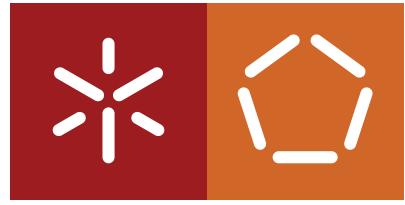


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Cecília da Conceição de Oliveira Soares

**Formalização da reconfiguração de protocolos
de consenso usando Alloy**

outubro de 2022



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Cecília da Conceição de Oliveira Soares

**Formalização da reconfiguração de protocolos
de consenso usando Alloy**

Master dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Manuel Alcino Pereira Cunha

outubro de 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

LICENÇA CONCEDIDA AOS UTILIZADORES DESTE TRABALHO:



CC BY

<https://creativecommons.org/licenses/by/4.0/>

AGRADECIMENTOS

Em primeiro lugar, não há palavras suficientes para agradecer ao Alfrânio por todo o apoio, amor, paciência e companheirismo.

Em segundo lugar, quero agradecer ao Professor Alcino Cunha por ter despertado em mim a vontade de aprofundar o estudo da linguagem Alloy, por ter aceitado orientar esta tese, pela sua disponibilidade e partilha de conhecimento.

Em terceiro lugar, quero agradecer aos meus pais pelo apoio e pela presença constante ao longo de todos os momentos.

Ao longo da licenciatura e do mestrado, várias pessoas tornaram este percurso muito mais agradável. Por isso, quero ainda agradecer aos meus colegas de curso que tornaram esta experiência académica tão divertida e enriquecedora. Um agradecimento especial à Catarina Machado, ao Filipe Monteiro, ao João Pedro Vilaça e ao Luís Abreu pela amizade e pelos bons momentos.

Por último, mas não menos importante, quero agradecer a todos os Professores que me incentivaram, me fizeram questionar e me dedicaram o seu tempo, em especial aos Professores Alcino Cunha, José Bernardo Barros e José Nuno Oliveira.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

ABSTRACT

State machine replication (**SMR**) protocols have a crucial role in distributed systems. At the heart of these protocols are the consensus algorithm, such as *Paxos*, responsible for **SMR**'s consistency. However, modern systems cannot only rely on **SMR** techniques, they must implement reconfiguration strategies, which consist in changing their configurations by adding, removing or replacing their processes. Due to its complexity, implementing a reconfiguration algorithm is error-prone, therefore its specification, validation and verification is advisable.

In this work, we present a specification, in *Alloy*, of the reconfiguration protocol *Vertical Paxos* and the consensus protocol *Paxos*. Besides, we model the *Multi-Paxos* protocol which implements a **SMR**. These three protocols are intrinsically related and, once we are familiar with *Paxos* and *Multi-Paxos*, understanding *Vertical Paxos* becomes straightforward. Nowadays, *Alloy* is one of the most popular specification languages, but little-explored in modeling and analyzing distributed algorithms. As far as we know, there is still no specification of these protocols in *Alloy*.

The aim of this work is to model and validate these protocols, as well as to verify their *safety* properties, in order to obtain confidence in our specifications. Furthermore, we evaluate the performance of different *solvers* and decomposition strategies. Finally, we carry out a brief comparative analysis with **TLA+**.

KEY WORDS : *Alloy, Multi-Paxos, Paxos, SMR, Vertical Paxos.*

R E S U M O

O protocolo de máquinas de estado replicadas ([MER](#)) é uma peça fundamental dos sistemas distribuídos. No centro deste protocolo estão os algoritmos de consenso, como o *Paxos*, usados para manter a consistência das [MER](#). Todavia, os sistemas modernos não podem depender estritamente das técnicas de [MER](#), estes devem também implementar estratégias de reconfiguração. Estas estratégias consistem em alterar a configuração do sistema, adicionando, removendo ou substituindo os processos que o compõem. Dada a sua complexidade, a implementação de protocolos de reconfiguração é muito suscetível a erros, daí que seja aconselhável a especificação, validação e verificação dos mesmos.

No presente trabalho apresentamos uma especificação em linguagem *Alloy* do protocolo de reconfiguração *Vertical Paxos* e do protocolo de consenso *Paxos*. Além destes, modelamos o protocolo *Multi-Paxos*, o qual implementa uma [MER](#). Estes protocolos estão intrinsecamente relacionados e a compreensão do primeiro é facilitada com o conhecimento dos demais. Atualmente, o *Alloy* é uma das linguagens de especificação mais populares, mas pouco explorada na modelação de algoritmos distribuídos e, tanto quanto sabemos, não existe ainda nenhuma especificação dos referidos protocolos em *Alloy*.

O presente trabalho visa modelar e validar os referidos protocolos, bem como verificar as suas propriedades de *safety*, de modo a obtermos confiança nas especificações. Ademais, realizamos uma avaliação de desempenho de diferentes *solvers* e estratégias de decomposição nativas do *Alloy*, bem como uma breve análise comparativa com o [TLA+](#).

