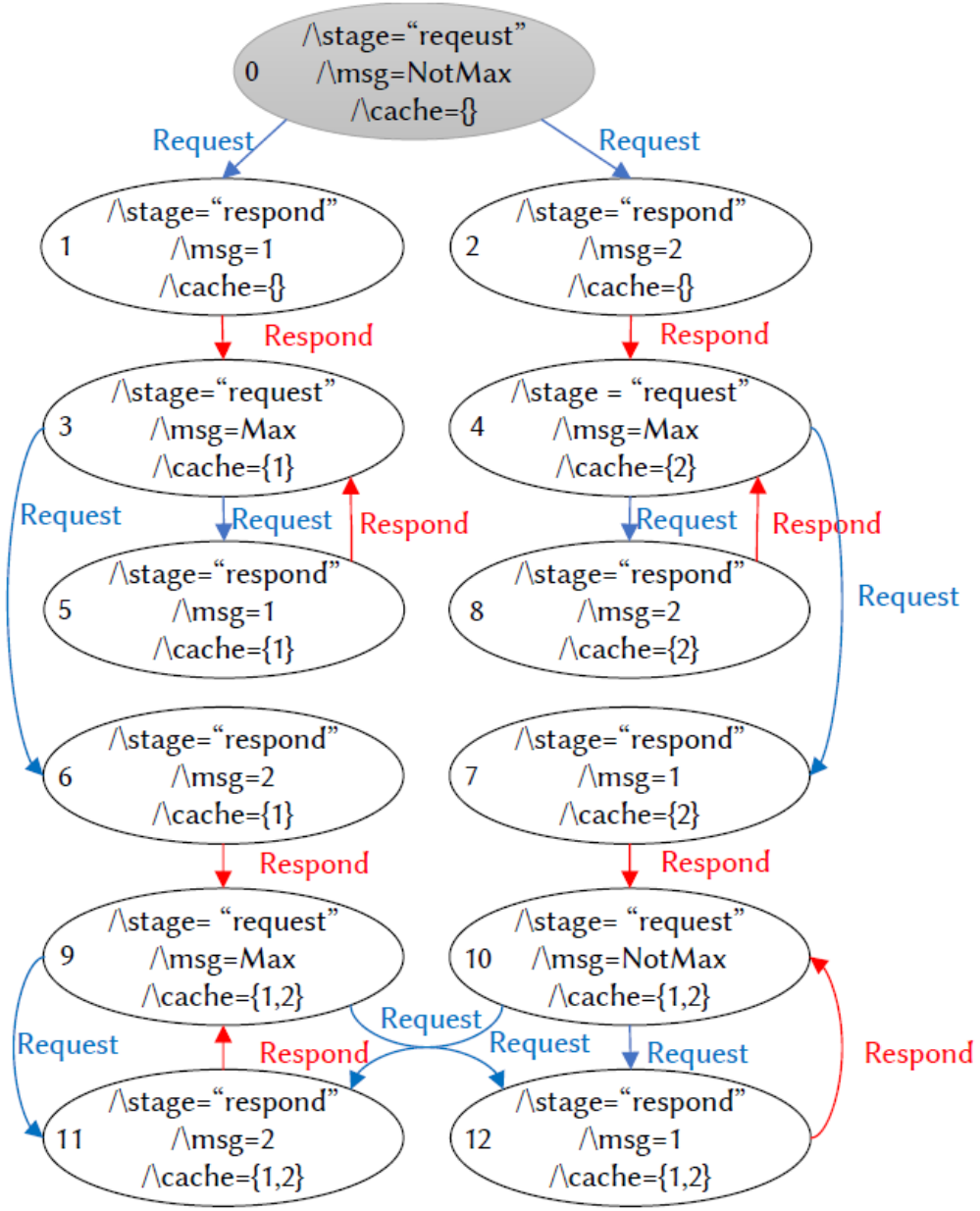
VARIABLES

ACTIONS

PROPERTIES

```
1. CONSTANTS Max, NotMax, Data, Nil
2. VARIABLES msg, cache, stage
3. vars ≜ <msg, cache, stage>
4. Init ≜ ∧ msg = Nil
5.       ∧ stage = "request"
6.       ∧ cache = {}
7.
8. getMax(S) ≜ CHOOSE t ∈ S : ∀s ∈ S : t ≥ s
9. Request(data) ≜ ∧ stage = "request"
10.                ∧ stage' = "respond"
11.                ∧ msg' = data
12.                ∧ UNCHANGED ≪ cache ≫
13. Respond ≜ ∧ stage = "respond"
14.           ∧ stage' = "request"
15.           ∧ cache' = cache ∪ {msg}
16.           ∧ msg' = IF msg = getMax(cache')
17.                    THEN Max ELSE NotMax
18. Next ≜ ∨ ∃ d ∈ Data : Request(d)
19.        ∨ Respond
20. Spec ≜ Init ∧ □[Next]_vars
21.
22. Invariant ≜ Cardinality(cache) ≤ Cardinality(Data)
```

