南京大學

研 究 生 毕 业 论 文

（申 请 硕 士 学 位）

论 文 题 目 **PaxosStore** 中共识协议 **TPaxos** 的 规约、精化与定理证明

作 者 姓 名 易星辰

学科、专业方向 计算机技术

研 究 方 向 分布式算法

指 导 教 师 黄宇 教授、魏恒峰 助理研究员

2021 年 **5** 月 **30** 日

学　　号：**MF1833088**

论文答辩日期：**2021** 年 **5** 月 **25** 日

指 导 教 师：　　（签字）

TPaxos in PaxosStore: Specification, Refinement, and Theorem Proving

by

Xingchen Yi

Supervised by

Professor Yu Huang, Assistant Researcher Hengfeng Wei

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER
in
Computer Technology

Department of Computer Science and Technology
Nanjing University

May 25, 2021

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：PaxosStore 中共识协议 TPaxos 的规约、精化与定理证明

计算机技术 专业 2018 级硕士生姓名：易星辰

指导教师（姓名、职称）：黄宇 教授、魏恒峰 助理研究员

摘　要

为了提高系统的可用性与容错性，分布式存储系统通常采用数据复制技术，将同一份数据以副本形式存放在多个物理节点上。然而这也带来了数据一致性问题。分布式共识协议是解决数据一致性问题的重要手段，它可以在多个副本之间提供数据强一致性。Paxos 协议是一种经典的分布式共识协议，被广泛用于需要提供强一致性的分布式系统中，包括 Google 的 Chubby 分布式锁服务系统、微软的 Autopilot 自动数据中心管理系统以及阿里的 OceanBase 企业级数据库等等。

PaxosStore 是腾讯开发的高可用分布式存储系统，现已用于全面支持微信核心业务。PaxosStore 实现了分布式共识协议 Paxos 的一种变体，我们称之为 TPaxos。TPaxos 的新颖之处在于它的“统一性”：它为每个参与者维护统一的状态类型，并采用统一格式的消息进行通信。然而，这种设计方案也带来了 TPaxos 与 Paxos 之间的诸多差异，给理解 TPaxos 造成了障碍。其次，虽然腾讯开源了 TPaxos 协议的核心代码（包括伪代码与 C++ 代码），TPaxos 仍缺少抽象而精确的形式化规约。最后，就我们所知，TPaxos 的正确性尚未经过必要的数学论证或者形式化工具的检验。为了解决上述问题，我们在构建 TPaxos 规约的基础上，不仅刻画了其与经典 Paxos 的关系，并且严格证明了它的正确性。具体包括以下四个主要贡献：

首先，我们从经典的 Paxos 协议出发，论证如何逐步推导出 TPaxos 协议。基于这种推导，我们可以将 TPaxos 看作 Paxos 的一种自然变体，更易于理解。

其次，我们给出了 TPaxos 协议的 TLA$^+$ 形式化规约。在开发规约的时候，我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处：在消息处理阶段，参与者（作为接受者角色）是先作出“不再接受具有更小编号的提议”的承诺（Promise）还是先接受（Accept）提议？这导致对 TPaxos 的两种

不同理解，并促使我们提出 TPaxos 的一种变体，称为 TPaxosAP。在 TPaxosAP 中，参与者先接受提议后作承诺。

接着，我们使用精化（refinement）技术建立了 Paxos 协议变体间的精化关系。对于 TPaxos，我们仍采用 Paxos 所使用的 Voting 投票机制，建立了从 TPaxos 到 Voting 的精化关系。对于 TPaxosAP，Voting 无法完全刻画 TPaxosAP 的行为，我们提出了一种新的投票机制 EagerVoting。并且，我们使用精化技术论证了 EagerVoting 和 Voting 的等价性。Voting 与 EagerVoting 投票机制准确刻画了 Paxos，TPaxos 与 TPaxosAP 之间的异同。

最后，我们复用已有的 Paxos 证明框架，使用 TLAPS（TLA$^+$ Proof System）定理证明系统严格证明了 TPaxos 与 TPaxosAP 的正确性。

关键词： 分布式共识；Paxos 协议；精化关系；TLA$^+$/TLAPS；定理证明；模型检验；

南京大学研究生毕业论文英文摘要首页用纸

THESIS: TPaxos in PaxosStore: Specification, Refinement, and Theorem Proving

SPECIALIZATION: Computer Technology

POSTGRADUATE: Xingchen Yi

MENTOR: Professor Yu Huang, Assistant Researcher Hengfeng Wei

Abstract

To achieve high system availability and fault tolerance, distributed data storage systems usually adopt the data replication technology, replicating multiple copies of the same data at multiple replicas. However, data replication introduces the data consistency problem. The distributed consensus protocols can address this problem by providing strong data consistency between multiple replicas. Paxos is a classic distributed consensus protocol, and it has been widely used in distributed systems that require strong consistency, including Google's distributed lock service system (Chubby), Microsoft's automated data center management system (Autopilot), Alibaba's enterprise database (OceanBase), etc.

PaxosStore is a highly available distributed storage system developed by Tencent Inc. to support the comprehensive business of WeChat. PaxosStore employs a variant of Paxos which is a classic protocol for solving distributed consensus. We call it TPaxos in this paper. The originality of TPaxos lies in its "uniformity": it maintains for each participant a unified state type and adopts a universal message format for communication. However, this design choice brings various differences between TPaxos and Paxos, rendering TPaxos hard to understand. Moreover, although the core code (including both pseudocode and source code in C++) for TPaxos is publicly available, there still lacks a formal specification of TPaxos. Finally, as far as we know, TPaxos has not yet been manually proven or formally checked. To address these issues, we construct the TPaxos protocol and not only describe its relationship with Paxos, but also prove its correctness rigorously, including the following four main contributions.

First, we demonstrate how to derive TPaxos from classic Paxos step by step. Based

on this derivation, TPaxos can be regarded as a natural variant of Paxos and is much easier to understand.

Second, we describe TPaxos in TLA$^+$, a formal specification language. In the course of developing the TLA$^+$ specification for TPaxos, we uncover a crucial but subtle detail in TPaxos which is not fully clarified: Upon messages, do the participants (as Acceptors) make promise that no proposals with smaller proposal numbers will be accepted before accepting proposals or vice versa? This leads to two different interpretations of TPaxos and motivates us to propose a variant of TPaxos, called TPaxosAP. In TPaxosAP, the participants accept proposals first and then make promise.

Third, we establish refinement mappings between the Paxos variants. For TPaxos, we still use the Voting mechanism used by Paxos to establish a refinement mapping from TPaxos to Voting. Particularly, since Voting cannot capture all the behaviors of TPaxosAP, we propose a new voting mechanism, called EagerVoting. We also demonstrate the equivalence between EagerVoting and Voting by refinement. The Voting and EagerVoting mechanisms precisely characterize the differences and similarities among Paxos, TPaxos, and TPaxosAP.

Finally, we use TLAPS (TLA$^+$ Proof System) to rigorously prove the correctness of TPaxos and TPaxosAP, based on the existing proof framework for Paxos.

keywords: Distributed Consensus, Paxos Protocol, Refinement Mapping, TLA$^+$/TLAPS, Theorem Proving, Model Checking

目　　次

插图清单

附表清单

第一章　绪论

伴随着互联网应用数量的快速增长，各种类型的数据呈现爆炸式增长，各大公司的数据量已经达到 EB/ZB 级别。为了避免单点存储成为系统高可用与高扩展的瓶颈，分布式数据系统通常会采用数据复制技术，即在多个物理节点上保存同一份数据的多个副本。数据副本的引入缩短了用户与服务器之间的距离，降低了访问的延迟。并且，避免了高并发访问下单点失效引起的系统崩溃，提高了系统的可用性与容错性。

1.1　分布式共识与 Paxos 协议

数据副本不可避免的引入了一致性问题，即分布式共识（Consensus）问题，它要求所有副本之间数据是一致的[1][2]。数据一致性是系统的基石，一旦无法保障数据一致性，该系统显然是不值得信任的。实践证明，共识协议是解决分布式共识问题的重要方法。通俗理解，共识是让多个参与者就某个值达成一致。考虑这样的场景，在一个分布式系统中，如果所有节点的初始状态一致，每个节点执行相同的操作序列，那么他们最终能达到一个一致的状态。在该场景下，共识协议保证了每个节点看到相同的操作序列。

Paxos（也称 Basic Paxos 或 Single-Decree Paxos）是一个备受瞩目的共识协议[3][4]。由 Leslie Lamport 于 1990 年在论文“The Part-time Parliament”中提出，从工程的角度而言它提供了一种最大化保障分布式系统一致性的机制。自从 Paxos 提出至今，衍生了 Fast Paxos、Disk Paxos、raft 及 Zookeeper 的 Zab 协议等诸多变体。Paxos 及其变体广泛用于需要提供强一致性服务的分布式系统中。在 Google 介绍其 Chubby 系统时提到[5][6]，到目前为止（Chubby 诞生），谷歌所有系统中涉及到异步共识的部分都采取以 Paxos 协议为核心的思想。同样，微软的 CosmosDB 以及 Autopilot 自动数据中心管理系统[7] 实现了 Paxos 协议保障了数据的高可用、持久性以及一致性。在国内，阿里云的分布式文件系统 PolarFS 实现了 raft 协议的变体 ParallelRaft[8]，为上层的分布式数据库 PolarDB 提供了高可用与高性能服务。微信于 2017 年推出来两个基于 Paxos 的

开源分布式数据库 PhxPaxos 和 PaxosStore[9]。

针对不同的使用场景以及功能需求，分布式系统中 Paxos 及其变体的实现往往具有独有的特性。如 PolarFS 中的 ParallelRaft 改进了 raft，支持日志的乱序提交与乱序执行，提高了系统在高并发场景下的性能。本文关注 PaxosStore 系统所实现的 Paxos 变体，我们称之为 TPaxos。TPaxos 针对微信应用程序中的高吞吐、低延迟等要求，消除了协议过程中的冗余状态与消息，从而降低了系统执行的复杂度。

1.2 Paxos 协议验证的动机与挑战

PaxosStore 系统全面支持微信的各种服务，包括即时通讯、社交网络、在线支付等业务。而作为系统的基础协议，TPaxos 协议的正确性至关重要。然而，验证 TPaxos 的正确性面临着许多挑战。

第一，从系统实现角度而言，验证一个分布式协议的正确性通常比实现本身要困难。谷歌无法确保 Chubby 系统是绝对可靠的，雅虎也无法给出 Zookeeper 的完整证明过程。事实上，绝大部分分布式系统都是依赖在实践过程中得到大量数据的训练，经过长时间的测试及实战，逐渐达成将错误收敛直到归零的目标。因此，找到一种耗时更短、效率更高的验证方法显得尤为重要。

第二，从 TPaxos 协议而言，TPaxos 的特性给验证带来了许多挑战。在 TPaxos 中，每个参与者（即，数据副本节点）都同时具有提议者（Proposer）、接受者（Acceptor）与学习者（Learner）三种角色。TPaxos 的新颖之处在于它的“统一性”：它为每个参与者维护统一的状态类型并采用统一格式的消息进行通信。在 TPaxos 中，参与者通过互相交换统一类型的消息，不断更新状态，最终达成共识。相比而言，经典的 Paxos 协议（及其众多变体）为不同的角色维护不同的状态类型，并且需要使用四种消息类型，以区分准备阶段（Prepare Phase；包含 Phase1a 与 Phase1b 两个子阶段）与接受阶段（AcceptPhase；包含 Phase2a 与 Phase2b 两个子阶段）中的两轮双向通信。这种设计方案可以带来精简而高效的系统实现。然而，这也带来了 TPaxos 与 Paxos 之间的诸多差异，给理解 TPaxos 造成了障碍。其次，虽然腾讯开源了 TPaxos 协议的核心代码（包括论文中的伪代码与实际系统中的 C++ 代码[①]，

[①] https://github.com/Tencent/paxosstore

TPaxos 仍缺少抽象而精确的规约。最后，据我们所知，TPaxos 的正确性尚未经过必要的数学论证或者形式化工具的检验。

1.3 本文贡献

针对上述挑战，本文有四个主要贡献：

- 第一，我们论证如何从 Paxos 出发，逐步推导出 TPaxos。首先，我们通过合并三种角色，统一参与者的状态类型。其次，基于统一的状态类型，我们可以统一消息内容。然后，通过合并某些子阶段，并避免使用消息状态信息区分协议阶段，我们可以得到一种比 TPaxos 更为精简的统一消息类型。最后，我们指出 TPaxos 在消息处理阶段所作的优化，从而推导出最终的 TPaxos 版本。
- 第二，我们给出了 TPaxos 的 TLA$^+$ 形式化规约。与已有的 Paxos 的 TLA$^+$ 规约类似，TPaxos 采用“已发送消息集合”作为通信信道以及消息发送与接收动作的抽象。这种抽象使得该规约不依赖于代码层面的具体通信机制。更重要的是，在开发 TLA$^+$ 规约的时候，我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处：在消息处理阶段，参与者（作为接受者角色）是先作出“不再接受具有更小编号的提议”的承诺（Promise）还是先接受（Accept）提议？这导致对 TPaxos 的两种不同理解，并促使我们提出 TPaxos 的一种变体，称为 TPaxosAP。在 TPaxosAP 中，参与者先接受提议后作承诺。虽然在 TLA$^+$ 规约层面，TPaxosAP 与 TPaxos（先作承诺后接受提议）相差甚小，但它却体现了不同的投票机制。具体来讲，Paxos 协议的投票机制 Voting 仍适用于 TPaxos，却不能完整刻画 TPaxosAP 的行为。我们将举例说明 TPaxosAP 与 TPaxos 的不同之处，并论证 TPaxosAP 的正确性。
- 第三，我们使用精化技术（refinement）构建了 Paxos 协议变体间的精化关系。对于 TPaxos，我们仍采用 Paxos 所使用的 Voting 机制，建立了从 TPaxos 到 Voting 的精化关系。对于 TPaxosAP，我们首先提出了一种新的投票机制，称作 EagerVoting。EagerVoting 允许参与者在接受提议的同时，作出比 Paxos/TPaxos 更“激进”的“不再接受具有更小编号的提议”的承诺。然后，我们使用精化技术论证了 EagerVoting 和 Voting 的等价性。接着，我们建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus

的精化关系，并使用 TLC 模型检验工具验证上述精化关系的正确性。我们建立的精化关系从本质上揭示了经典 Paxos、TPaxos 与 TPaxosAP 核心的一致性，即采用了相同的投票机制 Voting 及其等价的 EagerVoting，这从本质上证明了 TPaxos 以及 TPaxosAP 的正确性。针对其他 Paxos 变体，我们可以采用相同的方法构建精化关系以论证其正确性。

- 第四，我们使用 TLAPS 定理证明系统严格证明了 TPaxos 与 TPaxosAP 的正确性。我们复用已有的 Paxos 证明框架，给出了 TPaxos 与 TPaxosAP 精化 Consensus 的完整证明过程。

1.4 本文组织结构

本文余下部分的组织结构如下：

第二章介绍本文的相关工作。首先简要介绍 PaxosStore 系统以及 Paxos 协议的诸多变体，然后总结形式化验证在工业界与学术界的使用情况，着重介绍精化技术及其应用。

第三章介绍准备知识。首先介绍本文使用的形式化语言 TLA$^+$ 及其相应的定理证明系统 TLAPS，接着给出 Paxos 协议的理论描述以及 Paxos 的 TLA$^+$ 规约，最后阐述目前已有的 Paxos 精化关系。

第四章介绍 TPaxos 协议。首先给出从 Paxos 到 TPaxos 的逐步推导，然后使用 TLA$^+$ 语言对 TPaxos 进行规约，并且提出了一个新的协议 TPaxosAP，最后给出 TPaxos 以及 TPaxosAP 的正确性证明。

第五章介绍精化关系。我们构建了 Paxos 变体间的精化关系，揭示了 TPaxos、TPaxosAP 以及经典 Paxos 核心的一致性。

第六章介绍 TPaxos 的定理证明。我们使用 TLAPS 定理证明系统严格证明了 TPaxos 的正确性。

第七章使用 TLC 模型检验工具验证了精化关系并且展示了实验结果。

第八章总结全文，并讨论可能的未来工作。

第二章　相关工作

本章介绍 TPaxos 验证的相关工作。首先，介绍 PaxosStore 的设计背景以及整体架构。然后，总结 Paxos 的多种变体及其验证工作，包括使用模型检验技术以及精化技术的验证工作。最后，介绍形式化语言 TLA$^+$ 在工业界中的应用。

2.1　腾讯 PaxosStore 系统简介

作为国内最流行的社交软件，微信日活跃用户高达 10 亿，有 3.3 亿用户进行了视频通话，1.2 亿用户发表朋友圈，其中照片 6.7 亿张，短视频 1 亿条[①]。在这种高吞吐的场景下，一个高可用且可靠的存储系统显得尤其重要。同时，微信业务广泛，提供的服务不仅仅局限于消息传递，还包括各种小程序、移动支付、公众号等。这也要求存储系统能够支持多种数据服务。基于以上要求，微信设计出分布式存储系统 PaxosStore[②]。

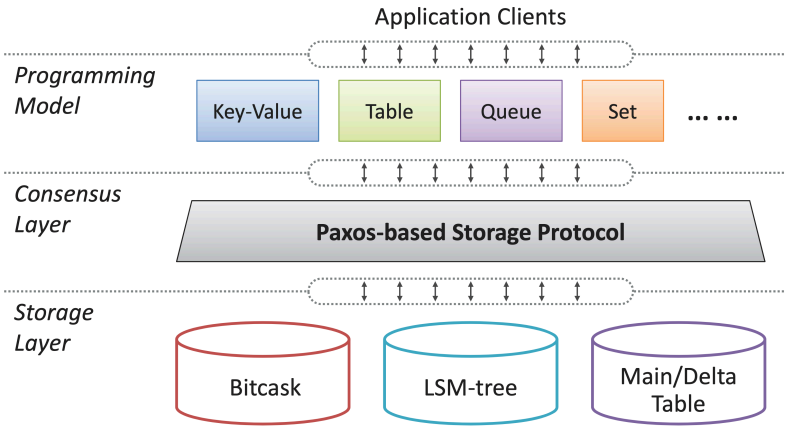

图 2-1: PaxosStore 整体架构

如图 2-1所示，PaxosStore 系统包括三层结构，编程模型、共识层、存储层。编程模型给外部服务提供不同的数据结构，共识层实现了基于 Paxos 的存

[①]https://mp.weixin.qq.com/s/LwkXAhTHx3fkToQMsy5lDg
[②]https://github.com/Tencent/paxosstore

储协议，而存储层包含多个存储引擎，这些引擎基于不同的存储模型满足了不同服务的需求。不同于传统的分布式数据库，PaxosStore 将共识层提取出来，并且共识层对底层的存储引擎提供数据一致性服务。这样，所有支持的存储模型可以共享提供数据一致性的通用服务，从而使得跨模型的数据共识变得方便。

本文关注的重点就是 PaxosStore 共识层正确性的验证，共识层的提取方便了我们的抽象建模。在具体的验证过程中，我们可以暂时忽略具体的数据，不考虑存储方式、存储引擎等问题。

2.2 Paxos 的变体及其验证

Paxos 协议衍生出了许多变体[10]，如 Disk Paxos[11]、Cheap Paxos[12]、Fast Paxos[13]、GeneralizedPaxos[14]、Stoppable Paxos[15]、Vertical Paxos[16]、Byzantine Paxos、EPaxos（Egalitarian Paxos）[17]、Raft[18]、CASPaxos[19] 等。本文关注腾讯开发的分布式存储系统 PaxosStore 中实现的 TPaxos[9] 变体。TPaxos 的新颖之处在于它的" 统一性": 它为每个参与者维护统一的状态类型，并采用类型统一的消息进行通信。我们从 Paxos 出发，论证了如何逐步推导出 TPaxos。基于这种推导，我们可以将 TPaxos 看作 Paxos 的一种自然变体。

使用形式化规约语言描述分布式协议并使用相应的模型检验工具进行验证可以有效提高协议的可信度[20] 。近年来，研究者使用 TLA+/TLC 描述并验证了 Paxos 协议及其多种变体。Lamport 等人使用 TLA+ 分别描述了 Paxos[21]、Fast Paxos[13]、Disk Paxos[11] 以及 Byzantine Paxos[22]，并使用 TLC 在一定规模上验证了它们的正确性。Moraru 在博士论文中给出了 EPaxos 的 TLA+ 规约。最近，Sutra 发现并纠正了其中的错误[23]。Ongaro 给出了 Raft 协议的 TLA+ 规约①。在本文，我们使用 TLA+ 描述了 TPaxos。在开发规约的时候，我们发现 TPaxos 协议描述中存在至关重要但并未完全阐明的微妙之处：在消息处理阶段，参与者是先作出" 不再接受具有更小编号的提议" 的承诺还是先接受提议？这促使我们发现了 TPaxos 的一种变体，称之为 TPaxosAP。与 TPaxos 相比，TPaxosAP 改动很小，但却体现了一种不同于 Voting[21] 的投票机制。

精化（Refinement）[24][25] 技术有助于理解 Paxos 各种变体的正确性以及它们之间的关系。Lampson 提出了一个抽象的 Paxos 协议[26]（Abstract Paxos；记

①https://github.com/ongardie/raft.tla

为 AP）。AP 刻画了 Paxos 协议的核心，但是由于它使用了全局信息，无法直接在分布式系统中实现。接着，作者使用精化/模拟（refinement/simulation）技术分别建立了 Paxos、Disk Paxos 以及 Byzantine Paoxs 与 AP 的关系。在 AP 的视角下，它们都可以看作 AP 的具体实现。Lamport 等人提出了抽象的投票机制 Voting[21]，用于刻画接受者" 作承诺" 与" 接受提议" 两种核心行为。Voting 可以看作分布式共识问题的集中式解决方案，而 Paxos 是 Voting 的分布式实现。实际上，Lamport 等人给出了从 Paxos 到 Voting 以及从 Voting 到 Consensus 的精化映射[21]。同样基于 Voting，Lamport 还使用精化技术从 Paxos 推导出了 Byzantine Paxos[27]，并使用 TLC 验证了它们之间的精化关系。Maric 等人在 HO（Heard-Of）模型[28] 下研究了多种 Paxos 变体[29]。作者首先为它们建立了一个共同的抽象，也称为 Voting。根据 Voting 行为的差异，这些 Paxos 变体可以被归为三类。然后，作者构建了这三类变体之间的精化关系，并使用定理证明器 Isabelle/HOL 进行了形式化验证。在本文，我们分别建立了从 TPaxos 以及 TPaxosAP 到 Consensus 的精化关系。特别地，在 TPaxosAP 方面，我们首先在 Voting 的基础上提出了一种适用于 TPaxosAP 的投票机制 EagerVoting，然后建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系，并且使用精化技术验证了 EagerVoting 和 Voting 的等价性[①]。

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.