PALAVRAS-CHAVE: *Alloy, MER, Multi-Paxos, Paxos, Vertical Paxos.*

CONTEÚDO

Conteúdo	iii
1 INTRODUÇÃO	3
1.1 Objetivos e Contribuições	4
1.2 Estrutura do documento	4
2 CONTEXTUALIZAÇÃO DO PROBLEMA	6
2.1 Definição do Modelo	7
2.2 Problema do Consenso Distribuído	8
2.3 Máquina de Estados Replicada	9
2.4 Reconfiguração	10
3 PROTOCOLOS DE CONSENSO	13
3.1 Paxos	13
3.2 Multi-Paxos	16
3.3 Outras Variantes	18
3.4 Vertical Paxos	20
3.4.1 Vertical Paxos I	22
3.4.2 Vertical Paxos II	24
4 ALLOY	28
4.1 Linguagem Alloy	30
4.1.1 Lógica Relacional	30
4.1.2 Assinaturas e Relações	32
4.1.3 Módulos	33
4.1.4 Axiomas, Predicados, Funções e Asserções	33
4.2 Alloy Analyzer	37
4.2.1 Funcionalidades	40
4.2.2 Relações Auxiliares e o Analyzer	43
4.2.3 Verificação Automática Limitada e Ilimitada	45
4.2.4 Estratégias de Decomposição	46
5 ESPECIFICAÇÃO DOS PROTOCOLOS	48

5.1	Paxos	48
5.1.1	Módulos	48
5.1.2	Assinaturas	48
5.1.3	Axiomas	50
5.1.4	Ações	51
5.2	Multi-Paxos	54
5.2.1	Módulos	55
5.2.2	Assinaturas	55
5.2.3	Axiomas	57
5.2.4	Ações	58
5.3	Vertical Paxos I	60
5.3.1	Módulos	61
5.3.2	Assinaturas	61
5.3.3	Axiomas	64
5.3.4	Ações	65
5.4	Vertical Paxos II	70
5.4.1	Módulos	70
5.4.2	Assinaturas	71
5.4.3	Axiomas	72
5.4.4	Ações	72
6	VALIDAÇÃO E VERIFICAÇÃO DOS MODELOS	77
6.1	Estratégias Gerais de Validação	77
6.2	Representação de Eventos	79
6.3	Validação dos Modelos	82
6.3.1	Paxos	82
6.3.2	Multi-Paxos	83
6.3.3	Vertical Paxos	84
6.4	Verificação dos Modelos	86
6.4.1	Paxos	86
6.4.2	Multi-Paxos	90
6.4.3	Vertical Paxos	91
7	AVALIAÇÃO DE DESEMPENHO	93
7.1	Paxos Benchmarks	94
7.1.1	Análise Comparativa com o TLA+	102
7.1.2	ElectrodX vs. Glucose41	103
7.2	Multi-Paxos Benchmarks	105

7.3 Vertical Paxos Benchmarks	105
8 CONCLUSÃO	110

LISTA DE FIGURAS

Figura 1	Representação do protocolo <i>MER Multi-Paxos</i> .	10
Figura 2	Diagrama classificativo das metodologias de reconfiguração.	11
Figura 3	Etapas do protocolo <i>Paxos</i> .	16
Figura 4	Cronograma do protocolo <i>Paxos</i> e suas variantes.	19
Figura 5	Sequência de fases no protocolo <i>Vertical Paxos I</i> quando existe estado anterior a transferir.	24
Figura 6	Sequência de fases do protocolo <i>Vertical Paxos I</i> quando não existe estado anterior a transferir.	25
Figura 7	Sequência de fases do protocolo <i>Vertical Paxos II</i> quando existe estado anterior a transferir.	26
Figura 8	Sequência de fases do protocolo <i>Vertical Paxos II</i> quando não existe estado anterior a transferir.	27
Figura 9	Instância resultante do comando <i>run Chosen</i> .	37
Figura 10	Editor da linguagem <i>Alloy</i> .	38
Figura 11	Traços da instância do nosso modelo.	39
Figura 12	Representação sucinta do traço.	39
Figura 13	Formas alternativas de representação de uma instância.	40
Figura 14	Personalização de um tema para o nosso modelo.	41
Figura 15	Aplicação da funcionalidade <i>Magic Layout</i> à especificação do <i>Paxos</i> .	41
Figura 16	Exemplo de utilização do <i>Evaluator</i> .	42
Figura 17	Aviso de inexistência de mais instâncias satisfatórias.	42
Figura 18	Barra de ferramentas do <i>Alloy Analyzer</i> .	43
Figura 19	Funcionalidade de projeção de um conjunto.	43
Figura 20	Destaque de relações auxiliares no <i>Alloy Analyzer</i> .	44
Figura 21	Transições da instância em que definimos o escopo das assinaturas.	45
Figura 22	Informação sobre as estratégias de decomposição.	47
Figura 23	Diagrama de atividades da especificação <i>Vertical Paxos I</i> .	66
Figura 24	Diagrama de atividades da especificação <i>Vertical Paxos II</i> .	73
Figura 25	Representação de um evento no <i>Analyzer</i> .	80
Figura 26	Representação dos parâmetros de cada evento como atributos.	81
Figura 27	Contra-exemplo resultante da execução do <i>check singleEvent</i> .	82
Figura 28	<i>Output</i> gerado com os solvers <i>Lingeling</i> e <i>PLingeling</i> nos modos híbrido e paralelo.	94
Figura 29	<i>Log</i> com o erro referente ao solver <i>PLingeling</i> no modo <i>batch</i> .	95

Figura 30	Erro de execução do modo paralelo na versão <i>static messages</i>	96
Figura 31	Erro de execução do modo paralelo na versão <i>static messages</i> após execuções sucessivas.	96
Figura 32	Número de variáveis a tratar pelo <i>Analyzer</i> nas diferentes versões do <i>Paxos</i>	97
Figura 33	Exemplo de resultados inconsistentes nas estratégias híbrida e paralela.	99
Figura 34	Gráfico com os tempos de execução da verificação ilimitada do <i>Alloy</i> e <i>TLA+</i>	103
Figura 35	Exemplo do número de variáveis envolvidas no <i>Vertical Paxos I e II</i>	109