- 集合*cache*存储来自消息*msg*的数据
  - 用两个值响应：
- ***Max***：如果*msg* 是*cache*中目前为止最大的值
  - ***NotMax***：如果不是最大的



- **从规范自动生成状态空间**
- **并验证属性是否满足**
- **每个状态都会被标记一个唯一编号**
- **状态0是初始状态**

Part 1: 为什么需要形式化方法

分布式系统设计的困难

- 并发系统状态空间巨大
- 人为推理容易遗漏极端情况
- 测试无法覆盖所有 interleavings
- 复杂协议容易隐藏边界条件 bug

传统验证方法的问题

- Code Review 依赖经验
- Unit Test 覆盖有限
- Integration Test 难以枚举所有状态
- 系统规模越大，隐含 bug 越难发现

形式化方法的核心思想

- 用数学逻辑描述系统
- 系统行为用状态机表示
- 自动枚举所有可能状态
- 验证 Safety 与 Liveness 属性

Part 2: TLA+ 基础

什么是 TLA+

- TLA+ = Temporal Logic of Actions
- 由 Leslie Lamport 提出
- 用于描述并发和分布式系统
- 核心是状态 + 状态转移

TLA+ 的组成

- VARIABLES：系统状态
- CONSTANTS：系统常量
- Init：初始状态
- Next：状态转移规则

TLA+ 的优势

- 表达能力强
- 适合描述协议
- 支持模型检查
- 被 Amazon / Microsoft 等公司使用

Part 3: Modeling

系统 = 状态机

- 系统状态由 VARIABLES 表示
- 状态变化由 Actions 描述
- Next 定义所有合法转移
- 系统执行 = 状态序列

状态空间

- 每个状态是一组变量取值
- 状态通过 Action 相互连接
- 所有状态组成状态图
- 模型检查遍历状态图

数学的语言

$$\{f \in [1..N \longrightarrow 1..N] : \forall y \in 1..N : \exists x \in 1..N : f[x]=y\}$$

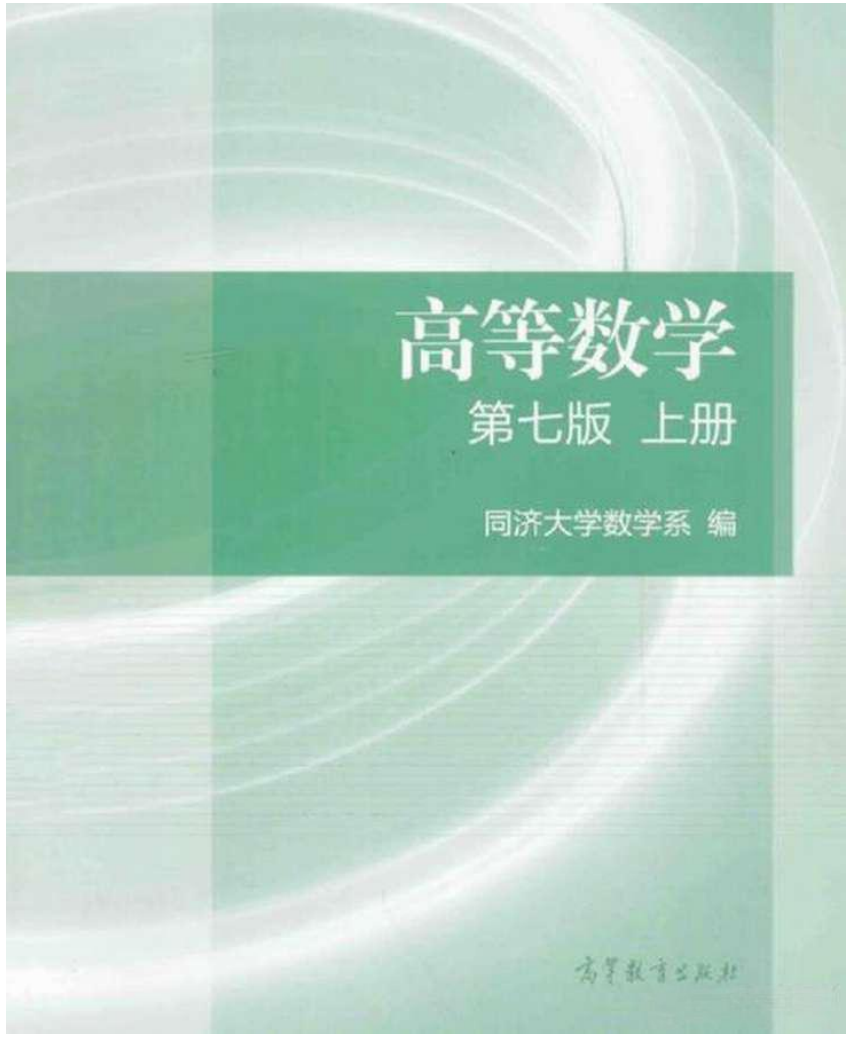
- **"…. I then asked the audience to raise their hands if they could look at this formula and understand it. About half the audience raised their hands. I then asked all the mathematicians to lower their hands. Almost no hands remained up. Hardly any of the computer scientists could understand it. "**
- **https://lamport.azurewebsites.net/tla/math-knowledge.html**