图 2-2: TLA$^+$ 在 Amazon 系统中的应用

[①]https://github.com/Starydark/PaxosStore-tla/blob/master/specification/EagerVoting.tla

2.3 TLA+ 在分布式系统中的应用

自从 TLA+ 提出以来，受到了各大互联网公司的青睐。TLA+ 在许多大型系统中发挥着重要的作用。

图 2-2是 Amazon 在他们实际系统中使用 TLA+ 的收获[30]。在图上所有的系统中，TLA+ 都体现了它的重要价值。不仅仅帮助优化系统的结构，还发现了很多细微的 bug。如在对 DynamoDB 系统验证时，发现了 3 个 bug，甚至包括一个需要 35 个步骤才能定位到的 bug。

不仅仅是 Amazon，微软在他们的云服务系统 Azure 上也深度使用 TLA+。从系统设计开始，TLA+ 就帮助他们完善相关算法，比如 CosmosDB 团队使用 TLA+ 来指定一致性服务需要满足的规范。再到系统测试时，TLA+ 很好地保障并且改善了系统的稳定性，如 Azure 的网络团队发现并纠正了一些 bug①。

① http://lamport.azurewebsites.net/tla/industrial-use.html

第三章　准备知识

本章先介绍形式化规约语言 TLA⁺ 与其相应的定理证明系统 TLAPS，然后介绍分布式共识问题，最后介绍经典的 Paxos 协议，包括它的 TLA⁺ 的规约与精化关系。

3.1　TLA⁺ 与 TLAPS

3.1.1　TLA⁺ 简介

TLA⁺ 是 Leslie Lamport 基于时序逻辑 TLA（Temporal Logic of Actions）[31] 开发的一种形式化规约语言[32]，适用于描述并发及分布式系统。TLA⁺ 将系统建模成状态机。一个状态机由它可能的初始状态（initial states）与一组动作（actions）来刻画。在 TLA⁺ 中，一个状态是系统中所有变量的实例化。一个行为是状态的序列。一个动作是新旧状态之间的转移，刻画了系统执行指令前后的变化。例如，对于指令 $x := x + 1$，TLA⁺ 采用动作 $x' = x + 1$ 进行描述。TLA⁺ 使用一个包含带撇变量（表示新状态的值）与不带撇变量（表示旧状态的值）的公式来描述动作。

一般而言，一个系统可以表达为 TLA 中的一个形如 $Spec \triangleq Init \wedge [Next]_{vars} \wedge L$ 的时序公式。其中，*Init* 和 *Next* 都是状态谓词（结果为布尔值），对系统状态进行刻画。*Init* 刻画了系统的初始状态，*Next* 是所有动作的析取式，描述了系统的所有动作。*vars* 是包含系统所有变量的元组，$\Box$ 是“总是（always）”的时序操作符。表达式 $[Next]_{vars}$ 为真当且仅当 *Next* 为真（即某个动作为真，意味着该动作对应的指令被系统执行），或者 *vars* 在新旧状态中保持不变，后者是 TLA⁺ 的一个特性，有利于构建系统间的精化关系。*L* 用于检测系统的活性（Liveness）。

在 TLA⁺ 中，*L* 通常是 *WF* 的析取式，*WF* 指弱公平性（weak fairness）。$WF_{vars}(A)$ 刻画了动作 *A* 的公平性，即要求 4 如果动作 *A* 从某个时刻开始是持

表 3-1: 本文使用的的 TLA+ 操作符

	操作符	含义
逻辑	$CHOOSE\, x \in S : p$	选择集合 S 中满足条件 p 的元素 x (通常用于符合条件的 x 是唯一的情况下)
集合	SUBSET S	S 的幂集
	$\{e : x \in S\}$	将 e 作用在 S 中所有元素得到的集合 比如 $\{x^2 : x \in S\}$
	$x \in S : p$	S 中满足条件 p 的元素构成的集合
函数	$f[e]$	函数 f 作用在参数 e 上
	$[x \in S \mapsto e]$	对于 $x \in S$, 使得 $f[x] = e$ 的函数 f
	$[f \text{ EXCEPT } ![e_1] = e_2]$	函数 $\hat{f} : \hat{f}[e] = \begin{cases} e_2, & \text{if } e = e_1 \\ f[e], & \text{otherwise} \end{cases}$
	$[f \text{ EXCEPT } ![c] = e]$, 其中 e 包含符号 @	e 中的 @ 表示 $f[c]$
记录	$e.h$	记录 e 的域 h
	$[h_1 \mapsto e_1, \ldots, h_n \mapsto e_n]$	域 h_i 为 e_i 的记录
	$[h_1 : S_1, \ldots, h_n : S_n]$	满足域 h_i 属于 S_i 的所有记录构成的集合
	$[r \text{ EXCEPT } !.h = e]$	记录 $\hat{r} : [h_1 : S_1, \ldots, h : e, \ldots, h_n : S_n]$
	$[r \text{ EXCEPT } !.h = e]$, 其中 e 包含符号 @	e 中的 @ 表示 $r.h$
元组	$e[i]$	元组 e 的第 i 个分量
动作操作符	e'	动作的新状态中 e 的值
	UNCHANGED e	e 不变:$e' = e$
	$[A]_e$	动作 A 成立或者 e 不变:$A \vee (e' = e)$
时序操作符	$\Box F$	F 在所有情况下均成立 ($\Box$ 表示"always")
	$\Diamond F$	F 最终成立 ($\Diamond$ 表示"eventually")
	$WF_e(A)$	动作 A 的弱公平性 (weak fairness)

续可执行的，则 A 最终将被执行。$WF_{vars}(A)$ 的定义如下:

$$WF_{vars}(A) \triangleq \Box(\Box ENABLED\langle A\rangle_{vars} \Rightarrow \Diamond\langle A\rangle_{vars})$$

其中，$\langle A\rangle_{vars}$ 表示动作 A 被执行，$ENABLED\langle A\rangle_{vars}$ 表示动作 A（在当前状态下）是可执行的，$\Diamond$ 是“最终（eventually）”的时序操作符。

TLA+ 在 TLA 的基础上，加入了一阶谓词逻辑以及 ZF 集合论，从而支持丰富的数据类型与表达式。表格 3-1总结了本文使用到的逻辑与集合操作符（operator）。文献[33] 给出了完整的 TLA+ 操作符列表。

TLA+ 规约以模块（module）的形式组织在一起。在每个模块中，我们可以声明常量（CONSTANTS）与变量（VARIABLES）或者提出定理（THEOREM）。一个模块 M 可以通过扩展其它模块 $M_1, \ldots, M_n$ 的方式引入声明、定义与定理；在模块 M 中，写作 EXTENDS $M_1, \ldots, M_n$。模块也可以被实例化。考虑 M 模块中的实例化语句：

$$IM_1 \triangleq \text{INSTANCE } M_1 \text{ WITH } p_1 \leftarrow e_1, \ldots, p_n \leftarrow e_n$$

其中，p_i 包含了 M_1 中的所有常量与变量，e_i 是 M 中的合法表达式。该语句是通过将 M_1 中的 p_i 替换为 M 中相应的 e_i 来进行 M_1 的实例化，我们可以使用 $IM_1!F$ 来访问模块 M_1 中的表达式 F。此外，TLA+ 中的隐式替换规则允许我们在 e_j 与 p_j 相同时省略 $p_j \leftarrow e_j$ 子句。

TLC[①]的模型检验工具，它通过遍历 TLA+ 规约的有穷状态空间来验证规约是否满足具体的性质[34]。然而有些分布式系统的状态并非有穷。例如，考虑一个包含多个服务器的分布式系统，服务器（定义为 $Server$）的数量可以任意大，甚至无穷。为了能够验证这一类系统规约，TLC 通过模型（model）的方式限制了这些系统的状态数，即 TLC 可以在模型中实例化服务器为有穷集合，如 $Server \triangleq \{s_1, s_2, s_3\}$。如果对于值 $\{s_1, s_2, s_3\}$ 进行任意重排（如将 s_1 替换为 s_2，s_2 替换为 s_3，s_3 替换为 s_1）不影响 TLC 的验证结果，那么我们可以将 $Server$ 标记为对称集[35]（SymmetrySet）以减少 TLC 需要遍历的状态空间。

在 TLA+ 中，精化（refinement）关系就是逻辑蕴含（logical implication）关系。例如，考虑模块 $AbsModule$ 中的 $AbsSpec$ 规约以及模块 $ImplModule$ 中的 $ImplSpec$ 规约。$AbsSpec$ 包含变量 $x_1, \ldots, x_m, y_1, \ldots, y_n$，$ImplSpec$ 包括变量 $x_1, \ldots, x_m, z_1, \ldots, z_p$。令 X、Y 与 Z 分别代表变量组 $x_1, \ldots, x_m$、$y_1, \ldots, y_n$ 与 $z_1, \ldots, z_p$。为了验证 $ImplSpec$ 实现/精化了 $AbsSpec$（也称 $ImplSpec$ 是 $AbsSpec$ 的精化；即 $ImplSpec \Rightarrow AbsSpec$），我们需要证明：对于每一个满足 $ImplSpec$ 的行为，都存在一种关于变量组 Y 的赋值方式，使得赋值后所得的行为（考虑变量组 X 与 Y）满足 $AbsSpec$。具体而言，对于每个 y_i，我们仅使用 X 与 Z 定义表达式 $\overline{y_i}$，在 $AbsSpec$ 中做替换 $y_i \leftarrow \overline{y_i}$，得到新的规约 $\overline{AbsSpec}$，然后证明 $ImplSpec$ 实现/蕴含了 $\overline{AbsSpec}$。这里，替换 $y_i \leftarrow \overline{y_i}$ 被称为精化映射（refinement mapping）。为了使用 TLC 检验在精化映射 $y_i \leftarrow \overline{y_i}$ 下，$ImplSpec$ 实现了 $AbsSpec$，我们在模块 $ImplModule$ 中添加定

[①] http://lamport.azurewebsites.net/tla/toolbox.html

义 $AbsSub \triangleq INSTANCE\ AbsModule\ WITH\ y_1 \leftarrow \overline{y_1}, \ldots, y_n \leftarrow \overline{h_n}$，并检验定理 $THEOREM\ ImplSpec \Rightarrow AbsSub!AbsSpec$。

3.1.2 TLAPS

TLA+ 定义证明系统 TLAPS[36][37]（TLA+ Proof System①）是一种机械化的交互式定理证明工具。交互式意味着用户可以引导整个证明过程。具体而言，用户首先使用 TLA+ 构造证明过程，证明通常是层次化结构[38]。接着，TLAPS 自动将证明过程转换并发送给后端的定理证明器。TLAPS 附带有多个定理证明器，常用的包括 CVC3（一个 SMT 定理证明器）、Zenon 以及 Isabelle[39,40]，并且 TLAPS 默认依次调用这三个定理证明器以完成证明。TLAPS 将输出失败日志当且仅当以上三个定理证明器均证明失败。TLAPS 也支持一些用户自定义功能，包括指定使用某些定理证明器、设定每个证明目标的超时时间等等。

TLA+ 的证明通常是声明式的，其目标是使证明独立于检查的证明器。具体来说，每一步证明步骤中只需要给出依赖的事实，不需要说明如何处理这些事实。从而，证明的编写者即便缺乏实际证明器工作原理的知识也可以完成 TLA+ 证明的编写。

我们给出一个简单的编写 TLA+ 证明的案例。*Spec* 是构造的规约，*Invariant* 是定义的一个不变式。我们要证明算法的正确性，即证明 *Invariant* 是 *Spec* 的一个不变式。整体的证明框架如下所示：

$Spec \triangleq Init \wedge \Box[Next]_{vars}$
THEOREM $Spec \Longrightarrow \Box Invariant$
⟨1⟩1. $Init \Longrightarrow Invariant$
⟨1⟩2. $Invariant \wedge [Next]_{vars} \Longrightarrow Invariant'$
⟨1⟩3. QED

证明的目标是 $Spec \Rightarrow \Box Invariant$，即不变式 *Invariant* 始终成立。整体的证明由一系列步骤构成，其中最后一步是 QED，并且每一步通常紧跟着具体的证明过程。⟨1⟩1 中前面的 1 表示分层，后面的 1 表示证明的步骤。目标被证明完成当且仅当所有的步骤都被证明，我们可以以任何顺序证明这些步骤，也可以同时进行证明。经验表明，QED 步骤最先证明是比较好的选择方式。

① http://tla.msr-inria.inria.fr/tlaps/content/Home.html

QED 表示“证明目标”，这里指 $Spec \Rightarrow \Box Invariant$。要完成这步证明，我们需要给出目标成立依赖的事实。当证明目标不需要依赖额外事实，我们可以使用关键字 OBVIOUS 完成证明，即

⟨1⟩3. QED
  OBVIOUS

在这个案例中，目标成立依赖事实 ⟨1⟩1 – ⟨1⟩3。具体的证明过程为

⟨1⟩3. QED
  BY ⟨1⟩1, ⟨1⟩2, ⟨1⟩3, *PTL* DEF *Spec*

其中，BY 引入了事实 ⟨1⟩1 – ⟨1⟩3，而 *PTL*（Propositional Temporal Logic）是 TLA⁺ 中关于命题时序逻辑的库模块。*PTL* 的引入是为了处理 *Spec* 定义中的时序逻辑符 □。DEF 关键字的作用是将定义展开，即定理证明器会将 *Spec* 替换为 $Init \wedge \Box [Next]_{vars}$。

对于证明步骤 ⟨1⟩1 及 ⟨1⟩2，我们可以将它们视为独立的证明目标来编写证明。如 ⟨1⟩2 的证明过程为

⟨1⟩2. $Invariant \wedge [Next]_{vars} \implies Invariant'$
  ⟨2⟩1. CASE *Next*
  ⟨2⟩2. CASE UNCHANGED *vars*
  ⟨2⟩3. QED

$[Next]_{var}$ 定义为 $Next \vee$ UNCHANGED *vars*，因此这里使用了 CASE 关键字分情况讨论。并且这两种情况的证明 ⟨2⟩1、⟨2⟩2 仍然可以看做独立的目标进行证明。这样，TLAPS 整体上使用了层次化的结构，这种结构可以完成对证明目标的逐步拆解，达到化繁为简的效果。

⟨1⟩1 的证明过程比较简单，我们可以使用关键字 ASSUME，即

⟨2⟩1. ASSUME *Init* PROVE *Inv*
  OBVIOUS
⟨2⟩2. QED

ASSUME 关键字引入了 *Init* 事实，并且将证明目标 $Init \Rightarrow Invariant$ 转换为 *Invariant*。转换后的目标可以简单的使用 DEF 或者 OBVIOUS 关键字完成证明。

3.2 分布式共识

我们使用“三种角色模型”定义分布式共识问题[3][13]。提议者（Proposers）可以提出值（values），接受者（Acceptors）负责选择值，学习者（Learners）需要获悉被选中（chosen）的值。分布式共识问题的安全性（safety）需求包括：

- 非平凡性（Nontriviality）。只有被提出过的值才能被选中。
- 一致性（Consistency）。最多只有一个值被选中。

TLA+ 模块 *Consensus*[21] 抽象地描述了分布式共识问题。常量 *Value* 表示所有可能被提出的值构成的集合，它建模了提议者角色。变量 *chosen* 表示已被选中的值构成的集合，它建模了接受者角色。在初始状态下，*chosen* 为空。该规约只有一个可能的动作：如果 *chosen* 为空，则从 *Value* 中任选一个值 v，加入 *chosen* 中（即，选中 v）。不变式 *Inv* 表达了分布式共识问题的安全性，其中第三条合取式是一致性的定义，描述了 *chosen* 中的元素个数不超过 1，即最多只有一个值被选中。显然，规约 *Spec* 满足 *Inv*，即有 $THEOREM\ Spec \Rightarrow Inv$ 成立。

```
MODULE Consensus
EXTENDS Integers, FiniteSets, Sets

CONSTANT Value   被提出值的集合
VARIABLE chosen  被选中值的集合

Init ≜ chosen = {}
Next ≜
    ∧ chosen = {}
    ∧ ∃ v ∈ Value : chosen' = {v}
Spec ≜ Init ∧ □[Next]_chosen

Inv ≜
    ∧ chosen ⊆ Value
    ∧ IsFiniteSet(chosen)
    ∧ Cardinality(chosen) ≤ 1
Liveness ≜ ◇(chosen ≠ {})
```

除了安全性，分布式共识问题还包括活性，*Liveness* 表达了活性的具体定义，即最终会有某个值被选中。

3.3 Paxos 协议

Paxos 是在异步消息传递系统中解决分布式共识问题的重要协议。它采用传统的非拜占庭（non-Byzantine）故障模型[3]。在该模型中：

- 每个参与者（包括提议者、接受者与学习者）以各自的速度异步执行。参与者可能终止执行（fail by stopping），也可能重启（restart）。
- 消息可能会延时到达、重复或丢失。但是，消息不会被篡改。

在 Paxos 中，提议者为每个提议值 v 附加一个（全局唯一的）提议编号 b。我们用 $\langle b, v\rangle$，表示提议。接受者按规则接受提议。如果某个提议被“足够多”的接受者接受了，则称该提议以及它包含的值被选中。这里的“足够多”通常是指接受者中的任意一个多数派（majority）。因此，Paxos 可以容忍不超过半数的接受者失效。“多数派”是一种特殊的议会系统（quorum system）[41]。设集合 $Q \subseteq 2^A$ 是由接受者集合 A 的子集构成的集合。如果 Q 满足以下两个条件，则称 Q 为一个议会系统：

- Q 构成 A 的集合覆盖（set cover）：$\bigcup_{Q \in Q} Q = A$。
- 任意两个议会（quorum）相交不为空：$\forall Q_1, Q2 \in Q : Q_1 \cap Q_2 \neq \emptyset$。

Paxos 包含两个阶段，每个阶段由两个子阶段构成：

- 准备（Prepare）阶段（也称第一阶段 Phase1）
  - 子阶段 Phase1a：提议者选取一个提议编号 b，然后向所有接受者[①]发送编号为 b 的 Prepare 请求。
  - 子阶段 Phase1b：如果一个接受者收到编号为 b 的 Prepare 请求，并且编号 b 大于它之前回复过的所有 Prepare 请求的编号，那么它就将自己接受过的编号最大的提议（包括编号与值）回复给相应的提议者，并承诺不会再接受任何编号小于 b 的提议。
- 接受（Accept）阶段（也称第二阶段 Phase2）
  - 子阶段 Phase2a：如果提议者收到来自接受者中的某个议会对编号为 b 的 Prepare 请求的回复，那么它就向所有接受者发送一个针对提议 $\langle b, v\rangle$ 的 Accept 请求。其中，v 由收到的回复决定：如果回复中不包含任何提议，则 v 可以是任意值（特殊的，选择该提议者自己的值）；否则，v

[①]在原始 Paxos 协议描述中，只要求向接受者中的某个议会发送请求。为了叙述方便，本文改为向所有接受者发送请求。显然，该改动不影响协议的正确性。

是这些回复中编号最大的提议对应的值。

- 子阶段 Phase2b：如果一个接受者收到针对提议 $\langle b, v\rangle$，的 Accept 请求，只要它尚未对编号大于 b 的 Prepare 请求作过回复，它就可以接受该提议。

3.4 Paxos 协议的 TLA+ 规约

3.4.1 常量

Paxos 的 TLA+[21] 规约包含三个常量：

- *Value*：所有可能的提议值构成的集合（如，$\{v_1, v_2, v_3\}$）。我们定义 $None \triangleq CHOOSE\ v : v \notin Value$ 表示不属于 Value 的某个值。
- *Acceptor*：所有接受者构成的集合（如，$\{a_1, a_2, a_3\}$）。
- *Quorum*：由接受者形成的议会系统。*QuorumAssumption* 给出了议会系统需要满足的条件。

我们用自然数表示可能的提议编号，即 $Ballot \triangleq Nat$，并用 –1 表示相关变量初始化时用到的最小提议编号。

3.4.2 变量

Paxos 的 TLA+ 规约包含四个变量：

- *maxBal*：$maxBal[a]$ 表示接受者 a 在 Phase1b 与 Phase2b 两个子阶段见到过的最大提议编号，也是 a 承诺可以接受的提议的最小编号。
- *maxVBal* 与 *maxVVal*：$maxVBal[a]$ 表示接受者 a 接受过提议的最大编号，$maxVVal[a]$[①]是对应提议中的提议值。我们使用符号 $\langle maxVBal[a], maxVVal[a]\rangle$ 表示该提议。
- *msgs*：所有已发送消息构成的集合。参与者通过向该集合添加消息或者从该集合读取消息模拟消息的发送（广播）与接收动作。$Send(m)$ 将消息 m 加入到 *msgs* 中。为了对应 Paxos 协议的四个子阶段，该规约定义了四种消息类型，分别为“1a”、“1b”、“2a”与“2b”。

TypeOK 定义了各个变量的类型。*Init* 给出了初始状态下各个变量的取值。

[①]Paxos 的原始 TLA+ 规约中使用了变量名 *maxVal*。为了与 *maxVBal* 对应，本文改用 *maxVVal*。

MODULE *Paxos*

CONSTANT *Value*, *Acceptor*, *Quorum*

ASSUME *QuorumAssumption* $\triangleq$ $\land \forall Q \in Quorum : Q \subseteq Acceptor$
$\land \forall Q1, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

$Ballot \triangleq Nat$

$None \triangleq$ CHOOSE $v : v \notin Value$

$Message \triangleq [type : \{\text{“1a”}\}, bal : Ballot]$
$\cup [type : \{\text{“1b”}\}, acc : Acceptor, bal : Ballot,$
$mbal : Ballot \cup \{-1\}, mval : Value \cup \{None\}]$
$\cup [type : \{\text{“2a”}\}, bal : Ballot, val : Value]$
$\cup [type : \{\text{“2b”}\}, acc : Acceptor, bal : Ballot, val : Value]$

VARIABLE *maxBal*,
maxVBal, 二元组 $\langle maxVBal[a], maxVVal[a]$ 是 a 接受过的编号
maxVal, 最大的提议，如果 a 没有接受过任何提议，则为 $\langle -1, None\rangle$
msgs 已发送消息的集合。

$vars \triangleq \langle maxBal, maxVBal, maxVal, msgs\rangle$

$TypeOK \triangleq \land maxBal \in [Acceptor \to Ballot \cup \{-1\}]$
$\land maxVBal \in [Acceptor \to Ballot \cup \{-1\}]$
$\land maxVal \in [Acceptor \to Value \cup \{None\}]$
$\land msgs \subseteq Message$

$Init \triangleq \land maxBal = [a \in Acceptor \mapsto -1]$
$\land maxVBal = [a \in Acceptor \mapsto -1]$
$\land maxVal = [a \in Acceptor \mapsto None]$
$\land msgs = \{\}$

$Send(m) \triangleq msgs' = msgs \cup \{m\}$

3.4.3 动作

Paxos 规约定义了四种动作，分别对应于协议的四个子阶段：

- *Phase*1*a*(*b*)：提议者选取提议编号 *b*，发起 Prepare 请求。消息类型为 $m.type \mapsto$ “1*a*”，并携带提议编号 $m.bal \mapsto b$。
- *Phase*1*b*(*a*)：接受者 *a* 收到类型为“1*a*”的消息 *m*。如果 *m* 中携带的提议编号 *m.bal* 大于 *maxBal*[*a*]，则接受者 *a* 将 *maxBal*[*a*] 更新为 *m.bal*，并将它接受过的最大编号提议（即，$\langle maxVBal[a], maxVVal[a]\rangle$）回复给提议者。消息类型为“1*b*”，并携带收到的提议编号 *m.bal*。