LISTA DE TABELAS

Tabela 1	Resumo da análise performance (em segundos) da especificação <i>Paxos static messages</i> , com 3 participantes.	98
Tabela 2	Resumo da análise performance (em segundos) da especificação <i>Paxos static messages</i> , com 4 participantes.	99
Tabela 3	Resumo da análise performance (em segundos) da especificação <i>Paxos dynamic messages</i> , com 3 participantes.	99
Tabela 4	Resumo da análise performance (em segundos) da especificação <i>Paxos dynamic messages</i> , com 4 participantes.	100
Tabela 5	Resumo da análise performance (em segundos) da especificação <i>Paxos no messages</i> , com 3 participantes.	101
Tabela 6	Resumo da análise performance (em segundos) da especificação <i>Paxos no messages</i> , com 4 participantes.	101
Tabela 7	Resumo da análise performance (em segundos) do <i>Alloy</i> (<i>Paxos no messages</i>), usando a técnica de verificação ilimitada, e do <i>TLA+</i>	103
Tabela 8	Resumo da análise performance (em segundos) da especificação <i>Paxos no messages</i> , com 3 participantes e com o <i>ElectrodX</i> na sua vertente de verificação limitada.	105
Tabela 9	Resumo da análise performance (em segundos) da especificação <i>Vertical Paxos I no messages</i> , com 3 participantes.	107
Tabela 10	Resumo da análise performance (em segundos) da especificação <i>Vertical Paxos I no messages</i> , com 4 participantes.	108
Tabela 11	Resumo da análise performance (em segundos) da especificação <i>Vertical Paxos II no messages</i> , com 3 participantes.	108
Tabela 12	Resumo da análise performance (em segundos) da especificação <i>Vertical Paxos II no messages</i> , com 4 participantes.	109

LISTA DE EXTRATOS DAS ESPECIFICAÇÕES

Extrato 4.1	Sintaxe das assinaturas e relações em <i>Alloy</i>	32
Extrato 4.2	Exemplo de importação de módulos em <i>Alloy</i>	33
Extrato 4.3	Exemplo de axiomas em <i>Alloy</i>	33
Extrato 4.4	Exemplo de predicados em <i>Alloy</i>	35
Extrato 4.5	Exemplo de funções em <i>Alloy</i>	36
Extrato 4.6	Exemplo de asserções em <i>Alloy</i>	36
Extrato 4.7	Exemplo dos comandos <i>check</i> e <i>run</i>	38
Extrato 4.8	Relações auxiliares captadas pela funcionalidade <i>Theme</i>	44
Extrato 4.9	Definição do escopo do comando <i>run</i>	45
Extrato 4.10	Definição do número de passos do comando <i>check</i>	46
Extrato 4.11	Definição de passos ilimitados.	46
Extrato 5.1	Assinaturas da especificação <i>Paxos</i>	49
Extrato 5.2	Axiomas da especificação <i>Paxos</i>	51
Extrato 5.3	Ação de resposta a uma mensagem <i>M1A</i>	52
Extrato 5.4	Predicado <i>safeNoVote</i> da especificação <i>Paxos</i>	53
Extrato 5.5	Predicado <i>safeVotedValue</i> da especificação <i>Paxos</i>	53
Extrato 5.6	Módulos utilizados na especificação <i>Multi-Paxos</i>	55
Extrato 5.7	Assinaturas da especificação <i>Multi-Paxos</i>	56
Extrato 5.8	Algoritmo de seleção dos <i>slots</i> ainda não votados.	59
Extrato 5.9	Envio da mensagem <i>Preempt</i>	60
Extrato 5.10	Assinaturas da especificação <i>Vertical Paxos I</i>	62
Extrato 5.11	Axiomas da especificação <i>Vertical Paxos I</i>	64
Extrato 5.12	Predicado <i>someOneVoted</i> da especificação <i>Vertical Paxos I</i>	67
Extrato 5.13	Predicado <i>completeReconfig</i> da especificação <i>Vertical Paxos I</i>	68
Extrato 5.14	Predicado <i>allValuesSafe</i> da especificação <i>Vertical Paxos I</i>	69
Extrato 5.15	Predicado <i>noPreviousValueChosen</i> da especificação <i>Vertical Paxos I</i>	70
Extrato 5.16	Assinaturas da especificação <i>Vertical Paxos II</i>	71
Extrato 5.17	Predicado <i>completeReconfig</i> da especificação <i>Vertical Paxos II</i>	74
Extrato 5.18	Predicado <i>noOneVoted</i> da especificação <i>Vertical Paxos II</i>	74
Extrato 5.19	Predicado <i>allValuesSafe</i> da especificação <i>Vertical Paxos II</i>	75
Extrato 6.1	Exemplo de ficheiro de teste com uma configuração específica.	78
Extrato 6.2	Cenário de teste com o operador ponto e vírgula.	78

Extrato 6.3	Enumeração dos possíveis eventos da especificação <i>Paxos</i>	79
Extrato 6.4	Função <i>nop_happens</i>	79
Extrato 6.5	Função <i>phase1A_happens</i>	80
Extrato 6.6	Funções <i>phase2A_happens</i> e <i>phase2B_happens</i>	80
Extrato 6.7	Função <i>events</i>	80
Extrato 6.8	Invariante <i>trace</i> e verificação da existência de um único evento em cada passo. .	81
Extrato 6.9	Guarda introduzida nos predicados <i>phase1A</i> e <i>phase2B</i>	82
Extrato 6.10	Assinaturas da especificação <i>Paxos no messages</i>	86
Extrato 6.11	Axioma que impõe a existência de todas as combinações de votos.	87
Extrato 6.12	Predicado <i>safeVotedValue</i> na versão <i>no messages</i>	88
Extrato 6.13	Assinaturas da especificação do <i>Paxos static messages</i>	88
Extrato 6.14	Predicado <i>phase1B</i> na versão <i>static messages</i>	89
Extrato 6.15	Predicado <i>chosenAt</i> nas duas variantes do <i>Vertical Paxos</i>	91
Extrato 6.16	Predicados <i>safeQuorum</i> e <i>safeAt</i> nas duas variantes do <i>Vertical Paxos</i>	92