Why Don't Computer Scientists Learn Math?

Last modified 28 March 2017

你一定学过的数学

$$\lim_{x \to a} f(x) = b$$



Epsilon-Delta（ε, δ)语言

$$\lim_{x \to a} f(x) = b$$

$$(\forall \varepsilon > 0)\,(\exists \delta > 0)\,(\forall x \in \mathbb{R})\,(0 < |x - a| < \delta \implies |f(x) - b| < \varepsilon)$$

“要多小，就有多小”

Part 4: Properties

Example: Cache System

- cache 保存已接收消息
- 收到新消息 msg
- 判断是否为最大值
- 返回 Max / NotMax

```
1. CONSTANTS Max, NotMax, Data, Nil
2. VARIABLES msg, cache, stage
3. vars ≜ <msg, cache, stage>
4. Init ≜ ∧ msg   = Nil
5.       ∧ stage = "request"
6.       ∧ cache = {}
7.
8. getMax(S) ≜ CHOOSE t ∈ S : ∀s ∈ S : t ≥ s
9. Request(data) ≜ ∧ stage  = "request"
10.                ∧ stage' = "respond"
11.                ∧ msg'   = data
12.                ∧ UNCHANGED ≪ cache ≫
13. Respond ≜ ∧ stage  = "respond"
14.           ∧ stage' = "request"
15.           ∧ cache' = cache ∪ {msg}
16.           ∧ msg' = IF msg = getMax(cache')
17.                    THEN Max ELSE NotMax
18. Next ≜ ∨ ∃ d ∈ Data : Request(d)
19.        ∨ Respond
20. Spec ≜ Init ∧ □[Next]_vars
21.
22. Invariant ≜ Cardinality(cache) ≤ Cardinality(Data)
```

Safety 属性

- 系统永远不会进入错误状态
- 例如：数据不会丢失
- 例如：不会出现两个 leader
- 形式：Invariant

Liveness 属性

- 正确的事情最终会发生
- 例如：请求最终得到响应
- 例如：事务最终提交
- 避免系统永久停滞

Safety / Liveness

- 分布式系统需要同时保证两者

Part 5: Model Checking

Model Checker 工作方式

- 枚举所有状态
- 构建状态图
- 检查属性是否满足
- 如果违反，生成 counter example

状态爆炸问题

- 系统规模越大，状态越多
- 需要抽象模型
- 使用有限域
- 逐步扩大模型规模

Part 6: 工业案例

Amazon 使用 TLA+

- 验证 DynamoDB
- 验证 S3
- 验证分布式锁
- 发现多处设计缺陷

微软使用 TLA+

- Azure Storage
- Hyper-V
- 分布式协议

Part 7: 学习路径

如何学习 TLA+

- 理解状态机建模
- 掌握基本语法
- 使用 TLC model checker
- 从简单协议开始

适合建模的系统

- 分布式系统
- 一致性协议
- 锁和并发控制
- 事务系统

总结

- TLA+ 是一种用于描述系统行为的形式化规范语言
- 适合验证并发系统
- 可以在设计阶段发现 bug
- 形式化方法正在工业界普及