- *Phase2a*(b, v)：提议者发起针对提议 $\langle b, v\rangle$，的 Accept 请求。该动作有两个前置条件：（1）第一个合取子句要求，针对提议编号 b，提议者尚未发起过 Accept 请求，即 *msgs* 中不存在类型为“2*a*”且编号为 b 的消息。（2）第

```
动作 Phase1a(b) 发送类型为“1a”的消息，携带的提议编号为 b。
Phase1a(b) ≜  ∧ Send([type ↦ “1a”, bal ↦ b])
              ∧ UNCHANGED ⟨maxBal, maxVBal, maxVal⟩
该动作更新 maxBal[a] 至 b 并且将提议 ⟨maxVBal[a], maxVVal[a]⟩ 回复给提议者。
Phase1b(a) ≜  ∧ ∃ m ∈ msgs :
              ∧ m.type = “1a”
              ∧ m.bal > maxBal[a]
              ∧ maxBal′ = [maxBal EXCEPT ![a] = m.bal]
              ∧ Send([type ↦ “1b”, acc ↦ a, bal ↦ m.bal,
                      mbal ↦ maxVBal[a], mval ↦ maxVal[a]])
              ∧ UNCHANGED ⟨maxVBal, maxVal⟩
该动作发送了类型为“2a”的消息，携带提议 ⟨b, v⟩，请求接受者接受该提议。
Phase2a(b, v) ≜
  ∧ ¬∃ m ∈ msgs  : m.type = “2a” ∧ m.bal = b
  ∧ ∃ Q ∈ Quorum :
      LET Q1b ≜ {m ∈ msgs : ∧ m.type = “1b”
                            ∧ m.acc ∈ Q
                            ∧ m.bal = b}
          Q1bv ≜ {m ∈ Q1b : m.mbal ≥ 0}
      IN  ∧ ∀ a ∈ Q : ∃ m ∈ Q1b : m.acc = a
          ∧ ∨ Q1bv = {}
            ∨ ∃ m ∈ Q1bv :
                ∧ m.mval = v
                ∧ ∀ mm ∈ Q1bv : m.mbal ≥ mm.mbal
  ∧ Send([type ↦ “2a”, bal ↦ b, val ↦ v])
  ∧ UNCHANGED ⟨maxBal, maxVBal, maxVal⟩
一旦接受者收到了“2a”消息 m，该接受者能执行该动作当且仅当 m.bal ≥ maxBal[a]。接受者将接受 m 携带的提议 ⟨b, v⟩，更新二元组 ⟨maxVBal[a], maxVVal[a]⟩ 至 ⟨b, v⟩，同时也会更新 maxBal[a] 至 b。
Phase2b(a) ≜  ∃ m ∈ msgs : ∧ m.type = “2a”
                           ∧ m.bal ≥ maxBal[a]
                           ∧ maxBal′ = [maxBal EXCEPT ![a] = m.bal]
                           ∧ maxVBal′ = [maxVBal EXCEPT ![a] = m.bal]
                           ∧ maxVal′ = [maxVal EXCEPT ![a] = m.val]
                           ∧ Send([type ↦ “2b”, acc ↦ a,
                                   bal ↦ m.bal, val ↦ m.val])
```

二个合取子句要求，存在接受者中的某个议会 Q，Q 中的每个接受者都回复过针对提议编号 b 的 Prepare 请求。前置条件成立时，提议者便根据 Q 中

每个接受者回复的消息确定 b 值（方法如 Paxos 协议的 Phase2a 子阶段所述），然后通过类型为“2a”的消息发起针对提议 $\langle b, v\rangle$ 的 Accept 请求。

- *Phase2b*(a)：接受者 a 接受某个提议。该动作的前置条件是存在类型为“2a”的消息 m，并且 m 中携带的提议编号 $m.bal$ 大于等于 $maxBal[a]$。前置条件成立时，接受者 a 将 $maxBal[a]$ 更新为 $m.bal$[①]，并且接受消息 m 中携带的提议 $\langle m.bal, m.val\rangle$，然后将该提议封装在类型为“2b”的消息中发送出去。

3.4.4 行为

Next 定义了次态关系。*Spec* 定义了完整的行为规约：

$$
\begin{aligned}
&Next \triangleq \lor \exists b \in Ballot : \lor Phase1a(b) \\
&\qquad\qquad\qquad\qquad \lor \exists v \in Value : Phase2a(b, v) \\
&\qquad \lor \exists a \in Acceptor : Phase1b(a) \lor Phase2b(a) \\
&Spec \triangleq Init \land \Box[Next]_{vars}
\end{aligned}
$$

3.5 Paxos 的精化

Paxos 协议的核心在于接受者何时可以接受提议，以及这些提议需要满足何种条件。一方面，接受者 a 只有在提议编号 b 大于等于它回复过的最大提议编号 $maxBal[a]$（也是它承诺可以接受的提议的最小编号）时，它才可能接受该提议。另一方面，可以被接受的提议 $\langle b, v\rangle$，需要满足两个关键条件：一是，每个提议编号（这里的 b）最多对应一个提议（这里的 $\langle b, v\rangle$）。该条件称为 *OneValuePerBallot*。二是，$\langle b, v\rangle$，是安全的，即对于任何提议 $\langle b', v'\rangle'$，其中 $b' \in 0 \ldots b-1, v' \neq v$ 都尚未被选中且将来也不会被选中。该条件称为 $SafeAt(b, v)$。Lamport 使用了 TLAPS 证明了[42]Paxos 协议正确性关键在于上述两个条件，即只要满足这两个条件，就能够保证 Paxos 的正确性。

为了更好地理解上述核心问题，Lamport 提出了一个更抽象的规约，记为 *Voting*[43]。在 *Voting* 中，每个接受者 a 仍然维护 $maxBal[a]$。此外，它还维护它曾接受过的所有提议 *votes*。该规约包含两个动作：在 $IncreaseMaxBal(a, b)$

[①]需要注意的是，自然语言描述的 Paxos 协议没有要求更新 $maxBal[a]$。然而，可以通过反例说明，如果不同时更新 $maxBal[a]$，则 Paxos 并不能保证论文中的不变式 P2c 成立。

中，接受者 a 可以提高自己的 $maxBal[a]$。该动作对应于 $Phase1a$ 与 $Phase1b$ 子阶段。在 $VoteFor(a,b,v)$ 中，接受者 a 接受提议 $\langle b,v\rangle$。其中，第二与第三个合取式保证了 $OneValuePerBallot$ 条件。第四个合取式保证了 $SafeAt(b,v)$ 条件。该动作对应于 $Phase2a$ 与 $Phase2b$ 子阶段。

```
MODULE Voting
EXTENDS Integers, FiniteSets, TLAPS

CONSTANT Value, Acceptor, Quorum
Ballot ≜ Nat
VARIABLES votes, maxBal
TypeOK ≜ ∧ votes ∈ [Acceptor → SUBSET (Ballot × Value)]
         ∧ maxBal ∈ [Acceptor → Ballot ∪ {−1}]

Init ≜
    ∧ votes = [a ∈ Acceptor ↦ {}]
    ∧ maxBal = [a ∈ Acceptor ↦ −1]
IncreaseMaxBal(a, b) ≜
    ∧ b > maxBal[a]
    ∧ maxBal′ = [maxBal EXCEPT ![a] = b]   作承诺
    ∧ UNCHANGED votes
VoteFor(a, b, v) ≜
    ∧ maxBal[a] ≤ b   遵守承诺
    ∧ ∀ vt ∈ votes[a] : vt[1] ≠ b
    ∧ ∀ c ∈ Acceptor \ {a} :
        ∀ vt ∈ votes[c] : (vt[1] = b) ⟹ (vt[2] = v)
    ∧ ∃ Q ∈ Quorum : ShowsSafeAt(Q, b, v)   满足 SafeAt 性质
    ∧ votes′ = [votes EXCEPT ![a] = votes[a] ∪ {⟨b, v⟩}]   投票
    ∧ maxBal′ = [maxBal EXCEPT ![a] = b]   作承诺

Next ≜
    ∃ a ∈ Acceptor, b ∈ Ballot :
      ∨ IncreaseMaxBal(a, b)
      ∨ ∃ v ∈ Value : VoteFor(a, b, v)
Spec ≜ Init ∧ □[Next]⟨votes, maxBal⟩
```

由于 $VoteFor(a,b,v)$ 使用了所有被接受过的提议这一全局信息，$Voting$ 可以看作分布式共识问题的集中式解决方案，而 $Paxos$ 是 $Voting$ 的分布式实现。实际上，Lamport 构建了从 $Paxos$ 到 $Voting$ 以及从 $Voting$ 到 $Consensus$ 的精化关系[21]。

3.6 小结

本章介绍了形式化语言 TLA$^+$ 与定理证明系统 TLAPS，并且给了简单的使用示例。接着，我们介绍了分布式共识问题，它是分布式计算领域的核心问题。我们详细描述了解决分布式共识问题的经典协议 Paxos，并且展示了它的 TLA$^+$ 规约。针对 Paxos 协议的正确性，我们介绍了其核心的抽象 Voting 规约。事实上，Lamport 不仅构建了从 Paxos 到 Voting 的精化关系，还给出了 Paxos 协议的 TLAPS 证明，我们将在第六章复用这个证明框架。

第四章　TPaxos 的推导及规约

本章首先介绍 TPaxos 协议，给出了如何从经典 Paxos 协议推导出 TPaxos 协议。然后，使用 TLA$^+$ 描述 TPaxos。我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处：在消息处理阶段，参与者是先作出“不再接受具有更小编号的提议”的承诺还是先接受提议？这可能导致对 TPaxos 的两种不同理解，并促使我们提出 TPaxos 的一种变体，称为 TPaxosAP。在 TPaxosAP 中，参与者先接受提议后作承诺。虽然在 TLA$^+$ 规约层面，TPaxosAP 与 TPaxos（先作承诺后接受提议）相差甚少，但它体现了不同的投票机制。具体来讲，Paxos 协议的投票机制 Voting 仍适用于 TPaxos 却不能完整刻画 TPaxosAP 的行为。为此，我们将举例说明 TPaxosAP 与 TPaxos 的不同之处，并论证 TPaxosAP 的正确性。

4.1　TPaxos 协议及其推导

本质上，TPaxos 协议也是 Paxos 协议的一种变体。为了刻画这两者之间的变体关系，我们首先详细介绍 TPaxos 协议，总结 TPaxos 协议与 Paxos 的异同。然后，使用演绎推理的模式论述如何从经典 Paxos 协议推导出 TPaxos 协议。

4.1.1　TPaxos 协议

在 TPaxos 协议中，所有参与者（即，数据副本节点；假设有 N 个）都同时具有提议者、接受者与学习者三种角色。TPaxos 协议的新颖之处在于它的“统一性”：它为每个参与者维护统一的状态类型，并采用统一格式的消息进行通信。相比而言，Paxos 及其众多变体为提议者与接受者维护不同的状态类型（即，维护不同的变量），并且需要四种消息类型来区分协议的四个子阶段。下面，我们先分别介绍统一的状态类型与消息类型，然后介绍具体协议过程。

在 TPaxos 中，每个提议 P 是一个二元组 $\langle b, v\rangle$，其中 b 表示提议编号，v 表示提议值。每个参与者 p 维护一个大小为 N 的状态向量 S^p。其中参与者 q 对应的向量值，记为 S_q^p，表示 p 对 q 的观察状态（*view state*）。特殊地，如果 $p = q$，则称 S_p^p 为 p 的当前实际状态（*acutal state*）。每个状态 S_q^p 记为 $(m, p)m$ 表示 q 承诺可以接受提议的最小编号；P 表示 q 最近一次接受的提议 $\langle b, v\rangle$。我们使用 $S_q^p.m$、$S_q^p.P$、$S_q^p.P.b$ 与 $S_q^p.P.v$ 访问相应的状态分量。

在 TPaxos 中，参与者 p 发送给 q 的统一消息类型可记为 $M_{p\to q} = \left(S_p^p, S_q^p\right)$。也就是说，$p$ 会将自身的实际状态 S_p^p 以及它对 q 的观察状态 S_q^p 一起发送给 q。

参与者之间通过相互交换此类消息不断更新状态，并在本地判断是否有值已被选中，从而最终达成共识。具体而言，TPaxos 的执行过程与 Paxos 类似，也分为两个阶段（伪代码见 Algorithm 1）。

Algorithm 1: TPaxos in PaxosStore

```
1  Procedure Prepare(b)
2      if S_p^p.m < b then
3          S_p^p.m ← b
4          foreach remote replica node q do
5              send M_{p→q}
6  Procedure Accept(P_i)
7      if |{∀S_q^p ∈ S^p | S_q^p.m = P_i.b}|×2 > |S^p| then
8          if |{∀S_q^p ∈ S^p | S_q^p.P.v ≠ null}|> 0 then
9              P' ← the proposal with maximum P.b in
                 S^p
10             S_p^p.P ← (P_i.b, P'.v)
11         else
12             S_p^p.P ← P_i
13         foreach remote replica node q do
14             send M_{p→q}
15 Procedure OnMessage(M_{p→q})
16     UpdateStates(q, S_p^p)
17     if S_q^p.m < S_q^q.m or S_q^p.P.b < S_q^q.P.b then
18         send M_{q→p}
19     if S_q^q is changed then
20         if IsValueChosen(q) is true then commit
21 Function UpdateStates(q, S_p^p)
22     if S_p^q.m < S_p^p.m then S_p^q.m ← S_p^p.m
23     if S_p^q.P.b < S_p^p.P.b then S_p^q.P ← S_p^p.P
24
25     if S_q^q.m < S_p^q.m then S_q^q.m ← S_p^p.m
26     if S_q^q.m ≤ S_p^p.P.b then S_q^q.P ← S_p^p.P
27 Function IsValueChosen(q)
28     n' ←occurrence count of the most frequent P.b
29     in S^q
30     return n' × 2 > |S^q|
```

- 准备阶段
  - Phase1a 子阶段：参与者 p 选取一个大于 $S_p^p.m$ 的提议编号 b，更新 $S_p^p.m$ 至 b，然后向其他所有参与者 $q \neq p$ 发送消息 $M_{p\to q} = \left(S_p^p, S_q^p\right)$（即，Prepare 请求）。
  - 消息处理（OnMessage）子阶段：参与者 q 收到并处理来自 p 的消息 $M_{p\to q} = \left(S_p^p, S_q^p\right)$。
    - 首先，q 根据 S_p^p 更新本地状态 S_p^q 与 S_q^q。此处重点关注 S_q^q 的更新：

      1） 作承诺：如果 $S_q^q.m < S_p^p.m$，则将 $S_q^q.m$ 更新为 $S_p^p.m$。这模拟了

Paxos 中的 Phase1b 阶段。

2) 接受提议：如果 $S_q^q.m < S_p^p.P.b$，则将 $S_q^q.P$ 更新为 $S_p^p.P$。这模拟了 Paxos 中的 Phase2b 阶段。

消息处理阶段（的核心）是 Phase1b 与 Phase2b 子阶段的合并。需要注意的是，根据 TPaxos 的伪代码，q 先作承诺后接受提议[①]。第4章将讨论另一种方案，即 q 先接受提议后作承诺。

- 然后，q 根据本地状态向量 S^q 判断是否有值已被选中；见 $IsValueChosen(q)$。
- 最后，如果 S_q^p 比更新后的 S_q^q 状态要旧，那么 q 向 p 发送消息 $M_{q\to p} = \left(S_q^q, S_p^q\right)$。

- 接受阶段
  - Phase2a 子阶段：参与者 p 根据本地状态向量 S^p 判断是否可以发起针对提议 $P = \langle b, v\rangle$ 的 Accept 请求。它要求存在某个议会 Q，使得对于 Q 中的每个参与者 q，都满足 $S_q^p.m = b$。这表明，p 观察到 Q 中每个参与者都回复了编号为 b 的 Prepare 请求。在这种情况下，p 将根据 S^p 确定提议值 v：如果对于所有的参与者 q，都满足 $S_q^p.P.b = -1$（即，p 未观察到某参与者接受过任何提议），则 v 可以为任意值；否则，v 是所有 $S_q^p.P$ 中最大提议编号对应的提议值。然后，p 更新本地状态 $S_p^p.P$ 为 $\langle b, v\rangle$，并向其他所有参与者 $q \neq p$ 发送消息 $M_{p\to q} = \left(S_p^p, S_q^p\right)$（即，Accept 请求）。

    在该阶段，有两点需要特别注意：第一，在 TPaxos 中，p 根据它对所有参与者的观察状态确定 v，而不仅限于议会 Q 中的参与者。第 5章将讨论如何处理这种差别。第二，p 在确定 v 之后，立即将本地状态 $S_p^p.P$ 设置为 $\langle b, v\rangle$。第4章将论述它可能带来的问题。
  - 消息处理子阶段：与准备阶段中的消息处理子阶段相同。

4.1.2 从 Paxos 推导 TPaxos

如前所述，TPaxos 采用统一的状态与统一的消息类型。这有助于精简而高效的工程实现。但是，这个特点也带来了 TPaxos 与 Paxos 之间的诸多差异，给理解 TPaxos 造成了障碍。本小节分四个步骤论证如何从 Paxos 推导出 TPaxos，

[①]此时，第 2）步 $S_q^q.m < S_p^p.P.b$ 中的 $S_q^q.m$ 可能已被更新。

使得 TPaxos 可看作 Paxos 的一种自然变体，更易于理解。

I. **统一状态类型**。TPaxos 之所以能够采用统一的状态类型，是因为在 TPaxos 中，每个参与者都同时具有提议者、接受者与学习者的角色。一方面，由于参与者 p 既是提议者也是接受者，所以 p 需要维护状态 (m, P)，其中，m 是 p 承诺可以接受提议的最小编号，$P = \langle b, \rangle v$ 是 p 最近一次接受的提议。另一方面，由于 p 也是学习者，它需要在本地判断是否有值/提议已被选中，所以 p 还需要维护它对其他所有参与者的观察状态。综上所述，在 TPaxos 中，每个参与者 p 需要维护一个大小为 N 的状态向量，即 S^p。

II. **统一消息内容**。在状态类型统一的基础上，我们进一步要求，每次 p 与 q 通信时，p 将它的真实状态 S_p^p 发送出去。需要注意的是，该步骤并未统一消息类型。这是因为，协议还需要在消息中添加类似“1a”，“2a”等标志消息，以区分多个子阶段。第III步推导将论述如何统一消息类型。另外，这步推导不要求发送 p 对 q 的观察状态 S_q^p。TPaxos 采用的消息类型 $M_{p\to q} = \left(S_p^p, S_q^p\right)$ 中的 S_q^p 的作用将在第IV推导中论述。

III. **统一消息类型**。Paxos 使用了消息类型以区分四个子阶段。为了统一这四种消息类型，我们先论证如何统一某些子阶段，然后讨论如何避免通过消息类型区分子阶段。

a. **将子阶段 Phase1b 与 Phase2b 统一为消息处理阶段。**

- 一方面，在 *Paxos* 规约的 *Phase2b*(a) 动作中，接受者 a 在接受类型为“2a”并且携带提议 $\langle m.bal, m.val\rangle$ 的消息 m 是，不仅更新了 $\langle maxVBal[a], maxVVal[a]\rangle$，也同时将 $maxBal[a]$ 更新为 $m.bal$（条件 $m.bal \geq maxBal[a]$ 成立）。也就是说，接受者 a 在接受某提议时，可以同时作出“不再接受具有更小编号的提议”的承诺。
- 另一方面，经过第II步的推导，接受者 q 收到的来自 p 的信息 S_p^p 不仅包含提议编号 m，还包含 p 最近接受的提议 $P = \langle b, v\rangle$。根据 Paxos 的 Phase1b 阶段，如果条件 $S_q^q.m < S_p^p.m$ 成立，q 就将 $S_q^q.m$ 更新为 $S_p^p.m$。然而，我们注意到，如果条件 $S_q^q.m < S_p^p.P.b$ 成立[①]，那么 q 也可以接受提议 $S_p^p.P$。

综上所述，在 Phase1b 与 Phase2b 子阶段，接受者都可以（在各自条件成立的情况下）既作承诺又接受提议。因此，我们可以将它们统一为一个阶段，即消息处理阶段。需要注意的是，由于在第II步推导中，p 不

[①]注意，$S_q^q.m$ 可能已被更新。

需要发送它对 q 的观察状态 S_q^p，因此在第II步推导得出的消息处理阶段只包含对 S_q^q 的更新，而不包含对 S_p^q 的更新。此外，q 需要将 S_q^q 回复给 p。与 TPaxos 最终版本不同的是，该回复是无条件执行的。这将在第IV步推导中论述。

b. **消息处理标志**。经过前面的推导，目前我们得到的协议包含三个阶段，分别是准备阶段、接受阶段与消息处理阶段。为了区分各个阶段，（统一内容的）消息中仍需携带标志信息。下面，我们说明这些标志信息是可以避免的。

- 首先，在 Paxos 中，接受者通过判断收到的消息类型是“1a”还是“2a”来确定进入 Phase1b 还是 Phase2b 子阶段。由于在第 III-a步推导中，Phase1b 与 Phase2b 被统一成消息处理阶段，所以不需要再区分消息类型“1a”与“2a”。
- 其次，在 Paxos 的 $Phase2a(b, v)$ 阶段，提议者需要收集类型为“1b”且携带提议编号 b 的消息。它们是由接受者发送的针对编号为 b 的 Prepare 请求的回复。在 TPaxos 中，参与者 p 可以通过 $S_q^p.m = b$ 是否成立判断 q 是否发送了这样的回复。
- 最后，在 Paxos 中，接受者使用类型为“2b”的消息将它最新接受的提议通知给学习者。由于在 TPaxos 中，参与者同时具有三种橘色，因此不再需要发送该类消息。参与者可以根据本地状态向量判断是否有提议已被选中。

IV. **优化消息处理阶段**。在第III-a步推导中，参与者在消息处理阶段会将自身真实状态无条件回复给消息的发送者。这导致任意两个参与者之间都会无休止地相互发送消息，交换各自的状态信息。为了避免这种情况，TPaxos 作了如下优化：首先，p 发送给 q 的消息会携带 p 对 q 的观察状态。也就是说，统一的消息类型变为 $M_{p\to q} = \left(S_p^p, S_q^p\right)$。其次，在 q 的消息处理阶段，只有当观察状态 S_q^p 比更新后的 S_q^q 状态要旧时，q 才回复消息 $M_{q\to p} = \left(S_q^q, S_p^q\right)$ 给 p。

4.2 TPaxos 与 TPaxosAP 的 TLA+ 规约

本节使用 TLA+ 规约描述 TPaxos 与 TPaxosAP 协议。针对 TPaxosAP，我们会论述 TPaxosAP 与 TPaxos 的不同之处，并且给出其正确性的证明。

4.2.1 TPaxos 的 TLA+ 规约

4.2.1.1 常量

与 Paxos 对应，TPaxos 的 TLA+ 规约也包含三个常量：

- *Value*：所有可能的提议值构成的集合（例如，$\{v_1, v_2, v_3\}$。*None* 表示不属于 *Value* 的某个值。
- *Participant*：所有参与者构成的集合（例如，$\{p_1, p_2, p_3\}$）。每个参与者都同时是提议者、接受者与学习者。
- *Quorum*：由参与者形成的议会系统。

为了避免多个参与者使用相同的提议编号发起多个 Accept 请求，我们使用 *Bals*(*p*) 为参与者预分配可用的提议编号（如，*p*1 使用 $\{1,4\}$，*p*2 使用 $\{2,5\}$，*p*3 使用 $\{3,6\}$）[①]。其中，*Ballot* 是所有可能的提议编号构成的集合，*NP* 是参与者数目，*PIndex* 为参与者分配了索引。

4.2.1.2 变量

在 TPaxos 中，每个参与者维护了一个状态向量，包含自身的实际状态以及它对其他参与者的观察状态。参与者之间交换具有统一类型的消息，不断更新状态向量，进而达成共识。因此，TPaxos 的 TLA+ 规约需要维护如下变量：

- *state*：全局状态矩阵。其中，*state*[*p*][*q*] 表示参与者 *p* 对参与者 *q* 的观察状态（即 S_q^p）。*state*[*p*] 则表示 *p* 维护的状态向量（即 S^p）。每个状态（定义见 *State*）包含三个分量 $(maxBal, maxVBal, maxVVal)$，含义与第 4.1 节中的 (m, b, v) 一一对应。
- *msgs*：所有已发送消息构成的集合。*Message* 表示所有可能的消息，它定义了格式统一的消息。需要注意的是，为了简化消息广播过程，我们选择每次发送整个状态向量。消息接受者可以根据需要提取协议规定的部分状态信息进行处理。

TypeOK 给出了变量 *state* 与 *msgs* 的类型约束。*Init* 给出了初始状态下变量 *state* 与 *msgs* 的值。

[①] *Paxos* 规约为采用预分配提议编号的方式。这有两个原因：第一，在 *Phase*1*a*(*b*) 与 *Phase*2*a*(*b*, *v*) 阶段，提议者并不改变自身状态，所以可由任意提议者执行。第二，它在 *Phase*2*a*(*b*, *v*) 中通过消息类型限制了最多只会产生一个针对提议编号 *b* 的 *Accept* 请求。