ACRÓNIMOS

MER Máquina de Estados Replicada. [d](#), [vi](#), [3](#), [4](#), [6](#), [9](#), [10](#), [11](#), [12](#), [17](#), [20](#), [21](#), [55](#), [113](#)

MIT Massachusetts Institute of Technology. [28](#)

OCPU Oracle Central Processing Unit. [93](#)

SAT Solvers And Tools. [29](#)

SMR State Machine Replication. [c](#)

SMV Symbolic Model Verification. [29](#)

TLA+ Temporal Logic of Actions. [c](#), [d](#), [iv](#), [vii](#), [viii](#), [3](#), [4](#), [5](#), [28](#), [93](#), [102](#), [103](#), [112](#)

TLC Temporal Logic Checker. [3](#)

1

INTRODUÇÃO

Os sistemas com alta disponibilidade são fundamentais para garantir a qualidade dos serviços, evitar a insatisfação dos seus clientes e não atrair atenções negativas. Normalmente, para aumentar a disponibilidade do sistema recorre-se à redundância dos seus componentes, sendo uma das técnicas mais comuns a implementação de máquinas de estados replicadas (MER) (Schneider, 1986). Esta garante que, a partir do mesmo estado inicial, diferentes processos cooperam na obtenção de uma sequência de comandos, cuja execução obriga a que todos evoluam para o mesmo estado final. A referida cooperação baseia-se em protocolos de acordo (ou consenso) distribuído, os quais despertam grande interesse no seio académico, o que tem contribuído para uma enorme variedade destes algoritmos (Lamport, 1998; Ongaro and Ousterhout, 2014; Malkhi et al., 2008; Lamport and Massa, 2004; Lamport et al., 2009b; Gafni and Lamport, 2003; Lamport, 2006; Whittaker et al., 2020; Chand et al., 2016; Howard et al., 2017; Moraru et al., 2013; Lamport, 2005; Liskov and Cowling, 2012).

Todavia, ter os estado replicado em diversos processos não é suficiente para garantir um serviço ininterrupto, pois é preciso substituí-los, o que é conseguido através do procedimento da reconfiguração, o qual consiste em alterar a configuração do sistema, adicionando, removendo ou substituindo os processos que o compõem (Lamport et al., 2010). Convém salientar que, com o advento da *Cloud*, democratizou-se a criação de sistemas elásticos que permitem atender a diferentes demandas ao mesmo tempo que mantêm a qualidade do serviço e minimizam os custos. Neste contexto, a reconfiguração assume um papel ainda mais relevante.

No entanto, a complexidade associada aos protocolos de reconfiguração faz com que a sua implementação seja mais suscetível a erros. E com uma técnica de reconfiguração mal concebida, inevitavelmente, o sistema deixará de funcionar. Daí que a especificação, validação e verificação dos algoritmos de reconfiguração sejam recomendáveis.

As linguagens de especificação formal permitem definir, com rigor matemático, a estrutura e o comportamento de um sistema, ajudando a desenvolver e a implementar um programa com maior segurança e fiabilidade. Contudo, a academia não tem dado o protagonismo merecido à formalização e verificação dos protocolos de reconfiguração, não obstante existirem linguagens e ferramentas poderosas para esse efeito.

O *Alloy* (Jackson, 2012) e o *Temporal Logic of Actions* (TLA+) (Lamport, 2003) são duas das linguagens de especificação mais populares. A primeira é uma linguagem de especificação formal baseada em lógica relacional com verificação automática através da sua ferramenta *Analyzer*, ao passo que a última combina lógica temporal com a lógica de ações (Lamport, 1994), dispondo do *model checker* TLC (*Temporal Logic Checker*).

Embora o TLA+ seja a linguagem formal de eleição para especificar protocolos de sistemas distribuídos, as potencialidades do Alloy neste contexto não foram ainda totalmente exploradas. Recentemente, o Alloy passou

por algumas transformações que o tornam mais adequado a formalizar sistemas distribuídos, nomeadamente com a introdução de relações variáveis e propriedades temporais. Ademais, as características do *Analyzer* permitem depurar e explorar a especificação de forma simples e intuitiva, tornando-o bastante apelativo.

1.1 OBJETIVOS E CONTRIBUIÇÕES

Face ao exposto, decidimos especificar em *Alloy* o protocolo de reconfiguração *Vertical Paxos* (Lamport et al., 2009a,b). Em primeiro lugar, escolhemos esta linguagem porque acreditamos ainda não existir nenhuma formalização de protocolos de reconfiguração em *Alloy*. E, em segundo lugar, por a mesma ser flexível, bastante simples e intuitiva, mas muito rica em elementos estruturais e comportamentais, com diversas funcionalidades de visualização, simulação e exploração de diferentes traços de execução (Jackson, 2012).

Optamos por estudar o *Vertical Paxos* por ser um algoritmo genérico e facilmente aplicável a diferentes soluções de *MER*. Além disso, uma vez que o *Vertical Paxos* é uma variante do protocolo *Paxos* e porque está intrinsecamente relacionado com as *MER*, entendemos ser relevante modelar este último, bem como o *Multi-Paxos*, o qual implementa uma *MER*.

Para validar e verificar a correção das especificações dos referidos protocolos foram utilizados diversos mecanismos e validação e de técnicas de verificação automática do *Alloy*. O nosso foco foi a verificação de propriedades de *safety*, de modo a obter mais segurança e confiança nos modelos. Neste âmbito, analisamos também as potencialidades e as limitações do *Alloy* na modelação de sistemas distribuídos.

Por último, porque a eficiência da ferramenta de verificação automática é uma característica importante aquando da escolha da linguagem de especificação, apresentamos também uma avaliação de desempenho de vários *solvers* e estratégias de decomposição nativas do *Alloy* (Brunel et al., 2018), bem como uma breve análise comparativa deste com o *TLA+*.