MODULE *TPaxos*

EXTENDS *Integers*, *FiniteSets*

CONSTANTS

Participant, 参与者集合

Value 参与者提出的值集合

$None \triangleq \text{CHOOSE } b : b \notin Value$

$NP \triangleq Cardinality(Participant)$ 参与者数量

$Quorum \triangleq \{Q \in \text{SUBSET } Participant : Cardinality(Q) * 2 \geq NP + 1\}$

$Ballot \triangleq Nat$

$Max(m, n) \triangleq \text{IF } m > n \text{ THEN } m \text{ ELSE } n$

$Injective(f) \triangleq \forall a, b \in \text{DOMAIN } f : (a \neq b) \implies (f[a] \neq f[b])$

$MaxBallot \triangleq Cardinality(Ballot) - 1$

$PIndex \triangleq \text{CHOOSE } f \in [Participant \to 1 \,..\, NP] : Injective(f)$

$Bals(p) \triangleq \{b \in Ballot : b\%NP = PIndex[p] - 1\}$ 将编号 b 分配给每个参与者

$State \triangleq [maxBal : Ballot \cup \{-1\},$
$\quad maxVBal : Ballot \cup \{-1\}, maxVVal : Value \cup \{None\}]$

$InitState \triangleq [maxBal \mapsto -1, maxVBal \mapsto -1, maxVVal \mapsto None]$

$Message \triangleq [from : Participant,$
$\quad to : \text{SUBSET } Participant,$
$\quad state : [Participant \to State]]$

VARIABLES

state, *state*[*p*][*q*]: 参与者 *p* 对参与者 *q* 的观察状态

msgs 所有发送消息的集合

$vars \triangleq \langle state, msgs \rangle$

$TypeOK \triangleq$

$\quad \wedge\ state \in [Participant \to [Participant \to State]]$

$\quad \wedge\ msgs \subseteq Message$

$Init \triangleq$

$\quad \wedge\ state = [p \in Participant \mapsto [q \in Participant \mapsto InitState]]$

$\quad \wedge\ msgs = \{\}$

$Send(m) \triangleq msgs' = msgs \cup \{m\}$

4.2.1.3 动作

TPaxos 包含三个动作，分别是发起准备请求、消息处理以及发起接受请求。

- *Prepare*(p, b)：参与者 p 选取提议编号 b，发起 Prepare 请求。它要求 b 是预分配给 p 的编号，并且 $b > state[p][p].maxBal$。p 先将 $state[p][p].maxBal$ 更新为 b，然后广播当前本地状态向量 $state'[p]$。

参与者 p 选择一个大于 $state[p][p].maxBal$ 的编号值 b 开始 Prepare 阶段

$$
\begin{aligned}
&Prepare(p, b) \triangleq \\
&\quad \land b \in Bals(p) \\
&\quad \land state[p][p].maxBal < b \\
&\quad \land state' = [state \text{ EXCEPT } ![p][p].maxBal = b] \\
&\quad \land Send([from \mapsto p,\ to \mapsto Participant \setminus \{p\},\ state \mapsto state'[p]])
\end{aligned}
$$

- *OnMessage*(q)：参与者 q 处理收到的消息 $m \in msgs$。消息 m 的发送者记为

参与者 q 收到来自 p 的消息，根据消息中携带的 pp 信息来更新自身状态 $state[q]$，该函数被函数 *OnMessage*(q) 调用。（$pp = m.state[p]$，pp 可能不等于此时 p 的真实状态 $state[p][p]$）

$$
\begin{aligned}
&UpdateState(q,\ p,\ pp) \triangleq \\
&\quad \text{LET } maxB \triangleq Max(state[q][q].maxBal,\ pp.maxBal) \\
&\quad \text{IN} \quad state' = [state \text{ EXCEPT} \\
&\qquad\qquad ![q][p].maxBal = Max(@,\ pp.maxBal), \\
&\qquad\qquad ![q][p].maxVBal = Max(@,\ pp.maxVBal), \\
&\qquad\qquad ![q][p].maxVVal = \text{IF } state[q][p].maxVBal < pp.maxVBal \\
&\qquad\qquad\qquad\qquad \text{THEN } pp.maxVVal \text{ ELSE } @, \\
&\qquad\qquad ![q][q].maxBal = maxB, \quad \text{先承诺后接受} \\
&\qquad\qquad ![q][q].maxVBal = \text{IF } maxB \leq pp.maxVBal \quad \text{接受} \\
&\qquad\qquad\qquad\qquad \text{THEN } pp.maxVBal \text{ ELSE } @, \\
&\qquad\qquad ![q][q].maxVVal = \text{IF } maxB \leq pp.maxVBal \quad \text{接受} \\
&\qquad\qquad\qquad\qquad \text{THEN } pp.maxVVal \text{ ELSE } @]
\end{aligned}
$$

参与者 q 收到消息并处理

$$
\begin{aligned}
&OnMessage(q) \triangleq \\
&\quad \exists m \in msgs : \land q \in m.to \\
&\qquad \land \text{LET } p \triangleq m.from \text{ IN} \quad UpdateState(q,\ p,\ m.state[p]) \\
&\qquad \land \text{LET } qm \triangleq [from \mapsto m.from,\ to \mapsto m.to \setminus \{q\},\ state \mapsto m.state] \\
&\qquad\qquad\quad nm \triangleq [from \mapsto q,\ to \mapsto \{m.from\},\ state \mapsto state'[q]] \quad \text{新消息} \\
&\qquad\quad \text{IN} \quad \text{IF } \lor m.state[q].maxBal < state'[q][q].maxBal \\
&\qquad\qquad\qquad\qquad \lor m.state[q].maxVBal < state'[q][q].maxVBal \\
&\qquad\qquad\qquad\quad \text{THEN } msgs' = (msgs \setminus \{m\}) \cup \{qm,\ nm\} \\
&\qquad\qquad\qquad\quad \text{ELSE } msgs' = (msgs \setminus \{m\}) \cup \{qm\}
\end{aligned}
$$

$p \triangleq m.from$。表达式 $m.state[p]$ 从消息 m 中抽取 p 发送的自身真实状态。q 先根据 $m.state[p]$ 更新本地状态向量，包括 q 的真实状态 $state[q][q]$ 以及 q 对 p 的观察状态 $state[q][p]$；见 $UpdateState(q, p, pp \triangleq m.state[p])$。其中，前三条更新观察状态，后三条更新真实状态，如第4.1节所述，包括“先作承诺（将 $state[q][q].maxBal$ 更新为 $pp.maxBal$）后接受（分别将 $state[q][q].maxVBal$ 与 $state[q][q].maxVVal$ 更新为 $pp.maxVBal$ 与 $pp.maxVVal$）”两个步骤。然后，如果 p 对 q 的观察状态 $m.state[q]$ 比更新后的 $state'[q][q]$ 要旧，则 q 广播当前本地状态向量 $state'[q]$。最后，为了避免 q 不必要地重复处理消息 m，该规约将 q 从 $m.to$ 中移除。

- $Accept(p, b, v)$：参与者 p 发起针对提议 $\langle b, v\rangle$ 的 Accept 请求。
  - 第一个前置条件要求提议编号 b 是参与者 p 的预分配编号，从而避免多个参与者使用相同的提议编号 b 发起多个 Accept 请求。

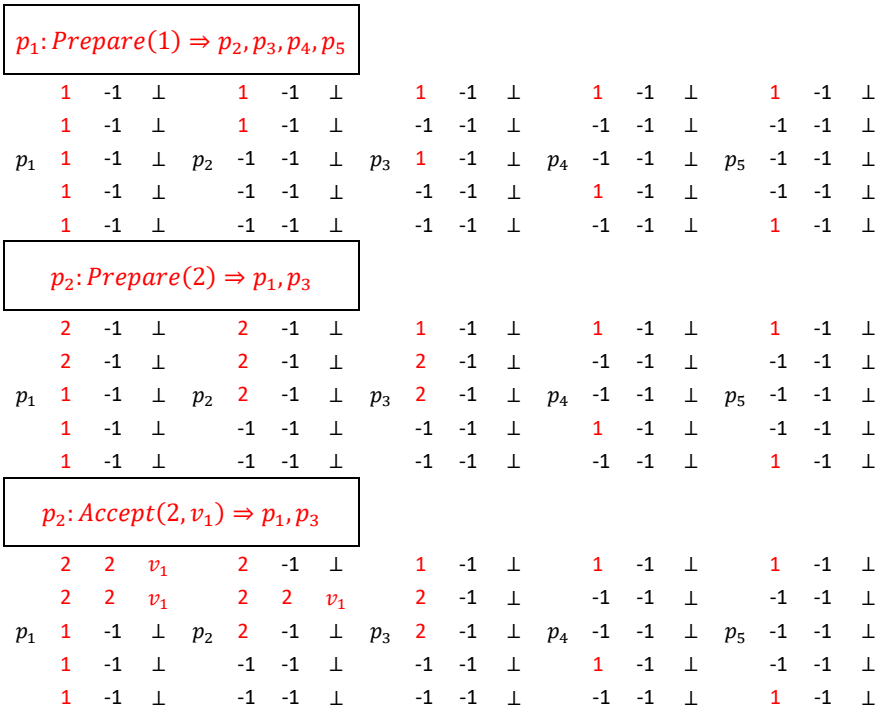

图 4-1: 针对 $Accept(p, b, v)$ 动作的第二个前置条件的示例

  - 第二个前置条件避免参与者接受违背其承诺的提议。下面，我们通过示例论述该条件的必要性。假设有五个参与者 $p1$、$p2$、$p3$、$p4$ 与 $p5$；见图4-1。首先，$p1$ 执行 $Prepare(1)$。$p2$、$p3$、$p4$ 与 $p5$ 收到该 Prepare 请求后更新状态并回复 $p1$。此时所有参与者的状态如图4-1第一行所示。

接着，$p2$ 执行 $Prepare(2)$。$p1$ 与 $p3$ 收到该 Prepare 请求后更新状态并回复 $p2$；见图 4-1第二行。然后，$p2$ 执行 $Accept(2, v_1)$。$p1$ 收到该 Accept 请求并更新状态；见图 4-1第三行。此时，$p1$ 观察到议会 $p3$、$p4$、$p5$，它们的 $maxBal$ 为 1. 如果没有前置条件 $2 = state[p][p].maxBal \leq b = 1$，则 $p1$ 可以执行 $Accept(1, v_1)$，并接受提议 $\langle 1, v_1 \rangle$。这违背了它“不再接受编号小于 $state[p][p].maxBal = 2$ 的提议”的承诺。

- 前三个前置条件共同保证 $OneValuePerBallot$ 成立，证明见第 4.2.3节。其中，如果第三个前置条件不成立，即 $state[p][p].maxVBal = b$，则表示 p 已经执行过一个一个 $Accept(p, b, v)$。然而，仅有第三个前置条件并不能避免 p 多次执行 $Accept(p, b, v)$。这是因为，p 可能会在之后的 $OnMessage$ 动作中接受具有更大编号的提议，从而使得 $state[p][p].maxVBal > b$。
- 第四个前置条件要求存在某个议会 $Q \in Quorum$，使得 Q 中的每个参与者 q 都回复了针对 b 的 Prepare 请求，即有 $state[p][q].maxBal = b$。
- 第五条合取式描述了根据本地状态向量 $state[p]$ 确定值 v 的方法。第六条合取式更新 p 的真实状态，即 p 接受了提议 $\langle b, v \rangle$。第七条合取式表示向其它参与者广播本地状态向量 $state'[p]$。

参与者 p 选择提议编号 b（b 应该等于 Prepare 阶段的 b）和值 v，v 的选择和 Paxos 的选择方式是相同的。

$$
\begin{aligned}
&Accept(p, b, v) \triangleq \\
&\quad \land b \in Bals(p) \\
&\quad \land state[p][p].maxBal \leq b \quad \text{对应于 } Voting \text{ 中的第一条合取式} \\
&\quad \land state[p][p].maxVBal \neq b \quad \text{对应于 } Voting \text{ 中的第二条合取式} \\
&\quad \land \exists Q \in Quorum : \\
&\qquad \forall q \in Q : state[p][q].maxBal = b \\
&\quad \text{从本地所有状态选值} \\
&\quad \land \lor \forall q \in Participant : state[p][q].maxVBal = -1 \\
&\qquad \lor \exists q \in Participant : \quad v \text{ 是提议编号最大的提议对应的值} \\
&\qquad\quad \land state[p][q].maxVVal = v \\
&\qquad\quad \land \forall r \in Participant : \\
&\qquad\qquad state[p][q].maxVBal \geq state[p][r].maxVBal \\
&\quad \land state' = [state \text{ EXCEPT } ![p][p].maxVBal = b, \\
&\qquad\qquad\qquad\qquad ![p][p].maxVVal = v] \\
&\quad \land Send([from \mapsto p, to \mapsto Participant \setminus \{p\}, state \mapsto state'[p]])
\end{aligned}
$$

4.2.1.4 行为

Next 定义了次态关系。*Spec* 定义了完整的行为规约:

$$
\begin{aligned}
&Next \triangleq \exists p \in Participant : \\
&\qquad \lor OnMessage(p) \\
&\qquad \lor \exists b \in Ballot : \lor Prepare(p, b) \\
&\qquad\qquad \lor \exists v \in Value : Accept(p, b, v) \\
&Spec \triangleq Init \land \Box[Next]_{vars}
\end{aligned}
$$

4.2.2 TPaxosAP 的 TLA+ 规约

TPaxosAP 与 TPaxos 的唯一不同在于，在消息处理阶段 $OnMessage(q)$ 的 $UpdateState(q, p, pp)$ 中，参与者 q 先接受提议 $\langle pp.maxVBal, pp.maxVVal\rangle$ 后作承诺。需要注意的是, 在 TLA+ 规约中，判断 q 能否接受提议的条件 $state[q][q].maxBal \leq pp.maxVBal$ 中使用的是当前值 $state[q][q].maxBal$，而不是像 TPaxos 那样使用了先作承诺而更新后的 *maxB*。

$$
\begin{aligned}
&UpdateState(q, p, pp) \triangleq \\
&\quad state' = [state \text{ EXCEPT} \\
&\qquad ![q][p].maxBal = Max(@, pp.maxBal), \\
&\qquad ![q][p].maxVBal = Max(@, pp.maxVBal), \\
&\qquad ![q][p].maxVVal = \text{IF } state[q][p].maxVBal < pp.maxVBal \\
&\qquad\qquad \text{THEN } pp.maxVVal \text{ ELSE } @, \\
&\qquad ![q][q].maxBal = Max(@, pp.maxBal), \quad \text{作承诺} \\
&\qquad ![q][q].maxVBal = \text{IF } state[q][q].maxBal \leq pp.maxVBal \quad \text{接受} \\
&\qquad\qquad \text{THEN } pp.maxVBal \text{ ELSE } @, \\
&\qquad ![q][q].maxVVal = \text{IF IF } state[q][q].maxBal \leq pp.maxVBal \\
&\qquad\qquad \text{THEN } pp.maxVVal \text{ ELSE } @]
\end{aligned}
$$

下面，我们举例说明 *TPaxosAP* 与 *TPaxos* 的不同之处。假设某参与者 q 收到了来自 p 的消息，消息内容包含 $\langle m, b, v \triangleq state[p][p].maxBal, state[p][p].maxVBal, state[p][p].maxVVal\rangle$，且 $m > b$。如果 p 的状态" 领先" 于 q，比如 p 比 q 多执行过几次 *Prepare* 与 *Accept*，则对 TPaxos 与 TPaxosAP 来说，

q 都可以在 $OnMessage(q)$ 中既作承诺又接受提议。具体而言，在 TPaxosAP 中，$\langle state[q][q].maxBal, state[q][q].maxVBal, state[q][q].maxVVal\rangle$ 被更新为 $\langle m, b, v\rangle$。这可以看作 q 先将其更新为 $\langle b, b, v\rangle$，然后再更新为 $\langle m, b, v\rangle$，相当于 q 先处理了针对提议 $\langle b, v\rangle$ 的 Accept 请求，然后处理了针对提议编号 m 的 Prepare 请求。然而，在 TPaxos 中，q 会将它的真实状态更新为 $\langle m, -, -\rangle$（$-$ 表示相应分量保持不变）。这相当于 q 先处理了针对提议编号 m 的 Prepare 请求，从而导致它不能再接受提议 $\langle b, v\rangle$。

我们将在第5章中论述 TPaxosAP 和 TPaxos 代表两种不同的投票机制。从上述例子得出，TPaxosAP 与 TPaxos 在实际应用中没有较大区别。

4.2.3 TPaxos 与 TPaxosAP 的正确性

我们证明 TPaxos 与 TPaxosAP 满足第3.5节介绍的两个条件 $OneValuePerBallot$ 与 $SafeAt(b, v)$，从而保证了一致性（Consistency）条件。$SafeAt(b, v)$ 由 $Accept$ 动作的第四和第五个合取式（确定值 v）保证，其证明与 Lamport 使用 TLAPS 证明 Paxos 的子阶段 Phase2a 保证了 $SafeAt(b, v)$[43] 类似。下面，我们证明 TPaxos 与 TPaxosAP 满足 $OneValuePerBallot$。该证明仅用到 $Accept$ 动作的前三个前置条件，因此对 TPaxos 与 TPaxosAP 都适用。我们先介绍两个简单的引理。

引理 4-1 对任意参与者 p，它接受过的提议编号 $state[p][p].maxVBal$ 是 (非严格) 单调递增的。

引理 4-2 对任意参与者 p，它承诺可以接受提议的最小编号始终大于等于它接受过的提议编号，即满足 $state[p][p].maxBal \leq state[p][p].maxVBal$。

$OneValuePerBallot$ 保证对于任意提议编号 b，最多对应一个提议 $\langle bb, vv\rangle$，即最多执行一次 $Accept(\_, b, \_)$（$\_$ 表示该分量可为任意值）。注意到 $Accept(\_, b, \_)$ 的第一个前置条件 $b \in Bals(p)$ 要求，只有 b 的拥有者 p 才能使用 b 执行 $Accept(p, b, \_)$，故只需要保证参与者 p 最多执行一次 $Accept(p, b, \_)$ 即可保证 $OneValuePerBallot$。

假设在时刻 t，参与者 p 执行了 $Accept(p, b, \_)$，此时 $state[p][p].maxVBal = b$。由 $Accept(p, b, \_)$ 的第三个前置条件 $state[p][p].maxVBal \neq b$ 可知，p 不能在 $state[p][p].maxVBal = b$ 的情况下执行 $Accept(p, b, \_)$。根据引理4-1，在任意时

刻 $t' \geq t$，都有 $state[p][p].maxVBal \geq b$。当 $state[p][p].maxVBal > b$ 时，根据引理 4-2可得，$state[p][p].maxBal \geq state[p][p].maxVBal > b$。由于 $Accept(p, b, \_)$ 的第二个前置条件 $state[p][p].maxBal \leq b$ 的限制，p 在时刻 $t' \geq t$ 也不能执行 $Accept(p, b, \_)$。综上所述，对于提议编号 b，它的拥有者 p 最多只能使用 b 执行一次 $Accept(p, b, \_)$，从而保证了 *OneValuePerBallot*。

4.3 小结

本章详细介绍了 TPaxos 协议。首先，我们给出了从经典 Paxos 到 TPaxos 的逐步推导。基于这个推导，我们可以将 TPaxos 看做 Paxos 的一种自然变体，降低了理解难度。接着，我们给出了 TPaxos 的 TLA+ 规约。针对消息处理阶段中先作承诺还是先接受提议的两种理解，我们提出了 TPaxos 的新变体，称为 TPaxosAP。我们举例说明了 TPaxosAP 与 TPaxos 的不同之处，并且论证了他们的正确性。

第五章　TPaxos 与 TPaxosAP 的精化

本章建立了 Paxos 协议变体间的精化关系。对于 TPaxos，我们仍采用 Paxos 所使用的 Voting 机制 (第3.5节)，建立了从 TPaxos 到 Voting 的精化关系。对于 TPaxosAP，我们发现 Voting 并不能完整刻画 TPaxosAP 的行为。因此，我们首先提出了一种新的投票机制，称作 EagerVoting。EagerVoting 允许参与者在接受提议的同时，作出比 Paxos/TPaxos 更“激进”的“不再接受具有更小编号的提议”的承诺。事实上，EagerVoting 和 Voting 是等价的，我们构建了这两者之间相互的精化关系证明了他们的等价性。然后，我们建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系。

5.1　TPaxos 的精化

需要注意的是，在 *TPaxos* 与 *TPaxosAP* 规约的 $Accept(p,b,v)$ 动作中，参与者 p 在前置条件满足的情况下，需要根据本地状态向量 $state[p]$ 确定值 v。然而，在 *Voting*（第3.5节）与 *EagerVoting*（将在第5.2.1节介绍）的 $VoteFor(a,b,v)$ 动作中，参与者需要根据某个议会的状态（而不是所有参与者）确定值 v。这并不会影响协议的正确性。但是，不难论证，存在某些情况是的两种方案确定的 v 值不同，这给构建从 *TPaxos* 到 *Voting* 以及从 *TPaxosAP* 到 *EagerVoting* 的精化关系造成了障碍。为此，在本章，我们修改了 *TPaxos* 与 *TPaxosAP* 的 $Accept(p,b,v)$ 动作，改用“从议会确定值 v”的方案[①]。修改后的 $Accept(p,b,v)$ 动作如下所示。

TPaxos 采取了与 *Paxos* 相同的投票机制，都可以用 *Voting*[43] 规约来刻画。本节构建从 *TPaxos* 到 *Voting* 的精化关系。*TPaxos* 中的动作与 *Voting* 中的动作存在着明确的对应关系。具体而言：

[①]我们将如下两个相关问题列入未来工作：第一，如何修改 *Voting* 的 *VoteFor* 动作，使其采用“从全体参与者确定值 v”的方案？第二，如何在采用不同（“确定值 v 的”）方案的 *Voting* 之间建立精化关系？

$$
\begin{array}{l}
Accept(p,\ b,\ v) \triangleq \\
\quad \land\ b \in Bals(p) \\
\quad \land\ \neg\exists\, m \in msgs : m.state[m.from].maxBal = b \\
\qquad\qquad\qquad \land\ m.state[m.from].maxVBal = b \\
\quad \land\ state[p][p].maxBal = b \quad \text{对应于 } Voting \text{ 中第一条合取式} \\
\quad \land\ state[p][p].maxVBal \neq b \quad \text{对应于 } Voting \text{ 中第二条合取式} \\
\quad \text{从本地一个议会中的状态选值} \\
\quad \land\ \exists\, Q \in Quorum : \quad \text{收到了“足够”的针对 } Prepare(p,b) \text{ 的回复} \\
\qquad \land\ \forall\, q \in Q : state[p][q].maxBal = b \\
\qquad \land\ \lor\ \forall\, q \in Q : state[p][q].maxVBal = -1 \quad \text{选择自己的值} \\
\qquad\quad \lor\ \exists\, c \in 0\,..\,(b-1) : \quad v \text{ 是提议编号最大的提议对应的值} \\
\qquad\qquad \land\ \forall\, r \in Q : state[p][r].maxVBal \leq c \\
\qquad\qquad \land\ \exists\, r \in Q : \land\ state[p][r].maxVBal = c \\
\qquad\qquad\qquad\qquad\ \land\ state[p][r].maxVVal = v \\
\quad \land\ state' = [state \text{ EXCEPT } ![p] = [state[p] \text{ EXCEPT} \\
\qquad\qquad\qquad\qquad\qquad ![p] = [state[p][p] \text{ EXCEPT} \\
\qquad\qquad\qquad\qquad\qquad\qquad !.maxVBal = b, \\
\qquad\qquad\qquad\qquad\qquad\qquad !.maxVVal = v]]] \\
\quad \land\ Send([from \mapsto p,\ to \mapsto Participant \setminus \{p\},\ state \mapsto state'[p]])
\end{array}
$$