1.2 ESTRUTURA DO DOCUMENTO

O presente documento está organizado da forma que de seguida se descreve. No capítulo 2 apresentamos a problemática do acordo distribuído e a sua relação com as *MER* e discutimos ainda os diferentes mecanismos de reconfiguração. No capítulo 3 analisamos o protocolo *Paxos* e algumas das suas variantes, com especial destaque para os protocolos *Multi-Paxos* e *Vertical Paxos*. O capítulo seguinte, 4, é dedicado à linguagem de especificação *Alloy* e ao seu instrumento de verificação automática *Analyzer*. Neste contexto, descrevemos a sintaxe e a semântica desta linguagem e enunciamos as funcionalidades e características da referida ferramenta de verificação automática. No capítulo 5 perscrutamos as especificações dos três protocolos supra referidos, analisando as assinaturas, os axiomas e as ações que os integram. No capítulo 6 apresentamos as diversas estratégias de validação utilizadas, os cenários validados e as propriedades das especificações que foram verificadas. Ademais, no capítulo 7 realizamos uma avaliação de desempenho de diversos *solvers* e estratégias de decomposição nativas do *Alloy*, bem como de diferentes versões dos vários protocolos. No mesmo capítulo efe-

tuamos ainda uma análise comparativa do *Alloy* com o [TLA+](#). Por último, no capítulo 8 expomos as conclusões da presente dissertação de mestrado e eventuais trabalhos futuros.

2

CONTEXTUALIZAÇÃO DO PROBLEMA

O consenso é um problema fundamental de sistemas distribuídos que deve ser tratado quando se pretende chegar a um acordo em contextos, tais como: a eleição de um líder, a exclusão mútua, a ordenação total de mensagens, a manutenção de um estado comum, a escolha de uma ação futura, entre outros.

O acordo distribuído está na base da implementação de uma [MER](#) ([Schneider, 1986](#)), sendo esta uma das técnicas mais comuns no desenvolvimento de sistemas com alta disponibilidade. A [MER](#) garante que, a partir do mesmo estado inicial, diferentes processos cooperam na obtenção de uma sequência de comandos, cuja execução obriga a que todos evoluam para o mesmo estado final. Esta cooperação baseia-se em protocolos de acordo (ou consenso) distribuído, sendo o protocolo [Paxos](#) ([Lamport, 1998](#)) uma solução muito elegante.

Ao longo dos anos, foram apresentadas diversas variantes do referido protocolo que visam torná-lo mais eficiente ou especializado. Mas antes de estudarmos este protocolo e os seus sucessores temos de abordar, ainda que de forma sucinta, outras questões complementares que ajudam a compreender melhor o mesmo.

Os protocolos de consenso distribuído que vamos abordar são *indulgentes* ([Guerraoui and Lynch, 2006](#)), i.e., algoritmos distribuídos que toleram falhas de processos, bem como informações imprecisas ou inexatas dadas por um oráculo acerca da correção dos mesmos. Convém salientar que, mesmo havendo um engano quanto à correção dos processos, nunca é violada a propriedade de *safety* do protocolo, ou seja, nada de mau acontece ([Guerraoui and Raynal, 2003](#)). O termo correção utilizado neste contexto refere-se ao comportamento dos processos, ou seja, um processo é correto quando se comporta conforme o esperado.

Ademais, impõe-se distinguir os conceitos de modelo e de especificação. De acordo com Fred Schneider, um modelo de um objeto consiste no conjunto dos seus atributos e as regras que regem como estes interagem ([Schneider, 1993](#)). Todavia, na semântica da *lógica matemática*, o termo modelo está relacionado com a satisfação, sendo que um modelo de uma fórmula é uma avaliação das suas variáveis livres que a torna verdadeira. Por seu turno, o termo especificação representa a descrição do comportamento de um sistema ([van Dalen, 2013](#); [Lamport, 2003](#)). Ao longo do nosso trabalho, utilizamos a noção de modelo com estes três significados, tal como acontece, comumente, na prática.

2.1 DEFINIÇÃO DO MODELO

A modelação de um sistema corresponde a uma representação, ainda que simplificada, deste. Assim, a modelação deve ser tratável e fidedigna, ou seja, deve caracterizar minimamente o sistema em causa e ser suficientemente específica, de modo a representar o problema que queremos solucionar (Schneider, 1993).

Na modelação de um sistema distribuído devemos descrever o comportamento dos processos, unidades do sistema que computam, bem como dos seus canais de comunicação. Note-se que é crucial determinar se se conhece ou não o tempo máximo de processamento e os limites temporais de atraso na entrega de mensagens. No que se refere a estes dois aspectos, os sistemas podem ser classificados como síncronos, parcialmente síncronos ou assíncronos (Cachin et al., 2011).

Nos sistemas síncronos conhecem-se o tempo máximo de processamento e o maior atraso que pode ocorrer na entrega de uma mensagem. Para além disso, cada processo possui um relógio físico local e sabe qual o limite superior que este se desvia do relógio global.

Nos sistemas parcialmente síncronos, as assunções de tempo referidas para os sistemas síncronos apenas se verificam ocasionalmente, mas não se sabe, efetivamente, quando ocorrem ou, quando muito, apenas se sabe que existem limites superiores de processamento e de transmissão, mas não se conhecem os seus valores.

Nos sistemas assíncronos não há quaisquer pressupostos sobre limites temporais das entidades (processos e canais de comunicação), nem quaisquer assunções acerca da existência ou acesso a relógios físicos. Na verdade, afirmar que um sistema é assíncrono é o mesmo que não assumir absolutamente nada relativamente ao âmbito temporal. Este tipo de sistema é tão vago e incerto que, de acordo com Fischer, Lynch, e Patterson (Fischer et al., 1985), não é possível implementar um algoritmo determinístico de consenso num sistema distribuído assíncrono com eventuais falhas de processos. Intuitivamente, esta afirmação explica-se pela impossibilidade de distinguir um processo que falhou de um processo muito lento, dada a inexistência de limites temporais. Porém, satisfeita um conjunto de assunções mínimas é possível chegar a um consenso neste tipo de sistemas (Guerraoui et al., 1999).