- 在 *TPaxos* 的 $Prepare(p,b)$ 中，参与者 p 将 $state[p][p].maxBal$ 增加至更大的提议编号 b。这对应于 *Voting* 中的 $IncreaseMaxBal(a,b)$ 动作（此处 $a = p$）。具体而言，$Prepare(p,b)$ 中的前置条件 $state[p][p].maxBal < b$ 对应于 $IncreaseMaxBal(a,b)$ 中的前置条件 $maxBal[a] < b$。另外，$Prepare(p,b)$ 中的状态更新（将 $state[p][p].maxBal$ 更新为 b）也对应于 $IncreaseMaxBal(a,b)$ 中的状态更新（将 $maxBal[a]$ 更新为 b）。
- 在 *TPaxos* 的 $Accept(p,b,v)$ 中，参与者 p 在确定值 v 之后，随即接受了提议 $\langle b,v\rangle$。这对应于 *Voting* 中的 $VoteFor(a,b,v)$ 动作（此处 $a = p$）。具体而言：
  - $Accept(p,b,v)$ 中的第二个前置条件 $state[p][p].maxBal \leq b$ 与第三个前置条件 $state[p][p].maxVBal \neq b$ 分别对应于 $VoteFor(a,b,v)$ 中的第一个前置条件 $maxBal[a] \leq b$ 与第二个前置条件 $\forall vt \in votes[a] : vt[1] \neq b$。

- $VoteFor(a,b,v)$ 中的第三个前置条件 $\forall c \in Acceptor \backslash \{a\} : \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v)$ 与第二个前置条件共同保证 *OneValuePerBallot*。在 $Accept(p,b,v)$ 中，*OneValuePerBallot* 由它的前三个前置条件共同保证。
- $Accept(p,b,v)$“从议会 $Q \in Quorum$ 确定值 v”的方法与 $VoteFor(a,b,v)$ 中的 $ShowSafeAt(Q,b,v)$ 对应。
- $Accept(p,b,v)$ 中的状态更新（即，分别将 $state[p][p].maxVBal$ 与 $state[p][p].maxVVal$ 更新为 b 与 v）对应于 $VoteFor(a,b,v)$ 中对 *votes* 的更新（将提议 $\langle b,v \rangle$ 加入到 $votes[a]$ 中），表示参与者 a（即 p）接受了提议 $\langle b,v \rangle$。
- $VoteFor(a,b,v)$ 会将 $max[a]$ 更新为 b。对于 *TPaxos*，参与者 p 可以执行 $Accept(p,b,v)$，表明 p 已执行过 $Prepare(p,b)$ 并更新了 $state[p][p].maxBal = b$。根据第4.2.1.3节对前置条件 $state[p][p].maxBal \leq b$ 的分析可知，在 p 执行 $Accept(p,b,v)$ 时，$state[p][p].maxBal = b$ 仍然成立。

- 现在考虑 *TPaxos* 中的 $OnMessage(q)$ 动作（第4.2.1.3节）。参与者 q 收到了来自 $p \triangleq m.from$ 的消息 $m \in msgs$。记 $pp \triangleq m.state[p]$ 为 m 中携带的 p 的真实状态，即 $(pp.maxBal, pp.maxVBal, pp.maxVVal)$。在 $UpdateState(q,p,pp)$ 中，q 需要根据 pp 更新自身真实状态 $state[q][q]$。这里可能有两种更新结果：第一种是，只“作承诺”，即将 $state[q][q].maxBal$ 更新至 $pp.maxBal$。此时，“接受提议”的条件不再成立。这种情况对应于 *Voting* 中的 $IncreaseMaxBal(a,b)$ 动作（此处 $a = p, b = pp.maxBal$）；第二种是，“作承诺”且“接受提议”，即将 $state[q][q].maxBal$ 更新至 $pp.maxBal$，并且分别将 $state[q][q].maxVBal$ 与 $state[q][q].maxVVal$ 更新为 $pp.maxVBal$ 与 $pp.maxVVal$。这要求 $state'[q][q].maxBal = pp.maxBal = pp.maxVBal$ 成立（$state'[q][q].maxBal$ 为“作承诺”更新后的值）。因此，这种情况对应于 *Voting* 中的 $VoteFor(a,b,v)$ 动作（此处 $a = p, b = pp.maxVBal, v = pp.maxVVal$）。

为了构建从 *TPaxos* 到 *Voting* 的精化关系，我们需要在 *TPaxos* 规约中模拟 *Voting* 中的变量 *maxBal* 与 *votes*，即参与者承诺可以接受提议的最小编号以及已接受的提议构成的集合；见模块 *TPaxosWithVotes*。从上面的分析可以得出，*Voting* 中的 $maxBal[p]$ 对应于 *TPaxos* 中的

$state[p][p].maxBal$。因此，针对 $maxBal$，我们定义精化映射为 $maxBal \triangleq [p \in Participant \mapsto state[p][p].maxBal]$。对于 $votes$，我们在 $TPaxos$ 的基础上添加辅助变量[25][44]$votes$（在 $TPaxosWithVotes$ 模块中，故与 $Voting$ 中的 $votes$ 冲突），为每个参与者 p 收集它已接受过的提议 $votes[p]$。具体而言，在 $Prepare(p,b)$ 中，参与者 p 未接受任何提议，所以在 $PrepareV(p,b)$ 中，$votes$ 保持不变。在 $Accept(p,b,v)$ 中，参与者 p 接受了提议 $\langle b,v\rangle$，所以，在 $AcceptV(p,b,v)$ 中，我们将 $\langle b,v\rangle$ 添加到 $votes[p]$。在 $OnMessage(q)$ 中，如果 $state[q][q].maxVBal$ 发生了改变，则表明 q 接受了新的提议①。在 $OnMessageV(q)$ 中，我们将该提议添加到 $votes[q]$。最后，$TPaxosWithVotes$ 模块给出了从 $TPaxos$ 到 $Voting$ 的精化映射（即用辅助变量替换 $Voting$ 中的对应变量）：

$$V \triangleq \text{INSTANCE } Voting \text{ WITH } Accept \leftarrow Participant,\ maxBal \leftarrow maxBal,\ votes \leftarrow votes$$

5.2 TPaxosAP 的精化

我们先介绍使用与 TPaxosAP 的投票机制 EagerVoting，然后建立从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系。

5.2.1 EagerVoting 规约

根据第5.1节的分析，在 $TPaxos$ 中，每个参与者 p 要么仅提高它承诺不再接受的最小提议编号 $state[p][p].maxBal$，要么在接受提议 $\langle b,v\rangle$ 的同时也将 $state[p][p].maxBal$ 更新为 b。这与 $Voting$ 规约是相符的。然而，$Voting$ 并不能完全刻画 $TPaxosAP$ 的行为。如第5.1节所述，$TPaxosAP$ 与 $TPaxos$ 的行为在 $OnMessage(q)$ 消息处理阶段的状态更新 $UpdateState(q,p,pp)$ 方面有所不同：在 $TPaxosAP$ 中，参与者先接受提议 $\langle pp.maxVBal, pp.maxVVal$（即，更新 $state[q][q].maxVBal$ 与 $state[q][q].maxVVal$）后作承诺（即，将 $state[p][p].maxBal$ 提高至 $pp.maxBal$）。这可能导致更新后的 $state[p][p].maxBal$ 不等于 $pp.maxVBal$，

①这是因为 $OneValuePerBallot$ 成立。

从而违反了 *Voting* 规约。这种情况是可能发生的。考虑如下示例:

MODULE *EagerVoting*

EXTENDS *Sets*

CONSTANT *Value*, *Acceptor*, *Quorum*

$Ballot \triangleq Nat$

VARIABLES *votes*, *maxBal*

$$
\begin{aligned}
TypeOK \triangleq\ & \land votes \in [Acceptor \to \text{SUBSET } (Ballot \times Value)] \\
& \land maxBal \in [Acceptor \to Ballot \cup \{-1\}]
\end{aligned}
$$

$$VotedFor(a, b, v) \triangleq \langle b, v\rangle \in votes[a]$$

$$DidNotVoteAt(a, b) \triangleq \forall v \in Value : \neg VotedFor(a, b, v)$$

$$
\begin{aligned}
&ShowsSafeAt(Q, b, v) \triangleq \\
&\quad \land \forall a \in Q : maxBal[a] \geq b \quad \text{承诺过} \\
&\quad \land \exists c \in -1 \,..\, (b-1) : \\
&\qquad \land (c \neq -1) \implies \exists a \in Q : VotedFor(a, c, v) \\
&\qquad \land \forall d \in (c+1) \,..\, (b-1), a \in Q : DidNotVoteAt(a, d)
\end{aligned}
$$

$$
\begin{aligned}
Init \triangleq\ & \land votes = [a \in Acceptor \mapsto \{\}] \\
& \land maxBal = [a \in Acceptor \mapsto -1]
\end{aligned}
$$

$$
\begin{aligned}
IncreaseMaxBal(a, b) \triangleq\ & \land maxBal[a] < b \\
& \land maxBal' = [maxBal \text{ EXCEPT } ![a] = b] \quad \text{作承诺} \\
& \land \text{UNCHANGED } votes
\end{aligned}
$$

EagerVoting 和 *Voting* 的唯一区别在于：在 *Voting* 中，我们只更新 *maxBal* 为 b，即 $maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$。

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land maxBal[a] \leq b \quad \text{不违背承诺} \\
&\quad \land \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land \forall c \in Acceptor \setminus \{a\} : \\
&\qquad \forall vt \in votes[c] : (vt[1] = b) \implies (vt[2] = v) \\
&\quad \land \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \quad \text{提议 } \langle b, v\rangle \text{ 是安全的，能够投票} \\
&\quad \land votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v\rangle\}] \quad \text{投票} \\
&\quad \land \exists c \in Ballot : c \geq b \\
&\qquad\qquad \land maxBal' = [maxBal \text{ EXCEPT } ![a] = c] \quad \text{作承诺}
\end{aligned}
$$

$$
\begin{aligned}
Next \triangleq\ & \exists a \in Acceptor, b \in Ballot : \\
& \lor IncreaseMaxBal(a, b) \\
& \lor \exists v \in Value : VoteFor(a, b, v)
\end{aligned}
$$

$$Spec \triangleq Init \land \Box[Next]_{\langle votes, maxBal\rangle}$$

假设有三个参与者 $p1$、$p2$、$p3$，$p1$ 先执行 $Prepare(1)$。$p2$、$p3$ 收到该 Prepare 请求后更新状态并回复 $p1$。接着，$p1$ 执行 $Accept(1, v_1)$ 以及 $Prepare(2)$，并发送携带 $\langle 2, 1, v_1 \rangle$ 的 Prepare 请求。如果 $p2$ 与 $p3$ 收到该请求并将自身状态更新为 $\langle 2, 1, v_1 \rangle$，则违反了 *Voting* 规约。

为了弥补 *Voting* 的不足，我们提出一种新的适用于 TPaxosAP 的投票机制，称为 *EagerVoting*。*EagerVoting* 与 *Voting* 的唯一不同在于：在 $VoteFor(a, b, v)$ 动作中，我们允许参与者 a 在接受提议 $\langle b, v \rangle$ 的同时，将 $maxBal[a]$ 提高到比 b 更大的提议编号 $c \geq b$。

5.2.2 从 EagerVoting 到 Consensus 的精化

```
MODULE TPaxosAPWithVotes
EXTENDS TPaxosAP
VARIABLE votes   votes[q]: 参与者 q 接受过的提议集合
varsV ≜ ⟨vars, votes⟩
----------------------------------------------------------
InitV ≜  ∧ Init
         ∧ votes = [q ∈ Participant ↦ {}]
PrepareV(p, b) ≜  ∧ Prepare(p, b)
                  ∧ votes' = votes
AcceptV(p, b, v) ≜  ∧ Accept(p, b, v)
                    ∧ votes' = [votes EXCEPT ![p] = @ ∪ {⟨b, v⟩}]
OnMessageV(q) ≜  ∧ OnMessage(q)
                 ∧ IF state'[q][q].maxVBal ≠ state[q][q].maxVBal
                      THEN votes' = [votes EXCEPT ![q] = @ ∪
                        {⟨state'[q][q].maxVBal, state'[q][q].maxVVal⟩}]
                      ELSE UNCHANGED votes
----------------------------------------------------------
NextV ≜ ∃ p ∈ Participant :
            ∨ OnMessageV(p)
            ∨ ∃ b ∈ Ballot :  ∨ PrepareV(p, b)
                              ∨ ∃ v ∈ Value : AcceptV(p, b, v)
SpecV ≜ InitV ∧ □[NextV]_varsV
----------------------------------------------------------
maxBal ≜ [p ∈ Participant ↦ state[p][p].maxBal]
V ≜ INSTANCE EagerVoting WITH Acceptor ← Participant
                              votes ← votes, maxBal ← maxBal
THEOREM SpecV ⟹ V!Spec
```

为了构建从 *EagerVoting* 到 *Consensus* 的精化关系，我们需要在 *EagerVoting* 中模拟 *Consensus* 规约中的变量 *chosen*，即协议已选中的值构成的集合；见 *EagerVotingWithChosen*。一个提议 $\langle b, v\rangle$ 以及它包含的值被选中当且仅当存在某个议会 $Q \in Quorum$ 的接受者都接受了该提议；见 $ChosenAt(b, v)$。辅助变量 *chosen* 则收集了所有被选中的值。最后，该模块给出了从 *EagerVoting* 到 *Consensus* 的精化映射 $C \triangleq$ INSTANCE *Consensus* WITH $Acceptor \leftarrow Acceptor, chosen \leftarrow chosen$，即用辅助变量 *chosen* 替换 *Consensus* 中的变量 *chosen*。

5.2.3 从 TPaxosAP 到 EagerVoting 的精化

第5.1节描述的 *TPaxos* 和 *Voting* 之间的对应关系在 *TPaxosAP* 与 *EagerVoting* 之间仍然成立。需要注意的是，根据第5.2.1节中的论述，*TPaxosAP* 的 $OnMessage(q)$ 动作对“先接受提议后作承诺”的改动对应于 *EagerVoting* 的 $VoteFor(a, b, v)$ 动作对“允许将 $maxBal[a]$ 提高到 $c \geq b$”的改动。因此，*TPaxosAP* 中的动作与 *EagerVoting* 中的动作也存在着明确的对应关系。从 *TPaxosAP* 到 *EagerVoting* 的精化关系与从 *TPaxos* 到 *Voting* 的精化关系相同，构建方法也相同，此处不再赘述。

5.3 EagerVoting 和 Voting 的等价性

直观地讲，*Voting* 和 *EagerVoting* 是等价的。一方面，由于 *Voting* 的每个动作都是 *EagerVoting* 所允许的，因此 *Voting* 的每个行为也都是 *EagerVoting* 所允许的。另一方面，如果 *EagerVoting* 的动作 $VoteFor(a, b, v)$ 在更新 $maxBal[a]$ 时选择的提议编号 c 等于 b，则该动作与 *Voting* 中的 $VoteFor(a, b, v)$ 等价；如果 c 大于 b，则该动作可以看作 *Voting* 中的 $VoteFor(a, b, v)$ 与 $IncreaseMaxBal(a, c)$ 的组合。因此，*EagerVoting* 的每个行为都可以被 *Voting* 中的行为所模拟。

接下来，我们将使用精化技术分别构建从 Voting 到 EagerVoting 和从 EagerVoting 到 Voting 的精化映射，从而证明这两者的等价性。

5.3.1 Voting 精化 EagerVoting

事实上，对于 *Voting* 中的每个动作，我们都能够在 *EagerVoting* 找到对应的动作。首先，动作 *IncreaseMaxBal* 两者是一致的。其次，*Voting* 的 $VoteFor(a,b,v)$ 动作只会更新 *maxBal* 至 b，并且 *EagerVoting* 的 $VoteFor(a,b,v)$ 动作允许更新 *maxBal* 至 b。

为了构建从 *Voting* 到 *EagerVoting* 的精化关系，我们需要在 *Voting* 中模拟 *EagerVoting* 规约中的变量 *votes* 和 *maxBal*。根据上述的分析，我们直接使用 *Voting* 的这两个变量进行模拟，即构造精化映射为 $V \triangleq$ INSTANCE *EagerVoting*WITH $maxBal \leftarrow maxBal, votes \leftarrow votes$。

5.3.2 EagerVoting 精化 Voting

构建从 *Voting* 到 *EagerVoting* 的精化映射比较简单，反过来的精化映射却比较复杂。这是由于，*EagerVoting* 的动作 $VoteFor(a,b,v)$ 可能等价于 *Voting* 中的 $VoteFor(a,b,v)$，也可能等价于 *Voting* 中的 $VoteFor(a,b,v)$ 与 $IncreaseMaxBal(a,c)$ 的组合。对于后一种组合情况，直接使用 *EagerVoting* 中的 *votes* 和 *maxBal* 等变量模拟 *Voting* 规约中的变量 *votes* 和 *maxBal* 会遇到较大的阻碍。为了顺利的构建从 *EagerVoting* 到 *Voting* 的精化关系，我们引入了一种称为 stuttering variables 的辅助变量。

精化映射是一种基于状态的映射关系，具体来说，它刻画了底层实现规约中的数据结构如何表示更高层规约中相应的结构。精化映射通常是一对一的，即底层规约的一个状态对应于高层规约的一个状态。然而，两个规约的等价性是行为上的等价性，可能是底层规约的一个状态对应于高层规约的多个状态，也可能是多个状态对应于一个状态。在状态数量不对等的情况下，我们通常引入辅助变量增删中间状态以构建精化映射。辅助变量的添加不会改变原先规约中变量的行为，也不需要被具体实现。而 stuttering variable 是其中一种辅助变量，它能够增加 stuttering 状态，即规约中的原有变量保持不变。接下来，我们举例说明该变量的使用方法。

对于一个 24 小时的时钟系统，我们可以只刻画时针 h 得到规约 $Spec_1$，也可以同时刻画时针 h 和分针 m 得到规约 $Spec_2$。直观来讲，$Spec_1$ 等价于 $Spec_2$。为了证明 $Spec_1$ 实现 $Spec_2$，我们需要构建从 $Spec_1$ 到 $Spec_2$ 的精化关系。注意到前者的一个动作（递增 h）对应于后者的 60 个动作（递增 m 直到

h 递增），我们首先需要解决状态数量不对等的问题。为此，我们在 $SpecV_1$ 中增加辅助变量 s，并且每次递增 h 时增加了 59 个改变 s 的动作，从而得到规约 $Spec_1^s$。接着，我们直接根据变量 s 和 h 模拟规约 $SpecV_2$ 中的变量 m 和 h，即构建精化映射为 $S \triangleq$ INSTANCE $SpecV_2$ WITH $h \leftarrow h, m \leftarrow s$[①]。

```
MODULE EagerVotingSStuttering
EXTENDS EagerVoting
VARIABLES s

InitS ≜ Init ∧ (s = top)
IncreaseMaxBalSStutter(a, b) ≜
    IF s = top
      THEN IncreaseMaxBal(a, b) ∧ s' = s
      ELSE UNCHANGED vars ∧ s' = s
VoteForPostSStutter(a, b, v) ≜
    IF s = top
      THEN  ∧ VoteFor(a, b, v)
            ∧ s' = IF b ≠ maxBal'[a]
                   THEN [acc ↦ a, val ↦ b]
                   ELSE top
      ELSE UNCHANGED vars ∧ s' = top

V ≜ INSTANCE Voting WITH
    votes ← votes,
    maxBal ← IF s = top
             THEN maxBal
             ELSE [a ∈ Acceptor ↦ IF a = s.acc
                                  THEN s.val
                                  ELSE maxBal[a]]
```

考虑到 *EagerVoting* 的 *VoteFor* 动作有两种对应关系，我们需要进行分类讨论。首先，当 $VoteFor(a,b,v)$ 等价于 *Voting* 中的 $VoteFor(a,b,v)$ 时，无需添加额外的辅助变量。其次，当 $VoteFor(a,b,v)$ 等价于 *Voting* 中的 $VoteFor(a,b,v)$ 与 $IncreaseMaxBal(a,c)$ 的组合，此时我们增加一个 stuttering 动作，即保持原有变量不变，并且记录更新后的 $maxBal[a]$ 值。这样，*VoteFor* 动作将对应于 *Voting* 中的 *VoteFor* 动作，而 stuttering 动作对应于 *Voting* 的

[①]这里构建了从 $Spec_1^s$ 到 $SpecV_2$ 的精化映射，而 $Spec_1^s$ 等价于 $SpecV_1$ 的证明过程比较复杂，Lamport 给出了这两者等价的约束。

IncreaseMaxBal 动作。构建精化关系时，*Voting* 中 *maxBal* 变量的模拟同样分为上述两种情况讨论，前者可以直接使用 *EagerVoting* 的 *maxBal*，后者需要使用 *maxBal* 以及记录的更新后 *maxBal*[*a*] 的值共同模拟。

5.4 Paxos 协议变体间的精化关系

综上所述，我们构建了一个如图5-1所示的精化关系。其中，左侧是目前已有的工作，即从 Paxos 到 Voting 再到 Consensus 的精化关系。右侧是本文的贡献，提出了等价于 Voting 的新投票机制 EagerVoting（构建了这两者相互的精化关系以论证等价性），并且构建了从 TPaxos 到 Voting、TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系。

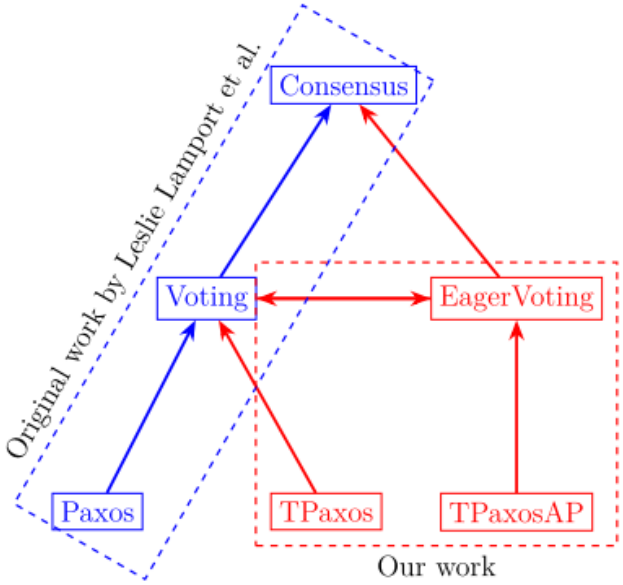

图 5-1: TPaxos 与 Paxos 的精化关系

精化关系的提出是必要的。首先，针对 TPaxos，精化关系直观的刻画了 TPaxos 协议与 Paxos 协议的变体关系。虽然没有直接展示第4.1.2节中的推导关系，精化关系直接阐明了 TPaxos 与 Paxos 协议核心的一致性，即采用了同样的投票机制 Voting。其次，在 Voting 的基础上，我们提出了一个等价的投票机制 EagerVoting。EagerVoting 丰富了 Paxos 类协议的投票机制，能更全面的揭示 Paxos 类协议的投票机制。最后，不局限于 TPaxos 与 TPaxosAP，更多的 Paxos 类协议都可以通过构建精化关系的方式来论证其正确性。

5.5 小结

本章使用精化关系论证了 TPaxos 与 TPaxosAP 的正确性。针对 TPaxos，我们使用了已有的 Voting 机制，建立了从 TPaxos 到 Voting 的精化关系。对于 TPaxosAP，我们引入了一个新的投票机制 EagerVoting。然后，我们构建了 EagerVoting 与 Voting 相互的精化关系，这揭示了这两者的等价性。接着，构建了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系。精化关系图 5-1直观地揭示了经典的 Paxos、TPaxos 与 TPaxosAP 的关系，即它们核心的一致性，采用了等价的投票机制。对于更多的 Paxos 类协议，我们可以采用类似的方式构建精化关系以刻画它们的变体关系。

第六章　TPaxos 协议的定理证明

本章介绍使用 TLAPS 证明 TPaxos 正确性的具体过程。首先，我们引入若干辅助谓词及不变式，即 TPaxos 需要满足的正确性的形式化定义。然后，给出证明思路的整体框架，该框架同样适用于证明其他分布式算法的正确性。接着，我们将介绍框架中的核心部分，即多个不变式的证明。最后，对 TPaxos 的证明进行总结，讨论证明过程中做出的优化。

6.1　辅助谓词

TPaxos 协议采取了与 Paxos 相同的投票机制，我们从“接受提议”这一角度给出 TPaxos 一致性的形式化定义。

谓词 $VotedForIn(p,\ b,\ v)$ 为真当且仅当接受者 p 接受过提议 $\langle b,\ v\rangle$。在 TPaxos 中，只有 *Accept* 动作和 *OnMessage* 动作允许接受者接受新提议，并且接受者在接受提议的同时发送了携带自身真实状态的消息。此时，真实状态中的 *maxBal* 和 *maxVBal* 分量都更新为 b。

$$
\begin{aligned}
&VotedForIn(p \in Participant,\ b \in Ballot, v \in Value) \triangleq \\
&\qquad \exists\, m \in msgs : \wedge\ m.from = a \\
&\qquad\qquad\qquad\qquad \wedge\ m.state[p].maxBal = b \\
&\qquad\qquad\qquad\qquad \wedge\ m.state[p].maxVBal = b \\
&\qquad\qquad\qquad\qquad \wedge\ m.state[p].maxVVal = v
\end{aligned}
$$

谓词 $ChosenIn(b,\ v)$ 为真当且仅当存在某个接受者的多数派 Q，Q 中的所有接受者都接受过提议 $\langle b, v\rangle$。

$$ChosenIn(b \in Ballot,\ v \in Value) \triangleq \exists\, Q \in Quorum : \forall\, a \in Q : VotedForIn(a,\ b,\ v)$$

谓词 $Chosen(v)$ 为真当且仅当存在某个提议编号 b，谓词 $ChosenIn(b,\ v)$ 成

立。*Chosen*(*v*) 的成立意味着 *v* 值被选中 (chosen)。

$$Chosen(v \in Value) \triangleq \exists\, b \in Ballot : ChosenIn(b,\ v)$$

谓词 *Consistency* 给出了 TPaxos 一致性的形式化定义，即最终有且只有一个值被选中。

$$Consistency \triangleq \forall\, v1,\ v2 \in\ Value : Chosen(v1) \wedge Chosen(v2) \Rightarrow v1 = v2$$

谓词 *WontVoteIn*(*p*, *b*) 成立当且仅当接受者 *p* 接受了一个编号大于 *b* 的 Prepare 请求，并且没有接受过也不会接受任何编号为 *b* 的提议。

$$WontVoteIn(p \in Participant,\ b \in Ballot) \triangleq$$
$$state[p][p].maxBal > b \wedge \forall v \in Value : \neg VotedForIn(p,\ b,\ v)$$

谓词 $SafeAt(b,\ v)$ 蕴含提议 $\langle b,\ v\rangle$ 是安全的，即对于任何提议 $\langle b',\ v'\rangle$，其中 $b' \in 0..(b-1)$, $v' \neq v$, $\langle b',\ v'\rangle$ 尚未被选中并且将来也不会被选中。具体而言，$SafeAt$ 断言对于任何编号小于 *b* 的编号 b'，都存在某个多数派 *Q*，*Q* 中所有的接受者要么接受过提议 $\langle b',\ v\rangle$，要么没有接受过并且将来也不会接受编号为 b' 的提议。

$$SafeAt(b \in Ballot,\ v \in Value) \triangleq \forall\, c \in 0..(b-1) : \exists\, Q \in Quorum :$$
$$\forall\, p \in Q : VotedForIn(p,\ c,\ v) \vee WontVoteIn(p,\ c)$$

与 Paxos 协议相对应，TPaxos 协议中可以接受的提议同样需要满足 *OneValuePerBallot* 和 $SafeAt$[42] 这两个关键条件。我们通过引入以下两个引理来证明这两个关键条件在 TPaxos 中依然成立。

引理 6-1 (VotedOnce) 对任意可以接受的提议，相同编号对应的提议值始终是相等的。即 $\forall\ p1,\ p2 \in Participant,\ b \in Ballot,\ v1,\ v2 \in Value$: $VotedForIn(p1,\ b,\ v1) \wedge VotedForIn(p2,\ b,\ v2) \Rightarrow v1 = v2$。

引理 6-2 (VotedInv) 如果接受者接受了提议 $\langle b,\ v\rangle$，那么谓词 $SafeAt(b,\ v)$ 成立。即 $\forall\ p \in Participant,\ b \in Ballot,\ v \in Value\ :\ VotedForIn(p, b, v) \Rightarrow SafeAt(b, v)$。

其中引理6-1证明了条件 *OneValuePerBallot* 的成立，引理6-2证明了条件 $SafeAt$ 的成立。

6.2 不变式

我们定义了三种类型的不变式，包括消息不变式，状态不变式以及类型不变式。其中，消息不变式描述了 TPaxos 各个参与者之间传递的消息需要满足的性质。状态不变式则对各个参与者的本地数据进行约束。另外，类型不变式确保 TPaxos 规约中的变量是正确的类型并且属于正确的取值范围。

6.2.1 类型不变式

谓词 $State$ 定义了状态内容，包含三个分量（$maxBal, maxVBal, maxVVal$）。$Message$ 定义了格式统一的消息，包含发送者（分量 $from$），接收者（分量 to），携带的状态向量（分量 $state$）。不变式 $TypeOK$ 描述了 TPaxos 规约中变量的类型及取值范围。其中，$msgs$ 是 $Message$ 的子集，并且其中所有的消息格式是统一的。$state$ 变量则是一个函数，定义域是参与者 $Participant$ 集合，值域同样是一个定义域为参与者 $Participant$ 并且值域为状态 $State$ 的函数。例如，参与者 p 维护的状态为 $state[p]$，该值的类型为 $[Participant \to State]$。其中，真实状态（$state[p][p]$）及对参与者 q 的观察状态（$state[p][q]$）属于 $State$ 集合。

$$
\begin{aligned}
State \triangleq\ & [maxBal : Ballot \cup \{-1\},\\
& \ maxVBal : Ballot \cup \{-1\},\ maxVVal : Value \cup \{None\}]\\
Message \triangleq\ & [from : Participant,\\
& \ to : \text{SUBSET } Participant,\\
& \ state : [Participant \to [maxBal : Ballot \cup \{-1\},\\
& \qquad maxVBal : Ballot \cup \{-1\},\\
& \qquad maxVVal : Value \cup \{None\}]]]\\
TypeOK \triangleq\ & \\
& \land msgs \subseteq Message\\
& \land state \in [Participant \to [Participant \to State]]
\end{aligned}
$$

6.2.2 消息不变式

MsgInv 定义了所有消息需要满足的不变式，其中：

$$
\begin{array}{l}
MsgInv \triangleq \forall\, m \in msgs : \\
\quad \text{LET } p \triangleq m.from \wedge\ curState \triangleq m.state[p] \\
\quad \text{IN} \quad \wedge\, m.from \notin m.to \ \wedge\, curState.maxBal \in Ballot \\
\qquad \wedge\ \vee\ \wedge\, curState.maxVVal \in Value \wedge curState.maxVBal \in Ballot \\
\qquad\qquad \wedge\, VotedForIn(m.from,\, curState.maxVBal,\, curState.maxVVal) \\
\qquad\quad \vee\ \wedge\, curState.maxVVal = None \wedge curState.maxVBal = -1 \\
\qquad \wedge\, curState.maxBal \geq curState.maxVBal \\
\qquad \wedge \forall\, q \in Participant :\ \wedge\, m.state[q].maxVBal \leq state[q][q].maxVBal \\
\qquad\qquad\qquad\qquad \wedge\, m.state[q].maxBal \leq state[q][q].maxBal \\
\qquad \wedge\, curState.maxBal \neq curState.maxVBal \\
\qquad\qquad \Rightarrow\ \wedge \forall\, c \in (curState.maxVBal + 1)\,..\,(curState.maxBal - 1) : \\
\qquad\qquad\qquad \neg\exists\, v \in Value : VotedForIn(p,\, c,\, v) \\
\qquad \wedge\, curState.maxBal = curState.maxVBal \\
\qquad \Rightarrow\ \wedge\, SafeAt(curState.maxVBal,\, curState.maxVVal) \\
\qquad\quad \wedge \forall\, ma \in msgs : (ma.state[ma.from].maxBal = curState.maxBal \\
\qquad\qquad \wedge\, ma.state[ma.from].maxBal = ma.state[ma.from].maxVBal) \\
\qquad\qquad\quad \Rightarrow ma.state[ma.from].maxVVal = curState.maxVVal
\end{array}
$$

- 第一条合取式保证了发送者不会将消息发送给自己。
- 第二条和第三条合取式限制了消息携带的真实状态的各个分量取值范围，*maxBal* 分量取值范围为 *Ballot* 集合（初始化为 −1，后续无论 Prepare 动作还是 Accept 动作均会更新值）。*maxVBal* 和 *maxVVal* 这两个分量值是同时更新的，要么都为初始值（−1 和 *None*），要么分别属于 *Ballot* 集合和 *Value* 集合。并且这两个分量值不为初始值时，蕴含着发送者接受过提议 ⟨*maxVBal*, *maxVVal*⟩。
- 第四条合取式描述了消息携带的真实状态中 *maxBal* 分量与 *maxVBal* 分量的关系，即 $maxBal \geq maxVBal$。
- 第五条合取式描述了消息携带的状态向量 *m.state* 和参与者维护的真实状态之前的关系，*m.state* 各个分量代表的状态均等于或者落后于其对应的参与者维护的真实状态。注意到在 TPaxos 协议中，任意参与者真实状态的 *maxBal* 分量和 *maxVBal* 分量是（非严格）单调递增的，故可以使用等于小于来刻画等于或者落后关系。

- 考虑到处理不同请求产生的消息需要满足的性质是不同的，即便 TPaxos 协议将消息类型进行了统一，第六和第七条合取式仍需要对消息进行分类讨论。针对 Prepare 请求（$maxBal \neq maxVBal$），第六条合取式断言发送者未接受过任何编号介于 $maxVBal$ 与 $maxBal$ 之间的提议。针对 Accept 请求（$maxBal = maxVBal$），第七条合取式断言提议 $\langle curState.maxVBal, curState.maxVVal\rangle$ 是安全的，并且在所有处理 Accept 请求发出的消息中，同一个编号只对应唯一的一个提议值。

6.2.3 状态不变式

不变式 *AccInv* 描述参与者维护的状态向量所满足的性质，其中：

$$
\begin{aligned}
&AccInv \triangleq \\
&\quad \forall\, p \in Participant : \\
&\qquad \land\ (state[p][p].maxVBal = -1) \Leftrightarrow (state[p][p].maxVVal = None) \\
&\qquad \land\ \forall\, q \in Participant : state[p][q].maxVBal \leq state[p][q].maxBal \\
&\qquad \land\ (state[p][p].maxVBal \geq 0) \\
&\qquad\qquad \Rightarrow VotedForIn(p, state[p][p].maxVBal, state[p][p].maxVVal) \\
&\qquad \land\ \forall\, c \in Ballot : c > state[p][p].maxVBal \\
&\qquad\qquad\qquad \Rightarrow \neg\exists\, v \in Value : VotedForIn(p, c, v) \\
&\qquad \land\ \forall\, q \in Participant : \\
&\qquad\quad \land\ state[p][p].maxBal \geq state[q][p].maxBal \\
&\qquad\quad \land\ state[p][p].maxVBal \geq state[q][p].maxVBal \\
&\qquad \land\ \forall\, q \in Participant : \\
&\qquad\quad state[p][q].maxBal \in Ballot \\
&\qquad\qquad \Rightarrow \exists\, m \in msgs : \\
&\qquad\qquad\quad \land\ m.from = q \\
&\qquad\qquad\quad \land\ m.state[q].maxBal = state[p][q].maxBal \\
&\qquad\qquad\quad \land\ m.state[q].maxVBal = state[p][q].maxVBal \\
&\qquad\qquad\quad \land\ m.state[q].maxVVal = state[p][q].maxVVal
\end{aligned}
$$

- 第一条合取式约束了参与者真实状态的 $maxVBal$ 分量与 $maxVVal$ 分量的取值范围，要么均为初始值，要么均不为初始值。
- 第二条合取式描述了所有状态的 $maxVBal$ 分量与 $maxBal$ 分量之间的关系，$maxBal \geq maxVBal$。

- 第三条合取式描述了参与者 p 真实状态与其他参与者 q 对 p 的观察状态之间的关系，观察状态始终不更新于真实状态（$state[p][p].maxBal \geq state[q][p].maxBal \wedge state[p][p].maxVBal \geq state[q][p].maxVBal$）。
- 第四条合取式断言，对任意参与者 p，如果它接受过的提议编号 $state[p][p].maxVBal \geq 0$，则参与者 p 一定接受过提议 $\langle state[p][p].maxVBal, state[p][p].maxVVal\rangle$。
- 第五条合取式保证了，对任何参与者 p，它没有接受过任何编号大于其接受过的最大提议编号（$state[p][p].maxVBal$）的提议。
- 第六条合取式描述了参与者维护的状态向量的更新机制。对于观察状态，一旦该状态的 *maxBal* 分量发生更新（$maxBal \in Ballot$），则存在触发参与者更新其观察状态的消息。对于真实状态，相应的 *maxBal* 发生更新，该参与者总会发送携带其真实状态的信息。

6.3 证明框架

我们需要证明 TPaxos 协议满足 *Consistency* 性质，也就是定理 $Safety$。我们分成两步进行，首先给出不变式 *Inv* 的定义，并且证明 *Inv* 能保证 *Consistency* 的成立（引理 6-3）。然后，我们证明 TPaxos 规约能保证 *Inv* 始终成立（引理 6-4）。因此，根据时序逻辑，可得 *Consistency* 始终成立，即 $Safety$ 定理成立。下面，我们分别介绍这两个引理的证明。

$$\text{THEOREM } Safety \triangleq Spec \Rightarrow \Box Consistency$$

$$Inv \triangleq MsgInv \wedge AccInv \wedge TypeOK$$

引理 6-3 不变式 *Inv* 蕴含谓词 *Consistency* 的成立，即 $Inv \Rightarrow Consistency$。

引理 6-4 TPaxos 规约保证不变式 *Inv* 始终成立，即 $Spec \Rightarrow \Box Inv$。

6.3.1 $Inv \Rightarrow Consistency$

首先，我们将证明目标 *Consistency* 展开，即转换为在 $ChosenIn(b1, v1) \wedge ChosenIn(b2, v2) \wedge b1 \leq b2$ 的情况下，证明 $v1 = v2$。我们对 $b1$ 和 $b2$ 的关系进行分类讨论，证明在 $b1 = b2$ 及 $b1 < b2$ 的情况下 $v1 = v2$ 均成立。

- $b1 = b2$。将 *ChosenIn* 定义展开，可得 $VotedForIn(a \in Participant, b1, v1) \wedge VotedForIn(a \in Participant, b2, v2)$ 成立。根据引理6-1，可得，$v1 = v2$。
- $b1 < b2$。首先，根据引理6-2可得，$SafeAt(b2, v2)$ 成立。其次，将 *SafeAt* 定义展开，不难得出，存在某个多数派 $Q1$，$\forall p \in Q1 : VotedForIn(p, b1, v2) \vee WontVoteIn(p, b1)$ 成立。接着，由 $ChosenIn(b1, v1)$ 得出，存在多数派 $Q2$，$\forall q \in Q2 : VotedForIn(q, b1, v1)$ 成立。最后，多数派的定义保证了任何两个多数派之间都有交集，即 $\exists p \in Q1 \cap Q2 : VotedForIn(p, b1, v2) \wedge VotedForIn(p, b1, v1)$，根据引理6-1可得 $v1 = v2$。

下面是我们使用 TLAPS 证明引理6-3的代码。第一行我们通过 SUFFICES 关键字将证明目标转换，<1>1 和 <1>2 分别就上述两种情况分类讨论。对于证明过程中出现的存在量词，我们使用了 PICK 关键字来引入（例如，PICK $Q1 \in Quorum$）。

```
LEMMA Inv ⟹ Consistency
⟨1⟩ SUFFICES ASSUME NEW b1 ∈ Ballot, NEW v1 ∈ Value,
                    ChosenIn(b1, v1), Inv, NEW b2 ∈ Ballot, b1 ≤ b2,
                    NEW v2 ∈ Value, ChosenIn(b2, v2)
      PROVE  v1 = v2  BY DEFS Chosen, Consistency, Ballot
⟨1⟩1.CASE b1 = b2 BY ⟨1⟩1, VotedOnce DEFS Inv, ChosenIn
⟨1⟩2.CASE b1 < b2
  ⟨2⟩1. SafeAt(b2, v2)  BY VotedInv DEFS ChosenIn, Inv
  ⟨2⟩2. PICK Q2 ∈ Quorum :
            ∀ p ∈ Q2 : VotedForIn(p, b1, v2) ∨ WontVoteIn(p, b1)
    BY ⟨1⟩2, ⟨2⟩1 DEFS SafeAt, Ballot
  ⟨2⟩3. PICK Q1 ∈ Quorum : ∀ p ∈ Q1 : VotedForIn(p, b1, v1)
    BY  DEF ChosenIn
  ⟨2⟩4. QED  BY ⟨2⟩3, ⟨2⟩2
⟨1⟩ QED  BY ⟨1⟩1, ⟨1⟩2 DEFS Ballot
```

6.3.2 *Spec* ⇒ □*Inv*

引理6-4的证明是证明框架中最重要也是最复杂的一步。这部分证明主要有两个难点：

- 第一，目标引理包含时序逻辑。□*Inv* 为真当且仅当在 *Spec* 刻画的的所有状态下，*Inv* 都为真。

- 第二，*Inv* 的定义复杂。注意到 *Inv* 格式为三个子不变式的合取。同时，子合取式 *MsgInv* 及 *AccInv* 均包含多个合取式。这意味着所有的子合取式都要证明。

针对第一个难点，TLAPS 证明时序逻辑有一个使用归纳推理的系统证明方法。以证明引理6-4为例，其中 $Spec \triangleq Init \wedge \Box[Next]_{vars}$：

- 基础步骤，即证明 $Init \Rightarrow Inv$。在该情况下，变量 *state* 中所有分量都初始化为 $\langle -1, -1, None\rangle$，变量 *msgs* 初始化为空。不难论证，*MsgInv*、*AccInv* 以及 *TypeOK* 都成立。在 TLAPS 中，该引理的证明不需要引入额外的步骤，只需要将 *Init*、*Inv*、*MsgInv*、*AccInv* 以及 *TypeOK* 的定义向下迭代展开，直到展开的表达式中只包含和变量相关的定义。
- 归纳步骤，即证明 $Inv \wedge [Next]_{vars} \Rightarrow Inv'$。左侧的表达式是归纳假设，右侧的表达式是需要证明的目标。其中，$[Next]_{vars}$ 可以展开成 $Next \vee$ UNCHANGED *vars* 的形式。因此，我们可以分成 $Inv \wedge Next$ 及 $Inv \wedge$ UNCHANGED *vars* 这两种情况讨论。

```
LEMMA Invariant ≜ Spec ⟹ □Inv
⟨1⟩1. Init ⟹ Inv
  BY DEFS Init, AccInv, InitState, VotedForIn, MsgInv, TypeOK, Inv
⟨1⟩2. Inv ∧ [Next]_vars ⟹ Inv'
  ⟨2⟩ SUFFICES ASSUME Inv, [Next]_vars PROVE Inv'
     OBVIOUS
  ⟨2⟩1.CASE Next
    ⟨3⟩1. TypeOK'
      ⟨4⟩1. ASSUME NEW p ∈ Participant, NEW b ∈ Ballot, Prepare(p, b)
            PROVE TypeOK'
      ⟨4⟩2. ASSUME NEW p ∈ Participant, NEW b ∈ Ballot, NEW v ∈ Value,
                Accept(p, b, v), Inv
            PROVE TypeOK'
      ⟨4⟩3. ASSUME NEW p ∈ Participant, OnMessage(p), Inv
            PROVE TypeOK'
    ⟨3⟩2. MsgInv'
    ⟨3⟩3. AccInv'
  ⟨2⟩2.CASE UNCHANGED vars
    BY ⟨2⟩2 DEFS AccInv, MsgInv, TypeOK, VotedForIn, Next,
          SafeAt, WontVoteIn, VotedForIn
```

针对第二个难点，我们对 *Inv* 进行拆分，分别证明 *TypeOK*，*MsgInv* 及 *AccInv* 为真。

上面是TLAPS证明引理6-4的框架。其中，递推归纳的UNCHANGED *vars* 情况只需要使用 DEFS 关键字将涉及的谓词定义展开。针对 *Inv*，我们首先将 *Inv′* 拆分成三个子目标 ⟨3⟩1、⟨3⟩2 及 ⟨3⟩3，即分别证明 $Inv \wedge Next \Rightarrow TypeOK'$，$Inv \wedge Next \Rightarrow MsgInv'$ 和 $Inv \wedge Next \Rightarrow AccInv'$。然后，注意到 *Next* 是多个表达式的析取（多个动作）。我们对 *Next* 进行拆分，拆分后每个子目标继续向下分割成三个子目标，即证明 $Inv \wedge Prepare$、$Inv \wedge Accept$ 以及 $Inv \wedge OnMessage$ 分别推出 *TypeOK′*、*MsgInv′*、*AccInv′* 的成立。采取这样的拆分策略，尽管最开始的证明目标被拆分成九个子目标，然而这九个子目标之间是没有相互依赖关系的。换句话说，这几个子目标的证明可以独立完成，并且子目标可以当成事实被其他证明过程引用。

在上述的证明框架中，最终目标依据不同的动作以及不同的不变式转化成多个子目标。对于每个不变式，不同动作的证明方式是相似的，接下来分别介绍这三个不变式的证明框架。

6.3.3 不变式 *TypeOK′* 的成立

不变式 *TypeOK′* 的证明相对简单，只需要证明变量 *state* 及 *msgs* 始终保持正确的类型及取值范围。接下来以动作 *OnMessage* 展示这一部分的证明框架。

⟨4⟩3. ASSUME NEW $p \in Participant$, $OnMessage(p)$, Inv
PROVE $TypeOK'$
⟨5⟩2. $state' \in [Participant \to [Participant \to State]]$
BY ⟨4⟩3, $UpdateStateTypeOKProperty$DEFS $OnMessage$
⟨5⟩3. $msgs' \subseteq Message$
⟨6⟩1. $[from \mapsto p, to \mapsto \{mm.from\}, state \mapsto (state')[p]] \in Message$
⟨6⟩2.CASE $\vee\ (mm.state)[p].maxBal < (state')[p][p].maxBal$
$\vee\ (mm.state)[p].maxVBal < (state')[p][p].maxVBal$
⟨6⟩3.CASE $\neg(\vee\ (mm.state)[p].maxBal < (state')[p][p].maxBal$
$\vee\ (mm.state)[p].maxVBal < (state')[p][p].maxVBal)$

⟨4⟩3 是证明的目标，即在 *Inv* 成立以及执行了动作 $OnMessage(p)$ 的情况下证明 *TypeOK′* 成立。⟨5⟩2 及 ⟨5⟩3 分别证明了变量 *state* 及 *msgs* 下一状态的类型和取值范围符合约束。TPaxos 协议的各个动作始终约束了变量的类型和取值，因此 ⟨5⟩2 及 ⟨5⟩3 的证明只需要将相关的定义展开即可。

6.3.4 不变式 $MsgInv'$ 的成立

$MsgInv$ 是从消息中提取出来的不变式，并且 TPaxos 规约的几个动作都会生成新的消息，因此证明框架比较复杂。我们在证明过程中使用了很多细化策略，以下是这部分的证明框架。