Acresce que, os sistemas assíncronos, por serem mais genéricos, impõem menos restrições. Por conseguinte, os algoritmos desenvolvidos tendo por base este tipo de sistemas são passíveis de serem, corretamente, executados em sistemas síncronos e parcialmente síncronos (Aguilera, 2010).

Pelas razões expostas, no que respeita ao modelo, assumimos um sistema distribuído:

- assíncrono com um detetor de falhas inevitavelmente perfeito $\diamond P$ (Chandra and Toueg, 1996);
- com falhas de processos *crash-stop* (Cachin et al., 2011), ou seja, os processos executam corretamente, mas podem, eventualmente, falhar, sendo que quando falham não recuperam;
- com canais de comunicação que não criam, nem corrompem nenhuma mensagem e que permitem que todos os processos corretos, inevitavelmente, recebam as mensagens que lhe são endereçadas;
- sem falhas bizantinas, isto é, falhas decorrentes de processos maliciosos (Pease et al., 1980).

Conforme veremos, todos os protocolos estudados assumem explicita ou implicitamente a existência de um líder. Ora, caso o líder mude frequentemente ou haja suspeitas de ter falhado, o progresso do sistema pode ficar comprometido. Entretanto, um detetor de falhas inevitavelmente perfeito garante que, após determinado período de tempo: a) **nenhum** processo correto é tido como suspeito por outro processo correto; b) e todo o processo que falha é permanentemente considerado suspeito **por todos** os processos corretos. Assim, existirá um momento a partir do qual a propriedade de *liveness* estará assegurada.

De resto, porque a presente dissertação não tem como escopo analisar as propriedades de *liveness*, não vamos aprofundar esta temática.

2.2 PROBLEMA DO CONSENSO DISTRIBUÍDO

Em inúmeras situações, conforme referido anteriormente, os processos precisam de cooperar para chegar a um acordo. Este tipo de cooperação em sistemas distribuídos pode ser reconduzido a um conceito mais abrangente que designamos de *Distributed Consensus* (Lynch, 1996), em português *Consenso Distribuído*. Este é de tal modo importante que, ao longo dos anos, tem sido dissecado pelos especialistas (Lamport, 1998; Ongaro and Ousterhout, 2014; Malkhi et al., 2008; Lamport and Massa, 2004; Lamport et al., 2009b; Gafni and Lamport, 2003; Lamport, 2006; Whittaker et al., 2020; Chand et al., 2016; Howard et al., 2017; Moraru et al., 2013; Lamport, 2005; Liskov and Cowling, 2012).

Um consenso distribuído traduz-se num algoritmo em que os processos, partindo do mesmo estado inicial, inevitavelmente, escolhem (i.e., decidem) um único valor anteriormente proposto (Prisco et al., 2000). Em particular, é importante que o consenso seja obtido mesmo com a existência de um certo número de falhas de processos. Com efeito, em caso de falha de alguns dos processos, os processos corretos devem ser capazes de concordar numa decisão. Daí que este problema esteja intrinsecamente relacionado com os sistemas assíncronos e a tolerância a falhas.

Genericamente, o algoritmo de consenso é caracterizado por dois eventos: a proposta e a decisão. Inicialmente, todos os processos corretos propõem um valor, enviando aos demais a sua proposta. Segue-se a fase da decisão, em que todos os processos corretos acordam sobre o valor a eleger, sendo imprescindível que este seja único e que tenha sido anteriormente proposto. Desta forma, espera-se que o algoritmo de consenso satisfaça, cumulativamente, quatro propriedades (Cachin et al., 2011):

1. Terminação - Todo o processo correto, inevitavelmente, decide algum valor;
2. Integridade - Nenhum processo decide mais do que uma vez;
3. Acordo - Dois processos corretos não decidem de maneira diferente;
4. Validez - O valor decidido tem de ter sido anteriormente proposto.

Enquanto a propriedade de terminação garante que o sistema se mantém ativo, i.e., a propriedade de *liveness*, as restantes três asseguram que não acontece nenhum evento desconforme com o algoritmo de consenso, garantindo a propriedade de *safety*.

De notar que a exigência de acordo permite que um processo incorreto discorde dos processos corretos. Ora, por vezes, é necessário evitar que processos incorretos propaguem as suas decisões, visto que estas podem comprometer a propriedade de *safety*. Assim, é necessário considerar uma noção mais forte de acordo, *Acordo Uniforme*, exigindo-se que todos os processos, sem exceção, decidam pelo mesmo valor.

- Acordo Uniforme - Todos os processos que decidem, corretos ou não, decidem o mesmo valor.

No presente trabalho, os algoritmos de consenso estudados garantem um *Acordo Uniforme*, dado que a escolha de determinado valor pressupõe a sua legitimidade por parte de um quórum (Vukolic, 2010; Whittaker et al., 2021; Vukolic, 2012; Lamport, 2004). Por norma, a noção de quórum está associada a uma maioria de processos.

2.3 MÁQUINA DE ESTADOS REPLICADA

Um sistema distribuído pode ser definido como um conjunto de processos que colabora para atingir o mesmo fim, mas que se revela ao utilizador como um sistema único e coerente (van Steen and Tanenbaum, 2018). Apesar dos recursos destes sistemas atuarem de forma independente e poderem estar fisicamente distribuídos em várias máquinas, esta dispersão deve ser transparente ao utilizador. Efetivamente, para o utilizador deve ser irrelevante onde, quando e como ocorre a interação com o sistema.