⟨0⟩ 是证明的目标，即在 Inv 和 $TypeOK'$ 成立以及执行了动作 $Prepare$ 的情况下证明 $MsgInv'$ 的成立。如上所述，Inv 成立及动作 $Prepare$ 能推导出 $TypeOK'$ 的成立。引入 $TypeOK'$ 条件有利于精简证明框架。注意到动作 $Prepare$ 会生成新的消息，而新旧消息的证明方式是不同的，因此证明时我们将消息分成了两大部分，一是旧消息，二是动作生成的新消息。对于不变式 $MsgInv$ 中的全称量词，我们使用 NEW 关键字引入了消息 m，实现了全称量词的实例化。然后对 m 的新旧性质分类讨论，从而实现了消息的划分。其中 ⟨1⟩1 是对旧消息的证明过程，⟨1⟩2 是对新消息的证明过程。

由于 $MsgInv$ 对消息的类型进行分类讨论，因此在证明旧消息性质时仍需要分类讨论。⟨2⟩1 对 Preprae 请求产生的消息进行证明，⟨2⟩2 是 Accept 请求产生的消息的证明过程。总体而言，TPaxos 的动作都不会修改旧消息的内容，因此旧消息取值相关的性质始终保持成立，TLAPS 在证明时只需要简单的将定义展开。特殊的，$MsgInv$ 中使用了谓词 $SafeAt(b, v)$，我们通过引理 $SafeAtStable$ 完成证明。

$$SafeAtStable \triangleq Inv \land Next \land TypeOK' \Rightarrow$$
$$\forall v \in Value, b \in Ballot : SafeAt(b, v) \Rightarrow SafeAt(b, v)'$$

$SafeAtStable$ [①]的提出是必然的。第一，$SafeAt(b, v)$ 是证明 TPaxos 正确性需要满足的两个关键条件之一。$SafeAtStable$ 断言了，一旦 $SafeAt(b, v)$ 成立，则 $SafeAt(b, v)$ 始终成立。$SafeAtStable$ 的成立蕴含 TPaxos 满足条件 $SafeAt(b, v)$。第二，在 $MsgInv'$ 的证明框架中，不同动作的 $SafeAt$ 证明思路是相似的。$SafeAtStable$ 的提出可以减少冗余的证明，精简证明框架。

⟨1⟩2 是新消息的证明过程。对于 $MsgInv$ 的每一条合取式，都需要证明新发送的消息 m 满足其性质。其中，⟨2⟩1、⟨2⟩2、⟨2⟩5、⟨2⟩6、⟨2⟩7 只需要将相关的定义展开。特殊的，$Prepare$ 动作和 $Accept$ 动作分别产生 Prepare 和 Accept 消息，这两个动作的证明只需要分别证明第六条及第七条合取式。在 $OnMessage$

[①] $SafeAtStable$ 的证明过程比较复杂，详见附录。

⟨0⟩ ASSUME NEW $p \in Participant$, NEW $b \in Ballot$, $Prepare(p, b)$,
Inv, $TypeOK'$
PROVE $MsgInv'$
⟨1⟩ SUFFICES ASSUME NEW $m \in msgs'$
PROVE $MsgInv!(m)'$ OBVIOUS
⟨1⟩ DEFINE $mm \triangleq [from \mapsto p,\ to \mapsto Participant \setminus \{p\},\ state \mapsto state'[p]]$
⟨1⟩1.CASE $m \neq mm$
⟨2⟩b. $m.state[m.from].maxBal \in Ballot$
⟨2⟩c. $m.state[m.from].maxBal \geq m.state[m.from].maxVBal$
⟨2⟩d. $\forall\, q \in Participant : \land\, m.state[q].maxVBal \leq state'[q][q].maxVBal$
$\land\, m.state[q].maxBal \leq state'[q][q].maxBal$
⟨2⟩1.CASE $(m.state)[m.from].maxBal \neq (m.state)[m.from].maxVBal$
⟨3⟩1. $m.state[m.from].maxBal \leq state'[m.from][m.from].maxBal$
⟨3⟩2. $\lor\ \land\, (m.state)[m.from].maxVVal \in Value$
$\land\ \ldots$
⟨3⟩3. $\forall\, c \in (m.state)[m.from].maxVBal + 1\ ..\ (m.state)[m.from].maxBal - 1 :$
⟨2⟩2.CASE $(m.state)[m.from].maxBal = (m.state)[m.from].maxVBal$
⟨3⟩1. $\lor\ \land\, (m.state)[m.from].maxVVal \in Value$
$\land\ \ldots$
⟨3⟩2. $SafeAt(m.state[m.from].maxVBal,\ m.state[m.from].maxVVal)'$
⟨3⟩3. $\forall\, ma \in msgs' :$
$(ma.state[ma.from].maxBal = m.state[m.from].maxBal$
$\land\, ma.state[ma.from].maxBal = ma.state[ma.from].maxVBal)$
$\implies ma.state[ma.from].maxVVal = m.state[m.from].maxVVal$
⟨1⟩2.CASE $m = mm$
⟨2⟩1. $m.state[m.from].maxBal \neq m.state[m.from].maxVBal$
⟨2⟩2. $m.state[m.from].maxBal \leq state'[m.from][m.from].maxBal$
⟨2⟩4. $\lor\ \land\, (m.state)[m.from].maxVVal \in Value$
$\land\, (m.state)[m.from].maxVBal \in Nat$
$\land\, VotedForIn(m.from,\ (m.state)[m.from].maxVBal,$
$(m.state)[m.from].maxVVal)'$
$\lor\ \land\, (m.state)[m.from].maxVVal = None$
$\land\, (m.state)[m.from].maxVBal = -1$
⟨2⟩5. $\forall\, c \in (m.state)[m.from].maxVBal + 1\ ..\ (m.state)[m.from].maxBal - 1 :$
$\neg(\exists\, v \in Value : VotedForIn(m.from,\ c,\ v))'$
⟨2⟩6. $m.state[m.from].maxBal \in Ballot$
⟨2⟩7. $\forall\, q \in Participant : \land\, m.state[q].maxVBal \leq state'[q][q].maxVBal$
$\land\, m.state[q].maxBal \leq state'[q][q].maxBal$

动作中，这两类消息都有可能生成，因此需要对 m 分类讨论。对于 ⟨2⟩4 中的 *VotedForIn* 谓词，不同的动作证明过程是不同的。*Prepare* 动作不会接受新的提议，因此 *VotedForIn* 当前状态下成立，下一状态也保持成立。*Accept* 动作接受了新的提议，在下一状态 *VotedForIn*′ 成立。*OnMessage* 动作更新值的效果等价于处理 Preapre 请求或者 Accept 请求，因此谓词 *VotedForIn*′ 依然成立。对于 ⟨2⟩5 的全称量词，我们同样使用 NEW 关键字引入值 c（$c \in (curState.maxVBal + 1 \ldots curState.maxBal - 1)$）。然后，根据 *AccInv* 的第四条合取式，我们可以证明当前状态下 $\neg VotedForIn(m.from, c, v)$ 的成立。显然，该谓词在下一个状态依然成立。

6.3.5 不变式 *AccInv*′ 的成立

AccInv 是从参与者维护的状态向量中提出的不变式，TPaxos 规约的各个动作更新值时都需要保持这些不变式。以下是证明 *AccInv*′ 的框架。

⟨4⟩2. ASSUME NEW $p \in Participant$, NEW $b \in Ballot$, NEW $v \in Value$,
$Accept(p, b, v)$, Inv
PROVE $AccInv'$
⟨5⟩ DEFINE $nm \triangleq [from \mapsto p, to \mapsto Participant \setminus \{p\}, state \mapsto state'[p]]$
⟨5⟩4. $\forall a \in Participant$:
$\land\ state'[a][a].maxVBal = -1 \equiv state'[a][a].maxVVal = None$
$\land\ \forall q \in Participant : state'[a][q].maxVBal \leq state'[a][q].maxBal$
⟨5⟩5. $\forall a \in Participant : state'[a][a].maxVBal \geq 0$
$\implies VotedForIn(a, state[a][a].maxVBal, state[a][a].maxVVal)'$
⟨5⟩6. $\forall a \in Participant : \forall c \in Ballot : c > state'[a][a].maxVBal$
$\implies \neg\exists vv \in Value : VotedForIn(a, c, vv)'$
⟨5⟩7. $\forall a, q \in Participant$:
$\land\ state'[a][a].maxBal \geq state'[q][a].maxBal$
$\land\ state'[a][a].maxVBal \geq state'[q][a].maxVBal$
⟨6⟩1.CASE $\neg(q \neq a \land a = p)$
⟨6⟩2.CASE $q \neq a \land a = p$
⟨5⟩8. $\forall a, q \in Participant : state'[a][q].maxBal \in Ballot$
$\implies \exists m \in msgs' : \land\ m.from = q$
$\land\ m.state[q].maxBal = state'[a][q].maxBal$
$\land\ m.state[q].maxVBal = state'[a][q].maxVBal$
$\land\ m.state[q].maxVVal = state'[a][q].maxVVal$

我们采取的策略是对 *AccInv* 的每一条合取式进行证明。以动作 *Accept* 为例，⟨4⟩2 是证明的目标，即在谓词 *Inv* 成立以及执行了动作 *Accept* 的情况下证明 *AccInv′* 的成立。动作 *Accept* 更新状态分量 (m, P) 时，始终保证 $P.b \leq m$，并且提议 P 的分量要么均为初始值 $(P.b = -1 \land P.v = None)$，要么均不为初始值。相似的，动作 *Prepare* 及 *OnMessage* 同样满足这个属性。因此，TLAPS 在证明 ⟨5⟩4 只需要简单的将相关的定义展开。TPaxos 协议保证，参与者只有接受新的提议是才会更新 P，⟨5⟩5 的证明同样只需要把相关定义展开。⟨5⟩6 的证明思路是，当前状态满足 $\forall c > state'[a][a].maxVBal, v \in Value : \neg VotedForIn(a, c, vv)$，又因为动作 *Accept* 接受了提议编号为 $state'[a][a].maxVBal$ 的提议，故 ⟨5⟩6 成立。⟨5⟩7 是证明参与者的观察状态永远不新于真实状态，TLAPS 需要分成执行了动作 *Accept* 的参与者和其他参与者两类情况讨论。针对 ⟨5⟩8，由于动作 *Accept* 会发送新的消息，因而 TLAPS 只需要将定义展开。

6.4 证明优化

构建 TPaxos 的证明是复杂困难的。一方面是由于 TPaxos 的“统一性”，状态和消息的统一优化了系统实现，然而给证明带了许多障碍，如证明时需要考虑真实状态和观察状态之间的关系、不同动作发送的消息的不变式证明等。另一方面，TLAPS 工具目前性能不是非常强大，在证明二维数组相关的性质需要消耗较多的时间。针对这些问题，我们在证明过程中做了一些优化。

6.4.1 语法优化

在 TPaxos 规约中，我们使用三元组 ⟨*maxBal*, *maxVBal*, *maxVVal*⟩ 描述参与者的状态。元组能够直观的刻画每个参与者需要维护的值，然而不同访问和更新元组分量的方式会直接影响到 TLAPS 的证明效率。

以谓词 *VotedForIn* 为例，起初，直观上我们使用了 $[from \mapsto a, to \mapsto Acc, state \mapsto [maxBal \mapsto b, maxVBal \mapsto b, maxVVal \mapsto v]] \in msgs$ 的形式。后续我们替换成 $\exists m \in msgs : m.from = a \land m.state.maxBal = b \land m.state.maxVBal = b \land m.state.maxVVal = v$ 的形式。前一种形式 TLAPS 需要预先解析元组的各个分量，而后一种形式直接提供 TLAPS 各分量的值，因此后者在证明过程中效率要由于前者。

更复杂的是，TPaxos 规约使用了一个二维数组的数据结构表示所有参与者

的状态，其中每个数组元素是上述的三元组。动作 $OnMessage(q)$ 会同时更新 q 的真实状态和 q 对 p 观察状态，这要求对应的规约只用一个表达式同时更新 $state[q][q]$ 及 $state[q][p]$。起初，我们逐个更新 $state$ 的各个分量，即

$$\begin{aligned} state' = [state \text{ EXCEPT } &![q][p].maxBal = Max(@, pp.maxBal), \\ &![q][p].maxVBal = Max(@, pp.maxVBal), \\ &![q][p].maxVVal = \ldots, \\ &![q][q].maxBal = \ldots, \\ &![q][q].maxVBal = \ldots, \\ &![q][q].maxVVal = \ldots] \end{aligned}$$

在后续证明过程中，TLAPS 在解析上述表达式上消耗了大量的时间，甚至无法完成一些包含其他前置条件的引理证明。为此，我们将其替换为以下形式

$$\begin{aligned} state' = [&state \text{ EXCEPT} \\ &![q] = [state[q] \text{ EXCEPT} \\ &\quad ![p] = [maxBal \mapsto Max(@, pp.maxBal), \\ &\qquad maxVBal \mapsto Max(@, pp.maxVBal), \\ &\qquad maxVVal \mapsto \ldots], \\ &\quad ![q] = [maxBal \mapsto maxB, \\ &\qquad maxVBal \mapsto maxBV, \\ &\qquad maxVVal \mapsto maxVV]]] \end{aligned}$$

TLAPS 能非常迅速的解析该表达式。尽管这两种方式执行的结果始终一致，后者能有效的降低 TLAPS 工具性能问题带来的影响。

6.4.2 引理结构优化

在 TLAPS 中，引理断言了系统在当前状态跳转至下一状态时，谓词依然保持成立。引理的使用可以带来许多好处。

第一，引理可以提高证明效率。在 TLAPS 的证明框架里，子目标的证明会同时引用所有上层目标的事实。而随着证明目标不断地细化，后续的子目标可能会使用大量的无关事实，给证明带来了大量时间消耗。在这种情况下，我们将相应的子目标封装成引理，精简了使用的事实，进而提高证明的效率。

第二，引理可以精简证明。我们的证明框架中有许多子目标的证明思路是

相似的，如6.3.4节中的 *SafeAtStable* 相关的证明。使用引理对这些相同点进行封装可以减少大量冗余的证明，精简证明框架。

引理具有很好的复用性，在 TPaxos 的证明里，我们总共定义了 9 个引理，它们在 21 个地方被使用。

6.5 小结

本章使用 TLAPS 严格证明了 TPaxos 的正确性，TPaxosAP 的证明过程类似，故不再赘述。整体上，我们复用已有的 Paxos 证明框架[42]。首先，我们引入了类型不变式、消息不变式与动作不变式。接着，我们将证明目标转化为引理6-3与引理6-4的证明。前者证明相对简单。针对后者，我们对证明目标中的 *Inv* 以及动作进行拆分，将证明目标转化为多个独立子目标的证明。为了简化证明，我们优化了规约的语法，降低了 TLAPS 证明工具性能问题带来的影响。并且，我们引入了 9 个引理，这些引理在多个证明的子部分进行复用，精简了证明框架。

第七章　模型检验实验及 TLC 功能扩展

本章使用 TLC 模型检验工具验证 TPaxos 与 TPaxosAP 协议的正确性，并且验证了从 TPaxos 到 Voting、从 TPaxosAP 到 EagerVoting、从 EagerVoting 到 Consensus 以及 EagerVoting 和 Voting 相互之间的精化关系的正确性。另外，本章还会介绍基于 TLC 开发的一个功能扩展。

7.1　模型检验实验

7.1.1　实验设置

在所有的实验中[①]，我们调整了参与者集合 *Participant*，提议值集合 *Value*，提议编号集合 *Ballot* 的大小，并将前两者设置为对称集[45] 以提高 TLC 的验证效率。我们使用了 10 个线程进行了实验，以下是我们的实验统计结果：已遍历（BFS 方式遍历）的系统状态图的直径，TLC 已检验的所有状态的数量，TLC 已检验的不同状态的数量以及检验时间（格式为 hh:mm:ss）。当 TLC 已检验的不同状态数的数量超过某个阈值（设置为 1 亿[②]），我们就人为地停止该次实验。

实验所用的机器配置：

- 机器 1：2.40 GHz GPU，6 核以及 64GB 内存，TLC 版本号为 1.6.0. 负责 TPaxos 相关实验。
- 机器 2：2.10 GHz GPU，6 核以及 64GB 内存，TLC 版本号 1.6.0. 负责 TPaxosAP 相关实验。

[①]基于 Lamport 给出的从 Voting 到 Consensus 的 TLAPS 证明，我们使用 TLAPS 定理证明系统证明了从 EagerVoting 到 Consensus(https://github.com/Starydark/PaxosStore-tla/blob/master/specification/EagerVoting.tla) 的精化关系的正确性。

[②]以 3 个提议编号，3 个提议值和 3 个参与者为例，此时变量 state 总共有 27 个分量，粗略估计状态数最多有 3 的 27 次方（约为万亿量级）。

7.1.2 验证结果

7.1.2.1 TPaxos 与 TPaxosAP 满足 Consistency

表7-1与图7-2分别给出了在不同配置下验证 TPaxos 及 TPaxosAP 满足一致性的结果。总体而言，在相同配置下，这两种协议的模型检验结果相似。在不同的配置下，参与者数量和提议编号个数对协议的规模影响较大。相对而言，提议值的个数对协议的规模影响较小。

表 7-1: TPaxos 满足一致性的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(2, 2, 2)	20	27809	7991	0 : 00 : 01
(2, 2, 3)	31	69954174	12107912	0 : 07 : 03
(2, 4, 2)	20	27945	7991	0 : 00 : 02
(2, 4, 3)	31	69966456	12107912	0 : 17 : 35
(3, 2, 2)	21	322114689	100000019	0 : 45 : 04
(3, 2, 3)	18	270946706	100000022	0 : 42 : 09
(3, 4, 2)	22	319890347	100000023	2 : 38 : 37
(3, 4, 3)	17	279673321	100000031	2 : 28 : 30

表 7-2: TPaxosAP 满足一致性的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(2, 2, 2)	20	27809	7991	0 : 00 : 01
(2, 2, 3)	31	69954296	12107912	0 : 06 : 52
(2, 4, 2)	20	27945	7991	0 : 00 : 02
(2, 4, 3)	31	69966578	12107912	0 : 18 : 20
(3, 2, 2)	21	329426700	100602814	0 : 44 : 04
(3, 2, 3)	19	274590666	101397482	0 : 42 : 03
(3, 4, 2)	21	323929358	100147908	2 : 40 : 06
(3, 4, 3)	17	278887421	100000024	2 : 28 : 05

7.1.2.2 TPaxos 与 TPaxosAP 满足 Liveness

TPaxos 与 TPaxosAP 的活性要求只有一个提议者，即只允许某个参与者 p 来执行动作 $Prepare(p, b)$ 及 $Accept(p, b, v)$。以 TPaxos 为例，对于 TPaxos 的任何一个阶段，一旦其异常终止（即，没有提议被选中）参与者 p 将会选择增

大提议编号 b 重新开始新的 TPaxos 过程。b 将最终大于所有参与者维护的值 $maxBal$，从而，两个阶段将执行完成并且有提议最终被选中。

在 TLA$^+$ 中，一般使用公平性来验证活性。表达式 $Fairness$ 做出了相应的限制。参与者 p 为唯一的提议者，$MaxBallot$ 是最大的提议编号。参与者 p 选择 $MaxBallot$ 发起 $Prepare(p, MaxBallot)$ 和 $Accept(p, MaxBallot, v)$ 动作，第五条合取式保证了议会 Q 能接受 $Prepare$ 请求和 $Accept$ 请求（参与者 p 是根据本地状态判断能否进入 Accept 阶段，条件 $p \in Q$ 允许 p 和 Q 中的参与者通信以更新本地状态）。表达式 $Liveness$ 定义了 TPaxos 的活性，即最终一定会有值被选中。

$$
\begin{array}{l}
Fairness \triangleq \ \land \exists\, p \in Participant : \\
\qquad \land\ MaxBallot \in Bals(p) \\
\qquad \land\ \mathrm{WF}_{vars}(Prepare(p, MaxBallot)) \\
\qquad \land\ \forall\, v \in Value : \mathrm{WF}_{vars}(Accept(p, MaxBallot, v)) \\
\qquad \land\ \exists\, Q \in Quorum : \\
\qquad\quad \land\ p \in Q \\
\qquad\quad \land\ \forall\, q \in Q : \mathrm{WF}_{vars}(OnMessage(q)) \\
Liveness \triangleq \ \Diamond(chosen \neq \{\})
\end{array}
$$

图 7-3 与图 7-4 分别给出了多种配置下验证 TPaxos 与 TPaxosAP 满足活性的结果。由于使用公平性来验证活性的方法增加了对动作的约束，因此状态数相对而言较少。

表 7-3: TPaxos 满足活性的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh : mm : ss$)
(2, 2, 2)	15	385	275	0 : 00 : 01
(2, 2, 3)	22	27366	13684	0 : 00 : 02
(2, 4, 2)	15	735	523	0 : 00 : 01
(2, 4, 3)	22	54192	27042	0 : 00 : 03
(3, 2, 2)	34	206443509	62284985	2 : 27 : 08
(3, 4, 2)	34	444157073	134403809	6 : 21 : 41

表 7-4: TPaxosAP 满足活性的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(2, 2, 2)	15	385	275	0 : 00 : 01
(2, 2, 3)	22	27372	13684	0 : 00 : 02
(2, 4, 2)	15	735	523	0 : 00 : 01
(2, 4, 3)	22	54204	27042	0 : 00 : 02
(3, 2, 2)	34	206248747	62215595	1 : 17 : 38
(3, 4, 2)	34	442881157	133987469	16 : 28 : 41

7.1.2.3 TPaxos 与 TPaxosAP 的精化

图 7-5、图 7-6 与图 7-7 分别给出了多种配置下验证从 TPaxos 到 Voting、从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 精化关系的正确性。由于该组实验检验的性质包含时序操作符，不再是定义在单个状态上的不变式，验证算法较为复杂，所以需要消耗更多的时间。

表 7-5: TPaxos 精化 Voting 的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(2, 2, 2)	20	27809	7991	0 : 00 : 02
(2, 2, 3)	31	69954174	12107912	0 : 12 : 32
(2, 4, 2)	20	27945	7991	0 : 00 : 02
(2, 4, 3)	31	69966456	12107912	0 : 55 : 24
(3, 2, 2)	21	321158078	100000010	0 : 48 : 36
(3, 2, 3)	17	268108874	100000034	1 : 10 : 50
(3, 4, 2)	22	316078356	100000012	12 : 56 : 45
(3, 4, 3)	17	278338521	100000015	11 : 08 : 10

表 7-6: TPaxosAP 精化 EagerVoting 的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(2, 2, 2)	20	27809	7991	0 : 00 : 01
(2, 2, 3)	31	69954296	12107912	0 : 15 : 42
(2, 4, 2)	20	27945	7991	0 : 00 : 03
(2, 4, 3)	31	69966578	12107912	0 : 59 : 24
(3, 2, 2)	22	315969776	100254890	2 : 09 : 05
(3, 2, 3)	17	267572227	100000032	2 : 11 : 25
(3, 4, 2)	22	318518236	100000012	13 : 13 : 31
(3, 4, 3)	17	278245333	100000036	11 : 31 : 35

表 7-7: EagerVoting 精化 Consensus 的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(3, 2, 3)	16	11831	1352	0 : 00 : 01
(3, 2, 4)	24	124463	12177	0 : 00 : 05
(3, 4, 3)	30	31259	4012	0 : 00 : 04
(3, 4, 4)	31	385395	43631	0 : 00 : 35
(5, 2, 3)	18	883719	22714	0 : 02 : 05
(5, 2, 4)	31	34160291	706304	1 : 31 : 43
(5, 4, 3)	31	2443281	74644	0 : 10 : 26
(5, 4, 4)	31	111615681	2817234	9 : 01 : 06

7.1.2.4 EagerVoting 与 Voting 的等价性

图 7-8与图 7-9 分别给出了多种配置下验证从 Voting 到 EagerVoting 以及从 EagerVoting 到 Voting 精化关系的正确性。

表 7-8: Voting 精化 EagerVoting 的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(3, 2, 3)	9	8447	1352	0 : 00 : 01
(3, 2, 4)	11	82307	12177	0 : 00 : 06
(3, 4, 3)	17	22755	4012	0 : 00 : 03
(3, 4, 4)	21	266123	43631	0 : 00 : 28
(5, 2, 3)	14	583347	22714	0 : 01 : 26
(5, 2, 4)	19	19998859	706304	0 : 55 : 10
(5, 4, 3)	24	1662177	74644	0 : 05 : 19
(5, 4, 4)	31	69744729	2817234	4 : 33 : 53