Uma característica importante dos sistemas distribuídos é a noção de falha parcial, i.e., algum ou alguns componente(s) do sistema falha(m), enquanto os restantes continuam a operar corretamente. Esta especificidade obriga a que na construção deste tipo de sistemas deva ser ponderada a sua recuperação rápida e automática, de modo a que a sua disponibilidade e/ou performance não sejam significativamente afetadas¹.

A **MER** é uma abordagem muito utilizada na implementação de sistemas tolerantes a falhas (Schneider, 1986). Esta consiste em replicar o estado do(s) processo(s) (i.e., réplicas) e coordenar as interações entre estes, de forma determinística (Lamport, 2008). Assim, partindo do mesmo estado inicial e efetuando, segundo a mesma ordem, as mesmas transições de estado, todas as réplicas evoluem para o mesmo estado final. Na prática, isto significa que os processos necessitam de chegar a um consenso acerca da ordem de execução dos diversos comandos responsáveis pelas transições de estado.

A Figura 1 pretende ilustrar a **MER** do *Multi-Paxos*, o qual tem na sua base o protocolo de consenso distribuído *Paxos*. Conforme podemos observar todas as máquinas têm um *log*, representação de uma fila ordenada que armazena a mesma sequência de comandos a executar, cujo conteúdo resulta do acordo entre os processos. Cada uma das posições dessa fila tem a designação de *slot*. A redundância da informação torna o sistema mais resiliente, pois caso ocorra a falha de uma réplica é garantido que as restantes contêm uma cópia atualizada do estado. Podemos verificar que os comandos *v1* e *v2* foram já escolhidos e, por isso, foram armazenados no *log* de cada uma das réplicas. O próximo comando a ser proposto e, eventualmente, escolhido é o comando *v3*. Os referidos protocolos de consenso são descritos em detalhe no Capítulo 3.

¹Uma falta é a causa de um erro. Um erro é um evento que pode originar uma falha e esta última ocorre quando um processo se comporta de forma diferente do esperado (van Steen and Tanenbaum, 2018).

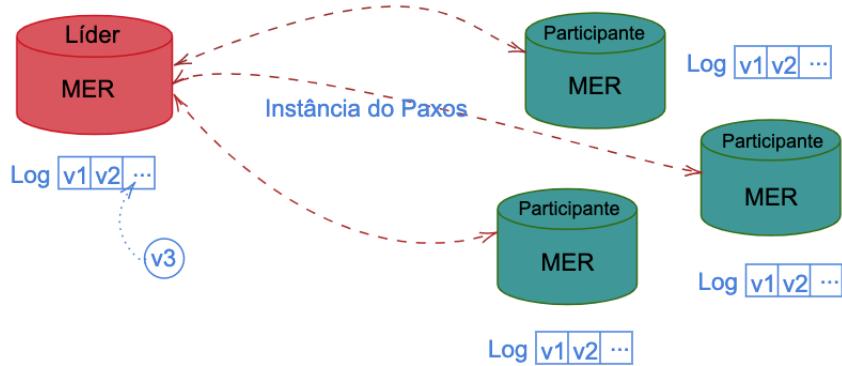


Figura 1: Representação do protocolo **MER Multi-Paxos**.

Acresce que, os sistemas distribuídos tolerante a falhas, geralmente, são altamente dinâmicos, na medida em que os processos podem “entrar e sair”, alterando-se a sua topologia consoante as necessidades de substituição de processos, de escalabilidade e de desempenho do sistema. Assim, o algoritmo de consenso distribuído mostra-se incapaz, por si só, de responder a estas questões. Porém, aliando-se a técnica de reconfiguração ao mecanismo das **MER** obtemos um sistema distribuído mais robusto, capaz de lidar com as exigências de fiabilidade e disponibilidade hodiernas.

A *Cloud* (Asad-ur-rehman et al., 2022; Cao et al., 2018, 2020) é um excelente exemplo de um sistema distribuído que necessita de alta disponibilidade e elasticidade para garantir a qualidade dos serviços prestados. Inserida numa economia de escala, em que diversos recursos de computação, armazenamento e comunicação são compartilhados, a falha de um componente pode afetar não apenas um, mas vários utilizadores.

2.4 RECONFIGURAÇÃO

O procedimento de alterar a configuração de um sistema distribuído, adicionando, removendo ou substituindo os processos que o compõem, é designado de **reconfiguração**. Esta é uma técnica essencial no desenvolvimento das **MER**, pois torna-as mais robustas. Com efeito, a **reconfiguração** não se trata de algo supérfluo na construção de um sistema distribuído, pois qualquer algoritmo de **MER**, sem um protocolo de reconfiguração associado irá, inevitavelmente, falhar (Whittaker et al., 2020).

Inicialmente, a reconfiguração era somente usada para substituir processos que falhavam durante a sua execução. Porém, nos últimos anos, esta tem vindo a ganhar destaque, tentando dar resposta às atuais exigências de elasticidade e mutabilidade dos sistemas, como acontece com a *Cloud* e a *Internet of Things*. Agora, a preocupação passa por adaptar o conjunto de processos do sistema à demanda do mesmo ou balancear a sua carga, deixando de ser uma técnica que meramente reage à falha de um processo.