表 7-9: EagerVoting 精化 Voting 的验证结果

TLC 模型 (参与者数量, 提议值个数, 投票轮数)	状态图直径	状态数	不同状态数	检验时间 ($hh:mm:ss$)
(3, 2, 3)	16	34133	2178	0 : 00 : 02
(3, 2, 4)	24	481511	22095	0 : 00 : 32
(3, 4, 3)	30	137639	6376	0 : 00 : 12
(3, 4, 4)	31	2385315	76963	0 : 03 : 39
(5, 2, 3)	20	1772289	42460	0 : 04 : 33
(5, 2, 4)	31	86520851	1578980	4 : 16 : 02
(5, 4, 3)	31	7092381	136632	0 : 26 : 00
(5, 4, 4)	31	438672481	6087802	7 : 04 : 11

7.2 TLC 的功能扩展

7.2.1 TLC 功能扩展需求

TLC 在模型检验上的功能很强大。首先，能够支持状态空间的完整遍历以及状态空间的可视化。其次，在 simulation 模式下，TLC 能够并发的完成多条单一路径（从根节点到叶子节点）的遍历。最后，对于违背给定不变式的执行流，TLC 不仅仅支持将其输出（可视化），还支持表达式的跟踪。表达式可以是一个状态中变量的函数，也可以是连续的状态中变量的表达式。作为示例，图 7-1展示了 DieHard [①]模型检验产生的状态空间图。其中，右侧是状态空间的可视化，圆圈表示状态，包含所有变量及它们的值。连接两个状态的箭头表示动作，并且这两个是连续的；左上方是 TLC 对于模型检验实验的统计，包括状态空间的最长路径（动作数量最多的路径），状态空间的状态总数以及各个动作执行的总数；左下方是错误路径中表达式的跟踪，即展示了每个状态下被跟踪表达式的值。

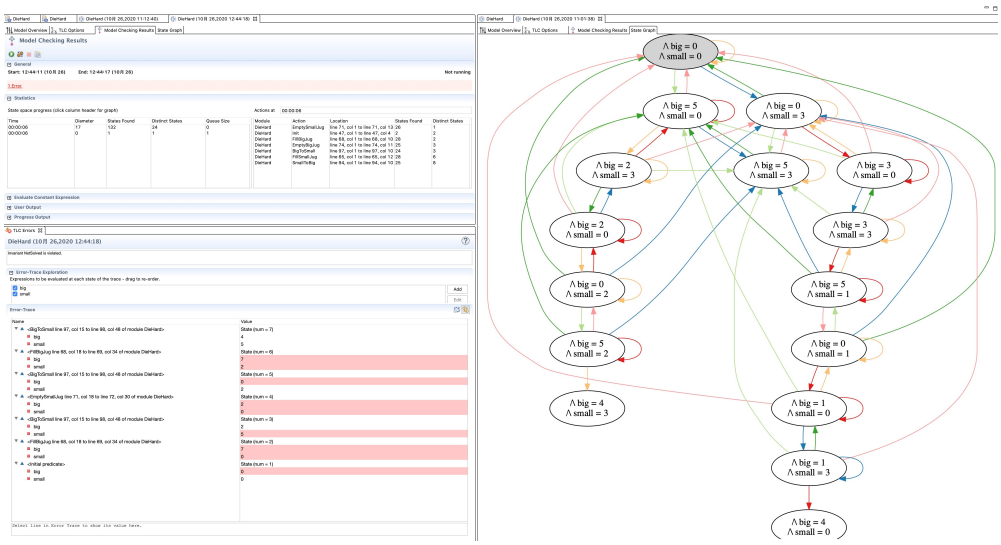

图 7-1: DieHard 模型检验的状态空间图

然而，在我们的实际使用时，经常有人为控制执行动作以检测特定执行流的需求，TLC 并不能很好的支持这一点。为此，我们在 TLC 的基础上，开发了一个能够检测指定执行流的程序。

[①]https://github.com/tlaplus/Examples/blob/master/specifications/DieHard/DieHard.tla

7.2.2 TLC 功能扩展实现

总体上，TLC 由以下四个模块组成：

- 资源管理器（Spec Explorer）：管理所有的规约。类似于文件管理系统，资源管理器中可以访问所有规约。
- 规约编辑器（Spec Editor）：规约的编辑器。包括 TLA+ 的规约编写，PlusCal 语言的编写以及 TLAPS 证明的编写。
- 模型管理器（Model Editor）：包括模型的编辑以及模型检验。模型检验器会遍历模型设定的状态空间以完成不变式或者其他约束的检测。
- 路径追踪器（Trace Explorer）：错误路径的管理。负责错误路径的可视化以及表达式的跟踪等。

其中，我们重点关注模型检验器，模型检验器也包括四个结构：

- FPS(Fingerprint Set)：所有已遍历状态的哈希表。
- SQ(State Queue)：所有未遍历状态的集合。
- WORKER：工作节点，如 cpu 等。
- PROP：规约满足的属性，如不变式等。

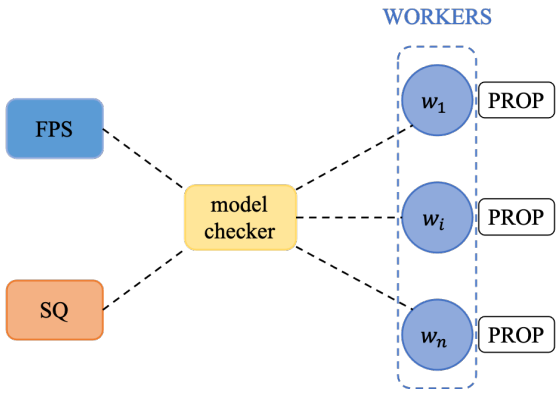

图 7-2: TLC 模型检验器架构

TLC Worker 在遍历状态空间时，首先会将输入的规约进行语义的解析，在判断合法性之后保存变量定义动作定义等信息。接着，Worker 将状态空间抽象成树的数据结构，状态空间的遍历支持广度优先遍历及深度优先遍历，每次遍历都需要进行相应的检测。具体而言，Worker 定位到当前的状态，对这个状态进行性质 PROP 是否成立的检测。然后，获取当前所有可以执行的动作，生成这些动作对应的所有次态。接着，根据已遍历的状态集 FPS 判断次态是否已被

遍历，对于未遍历的状态，Worker 将其保存至相应的数据结构 SQ 中。最后，Worker 递归的对 SQ 中所有次态进行遍历，直到检测出错误或者状态空间遍历完成。

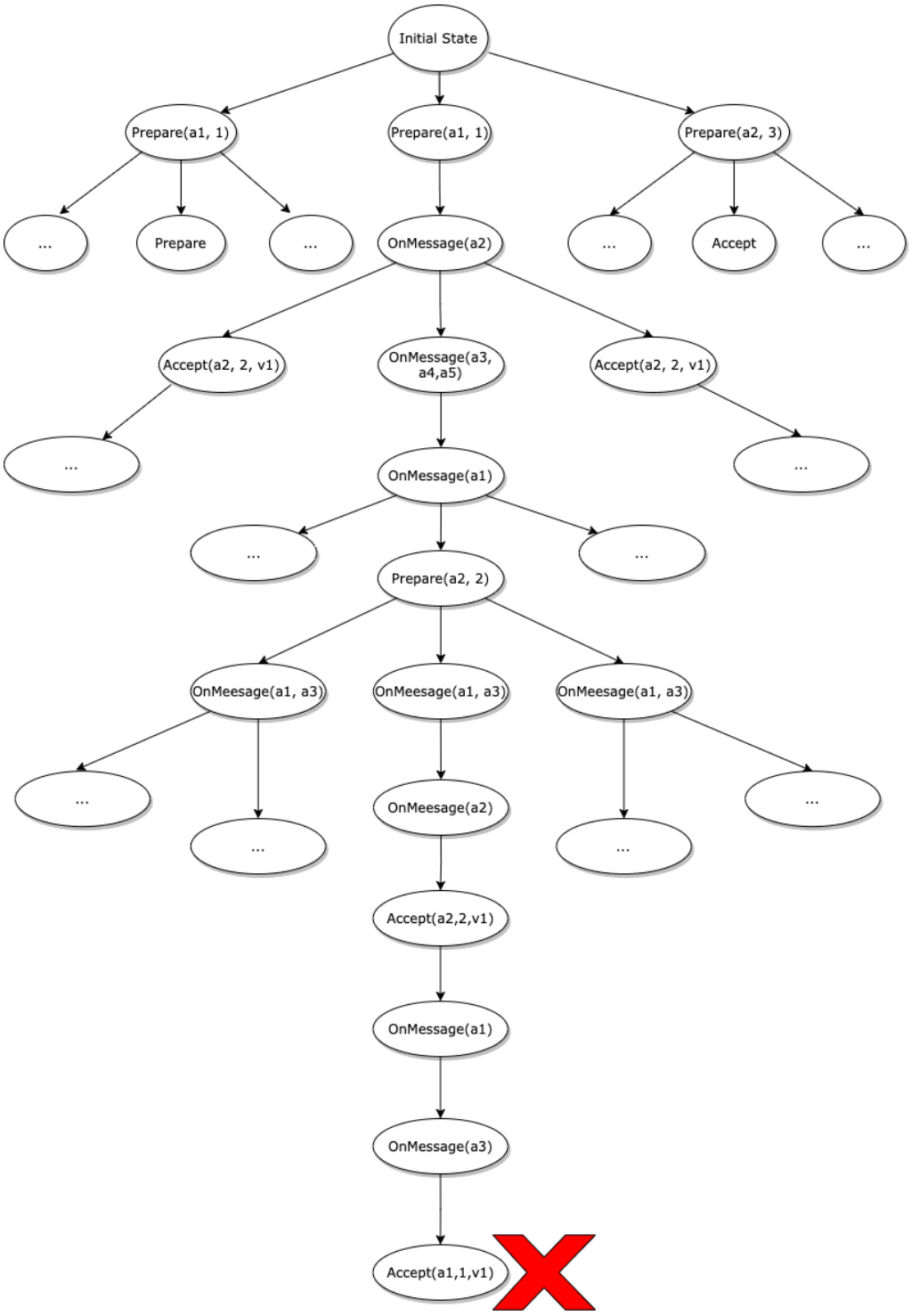

图 7-3: 针对反例 4-1的状态空间图

我们开发的程序对上述流程进行改造，加入人为的选择。在第三步时，程序会将当前状态以及所有可以执行的动作输出。在最后一步，程序将执行选定的动作并且跳转至新的状态。这样，我们可以生成任何指定的执行流。我们使用该扩展功能生成了第四章提到的反例 4-1。图 7-3 是这个反例的状态图。

第八章　总结与展望

本章将总结本工作做出的主要贡献，并且对未来的进一步研究工作进行展望。

8.1　工作总结

本文深入研究了 PaxosStore 系统中的 TPaxos 协议。TPaxos 的新颖之处在于它的“统一性”：它为每个参与者维护统一的状态类型，并采用统一类型的消息进行通信。然而，这种设计方案也带来了 TPaxos 与 Paxos 的诸多差异，给理解 TPaxos 造成了障碍。并且，TPaxos 的正确性尚未经过必要的数学论证或者形式化工具的检验。针对上述几个问题，本文的主要贡献包括以下几个方面：

- 本文从经典的 Paxos 协议出发，论证如何逐步推导出 TPaxos 协议。基于这种推导，我们可以将 TPaxos 看作 Paxos 的一种自然变体。
- 本文给出了 TPaxos 的 TLA$^+$ 规约。在开发规约的时候，我们发现 TPaxos 协议描述中存在至关重要但并未充分阐明的微妙之处：在消息处理阶段，参与者（作为接受者角色）是先作出“不再接受具有更小编号的提议”的承诺（Promise）还是先接受（Accept）提议？这导致对 TPaxos 的两种不同理解，并促使我们提出 TPaxos 的一种变体，称为 TPaxosAP。在 TPaxosAP 中，参与者先接受提议后作承诺。
- 本文使用精化技术（refinement）构建了 Paxos 协议变体间的精化关系。对于 TPaxos，我们仍采用 Paxos 所使用的 Voting 机制，建立了从 TPaxos 到 Voting 的精化关系。对于 TPaxosAP，我们首先提出了一种新的投票机制，称作 EagerVoting。EagerVoting 允许参与者在接受提议的同时，作出比 Paxos/TPaxos 更“激进”的“不再接受具有更小编号的提议”的承诺。然后，我们使用精化技术论证了 EagerVoting 和 Voting 的等价性。接着，我们建立了从 TPaxosAP 到 EagerVoting 以及从 EagerVoting 到 Consensus 的精化关系，并使用 TLC 模型检验工具验证上述精化关系的正确性。我们建立的

精化关系从本质上揭示了经典 Paxos、TPaxos 与 TPaxosAP 核心的一致性，即采用了相同的投票机制 Voting 及其等价的 EagerVoting，这从本质上证明了 TPaxos 以及 TPaxosAP 的正确性。针对其他 Paxos 变体，我们可以采用相同的方法构建精化关系以论证其正确性。

- 本文使用 TLAPS 定理证明系统严格证明了 TPaxos 与 TPaxosAP 的正确性。我们复用已有的 Paxos 证明框架，给出了 TPaxos 与 TPaxosAP 精化 Consensus 的完整证明过程

8.2 研究展望

本文从多个角度论证了 PaxosStore 系统中共识协议 TPaxos 的正确性，未来需要进一步探索的研究方向是验证更多的真实系统，包括验证其共识协议变体的正确性，构建精化关系揭示变体间核心的一致性。我们感兴趣的真实系统包括但不限于以下几个：

- TiDB。TiDB[46] ①是一个开源的 NewSQL 数据库，支持在线事务处理与在线分析处理。TiDB 使用了 raft 保证了数据在存储以及复制过程中的高可用性与安全性。
- Baidu。BRAFT②是百度开源的基于 BRPC 的 Raft 实现。BRAFT 在百度内部被广泛用于构建支持高负载、低延迟及高可用性系统，包括块存储、文件存储、NewSQL 数据库等。
- OceanBase。阿里的 OceanBase③是一个企业级的数据库，支持天猫双 11 及互联网金融等业务。其核心实现了 Paxos 以保证整个系统的高可用。

①https://docs.pingcap.com/zh/tidb/stable
②https://github.com/baidu/braft
③https://github.com/alibaba/oceanbase

参考文献

[1] FISCHER M J, LYNCH N A, PATERSON M S. Impossibility of distributed consensus with one faulty process[J]. Journal of the ACM (JACM), 1985, 32(2): 374 – 382.

[2] HERLIHY M. Wait-free synchronization[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1991, 13(1): 124 – 149.

[3] LAMPORT L, OTHERS. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18 – 25.

[4] LAMPORT L. The Part-Time Parliament[J]. ACM Transactions on Computer Systems, 1998, 16(2): 133 – 169.

[5] CHANDRA T D, GRIESEMER R, REDSTONE J. Paxos made live: an engineering perspective[C] // Proceedings of the twenty-sixth annual ACM Symposium on Principles of Distributed Computing. 2007: 398 – 407.

[6] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 1 – 22.

[7] ISARD M. Autopilot: automatic data center management[J]. ACM SIGOPS Operating Systems Review, 2007, 41(2): 60 – 67.

[8] CAO W, LIU Z, WANG P, et al. PolarFS: an ultra-low latency and failure resilient distributed file system for shared storage cloud database[J]. Proceedings of the VLDB Endowment, 2018, 11(12): 1849 – 1862.

[9] ZHENG J, LIN Q, XU J, et al. PaxosStore: high-availability storage made practical in WeChat[J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1730 – 1741.

[10] VAN RENESSE R, ALTINBUKEN D. Paxos made moderately complex[J]. ACM Computing Surveys (CSUR), 2015, 47(3): 1 – 36.

[11] GAFNI E, LAMPORT L. Disk paxos[J]. Distributed Computing, 2003, 16(1): 1 – 20.

[12] LAMPORT L, MASSA M. Cheap paxos[C] // International Conference on Dependable Systems and Networks. 2004: 307 – 314.

[13] LAMPORT L. Fast paxos[J]. Distributed Computing, 2006, 19(2): 79 – 103.

[14] LAMPORT L. Generalized consensus and Paxos[J]. TechReport, Microsoft Research, 2005.

[15] LAMPORT L, MALKHI D, ZHOU L. Stoppable paxos[J]. TechReport, Microsoft Research, 2008.

[16] LAMPORT L, MALKHI D, ZHOU L. Vertical paxos and primary-backup replication[C] // Proceedings of the 28th ACM Symposium on Principles of Distributed Computing. 2009: 312 – 313.

[17] MORARU I, ANDERSEN D G, KAMINSKY M. There is more consensus in egalitarian parliaments[C] // Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013: 358 – 372.

[18] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm[C] // Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference. 2014: 305 – 319.

[19] RYSTSOV D. CASPaxos: Replicated state machines without logs[J]. arXiv preprint arXiv:1802.07000, 2018.

[20] CLARKE E M, EMERSON E A, SIFAKIS J. Model checking: algorithmic verification and debugging[J]. Communications of the ACM, 2009, 52(11): 74 – 84.

[21] LAMPORT L, MERZ S, D D. A TLA$^+$ specification of Paxos and its refinement[EB/OL]. 2019.
https://github.com/tlaplus/Examples/tree/master/specifications/Paxos.

[22] LAMPORT L. The PlusCal Code for Byzantizing Paxos by Refinement[J]. TechReport, Microsoft Research, 2011.

[23] SUTRA P. On the correctness of Egalitarian Paxos[J]. Information Processing Letters, 2020, 156 : 105901.

[24] ABADI M, LAMPORT L. The existence of refinement mappings[J]. Theoretical Computer Science, 1991, 82(2) : 253 – 284.

[25] LAMPORT L, MERZ S. Auxiliary variables in TLA$^+$[J]. arXiv preprint arXiv:1703.05121, 2017.

[26] LAMPSON B. The ABCD's of Paxos[C] // Proceedings of the twentieth annual ACM Symposium on Principles of Distributed Computing : Vol 1. 2001 : 13.

[27] LAMPORT L. Byzantizing Paxos by refinement[C] // International Symposium on Distributed Computing. 2011 : 211 – 224.

[28] CHARRON-BOST B, SCHIPER A. The heard-of model: computing in distributed systems with benign faults[J]. Distributed Computing, 2009, 22(1) : 49 – 71.

[29] MARIC O, SPRENGER C, BASIN D. Consensus refined[C] // 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 2015 : 391 – 402.

[30] NEWCOMBE C. Why Amazon chose TLA$^+$[C] // International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. 2014 : 25 – 39.

[31] LAMPORT L. The temporal logic of actions[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(3) : 872 – 923.

[32] LAMPORT L. The TLA$^+$ Hyperbook[EB/OL]. 2019. http://lamport.azurewebsites.net/tla/hyperbook.html.

[33] LAMPORT L. Summary of tla+[EB/OL]. 2019. http://lamport.azurewebsites.net/tla/summary-standalone.pdf.

[34] YU Y, MANOLIOS P, LAMPORT L. Model checking TLA+ specifications[C] // Advanced Research Working Conference on Correct Hardware Design and Verification Methods. 1999 : 54 – 66.

[35] YUAN D, LUO Y, ZHUANG X, et al. Simple testing can prevent most critical failures: An analysis of production failures in distributed data-intensive systems[C] // Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation. 2014 : 249 – 265.

[36] CHAUDHURI K C, DOLIGEZ D, LAMPORT L, et al. A TLA+ proof system[J]. arXiv preprint arXiv:0811.1914, 2008.

[37] COUSINEAU D, DOLIGEZ D, LAMPORT L, et al. TLA+ proofs[C] // International Symposium on Formal Methods. 2012 : 147 – 154.

[38] LAMPORT L. How to write a 21st century proof[J]. Journal of fixed point theory and applications, 2012, 11(1) : 43 – 63.

[39] MERZ S, VANZETTO H. Automatic verification of TLA+ proof obligations with SMT solvers[C] // International Conference on Logic for Programming Artificial Intelligence and Reasoning. 2012 : 289 – 303.

[40] MERZ S, VANZETTO H. Harnessing SMT solvers for TLA+ proofs[C] // 12th International Workshop on Automated Verification of Critical Systems (AVoCS 2012) : Vol 53. 2012.

[41] GIFFORD D K. Weighted voting for replicated data[C] // Proceedings of the seventh ACM Symposium on Operating Systems Principles. 1979 : 150 – 162.

[42] LAMPORT L. A TLA+ specification of Paxos Consensus algorithm described in Paxos Made Simple and a TLAPS checked proof of its correctness[EB/OL]. 2019. https://github.com/tlaplus/tlapm/blob/master/examples/paxos/Paxos.tla.

[43] LAMPORT L. A TLA+ specification of Voting algorithm and a TLAPS-checked proof of its correctness[EB/OL]. 2019. https://github.com/tlaplus/tlapm/blob/master/examples/ByzPaxos/VoteProof.tla.

[44] HOWARD H, MORTIER R. A generalised solution to distributed consensus[J]. arXiv preprint arXiv:1902.06776, 2019.

[45] LAMPORT L, MATTHEWS J, TUTTLE M, et al. Specifying and verifying systems with TLA+[C] // Proceedings of the 10th workshop on ACM SIGOPS European workshop. 2002 : 45 – 48.

[46] HUANG D, LIU Q, CUI Q, et al. TiDB: a Raft-based HTAP database[J]. Proceedings of the VLDB Endowment, 2020, 13(12) : 3072 – 3084.

致　　谢

时间匆匆，转眼之间研究生生涯即将结束。在这三年的生活中，尽管遇到许多挫败，但也收获良多。既增长了专业知识，还培养了面对困难积极向上的态度，以及收获了同学之间宝贵的情谊。借此机会，向所有给予我帮助的人表达感谢。

感谢黄宇老师。黄老师在学业上对我谆谆教导，以平等开放的态度对待学生，在生活和学习过程中给了我很大帮助。

感谢魏恒峰老师。魏老师认真负责，言传身教地教给我面对问题的态度，耐心地指导我思考问题以及解决问题的方法。

感谢软件所全体老师对我的关心和帮助。

感谢软件所全体同学。我将铭记与你们一起度过的研究生生涯。

感谢我的父母和兄弟姐妹，感谢你们在人生的方向上提供指引。你们的支持是我前进的动力。

以此留念最后的学生时代。

简历与科研成果

基本信息

易星辰，男，汉族，1996 年 12 月出生，江西抚州人。

教育背景

2018 年 9 月 — 2021 年 6 月 南京大学计算机科学与技术系 硕士
2014 年 9 月 — 2018 年 6 月 南京大学计算机科学与技术系 本科

攻读硕士学位期间完成的学术成果

1. 易星辰, 魏恒峰, 黄宇, 乔磊, 吕建.PaxosStore 中共识协议 TPaxos 的推导、规约与精化. 软件学报,2020,31(8):2336–2361.

攻读硕士学位期间参与的科研课题

1. 面向分布式系统的复制数据类型理论与技术研究，国家自然科学基金青年科学基金项目，No.61702253。

学位论文出版授权书

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》（以下简称“章程”），愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名：_____

_____年_____月_____日

论文题名	PaxosStore 中共识协议 TPaxos 的规约、精化与定理证明				
研究生学号	MF1833088	所在院系	计算机科学与技术系	学位年度	2021
论文级别	☐ 硕士　☑ 硕士专业学位 ☐ 博士　☐ 博士专业学位　（请在方框内画勾）				
作者 Email	staryi@smail.nju.edu.com				
第一导师姓名	黄宇 教授、魏恒峰 助理研究员				

论文涉密情况：

☑ 不保密

☐ 保密，保密期：_____年_____月_____日 至 _____年_____月_____日

注：请将该授权书填写后装订在学位论文最后一页（南大封面）。