A abordagem clássica de reconfiguração é utilizar um comando dedicado nas **MER** para esse efeito, sendo a reconfiguração executada na ordem definida no *log*. No entanto, a reconfiguração pode ser executada de

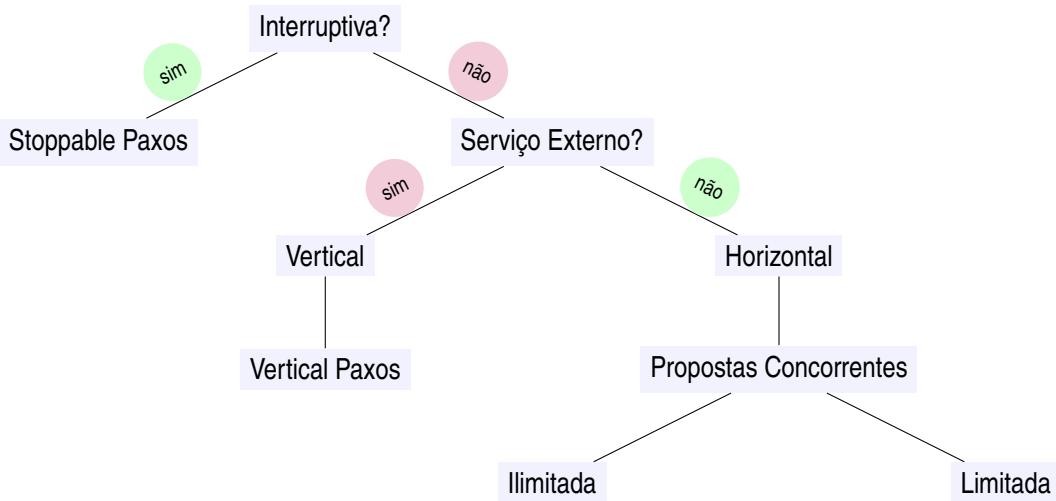


Figura 2: Diagrama classificativo das metodologias de reconfiguração.

outras formas, ver Figura 2. Para melhor compreender e estudar esta técnica caracterizamo-la de acordo com os seguintes critérios:

1. *Interruptiva* ou (*stoppable*) - a execução da MER é interrompida sempre que se inicia um procedimento de reconfiguração (Malkhi et al., 2008).
2. *Não interruptiva* - a técnica de reconfiguração não interrompe a execução da MER. Ambas são executadas em simultâneo.
3. A técnica não interruptiva pode ter um *Serviço de Reconfiguração Externo* - processos responsáveis, única e exclusivamente, pelo procedimento de reconfiguração. Nesse caso, estamos perante uma técnica de *Reconfiguração Vertical* (Lamport et al., 2009a; Whittaker et al., 2020), caso contrário, temos uma técnica de *Reconfiguração Horizontal* (Lorch et al., 2006).
4. Com número *limitado* (Lorch et al., 2006) ou *ilimitado* (Turner, 2016) de comandos propostos e escolhidos concorrentemente (*pipeline limited or ilimited*).

Em primeiro lugar, o método de reconfiguração pode ser caracterizado por parar ou não o processamento da MER. Em caso afirmativo, a reconfiguração do sistema implica sempre o reinício da execução da MER. Caso contrário, estamos perante uma técnica de reconfiguração *não interruptiva*, a qual permite que se execute a reconfiguração enquanto a MER está ativa e continua a decidir comandos. Ademais, a reconfiguração *não interruptiva* pode ainda ser dividida em *vertical* e *horizontal*. Na primeira utiliza-se um grupo específico de processos que auxilia no processamento do algoritmo de reconfiguração. Na segunda, a operação de reconfiguração é inscrita no *log* da MER como qualquer outro comando que deve ser executado. De resto, esta técnica pode ainda ser caracterizada tendo em consideração a quantidade de comandos que podem ser propostos e escolhidos em paralelo, podendo este número ser limitado ou ilimitado.

Neste trabalho pretendemos focar a nossa atenção na revisão bibliográfica e análise formal do protocolo de reconfiguração *Vertical Paxos*, descrevendo as suas duas variantes, *Vertical Paxos I* e *Vertical Paxos II*. Este

protocolo foi pensado por Leslie Lamport, Dahlia Malkhi e Lidong Zhou, em 2009, no artigo *Vertical Paxos and Primary-Backup Replication* (Lamport et al., 2009a). Optamos por estudar esta técnica de reconfiguração porque:

- oferece uma perspetiva teórica sólida para a compreensão dos protocolos de replicação já existentes (Lamport et al., 2009b).
- apresenta-se como uma estratégia mais genérica, podendo ser implementada nas MER que não se baseiam, ao contrário do *Multi-Paxos*, em protocolos que dependem da replicação de *logs* (Whittaker et al., 2020).
- dissocia o processamento das MER do algoritmo de reconfiguração (Lamport et al., 2009a).

3

PROTOCOLOS DE CONSENSO

O Paxos ([Lamport, 1998, 2001](#)) é um algoritmo muito importante, certamente um dos mais conhecidos, na obtenção de um consenso distribuído. Este deu origem a inúmeras variantes e alternativas que se propõem otimizar o protocolo original. Uma delas é o protocolo *Multi-Paxos* ([Lamport, 1998; Chand et al., 2016](#)), utilizado em serviços distribuídos como o *Chubby* ([Chandra et al., 2007](#)) da *Google* ou o *Autopilot* ([Isard, 2007](#)) da *Microsoft*. Outra é o *Vertical Paxos*, um protocolo de reconfiguração que se baseia no *Paxos* para conseguir alterar a configuração do sistema.

No presente capítulo iremos analisar o protocolo *Paxos* e algumas das suas variantes, com especial destaque para as duas supra mencionadas.

3.1 PAXOS

O *Paxos*, proposto por *Leslie Lamport* ([Lamport, 1998, 2001](#)), na década de 90 do século passado, mostra-se uma solução elegante para alcançar um acordo distribuído² No *Paxos*, o consenso é satisfeito quando um único valor, anteriormente proposto, é escolhido.

Este protocolo tem três tipos distintos de intervenientes: *Proposers*, *Acceptors* e *Learners*. O primeiro, durante designado de líder³, é responsável por coordenar o procedimento de votação. Um *Acceptor* ou participante, é um eleitor, i.e., um membro da configuração do sistema com o direito de voto na escolha de um valor. Finalmente, os *Learners*, tal como o nome indica, limitam-se a aprender o valor escolhido.

Para que a escolha de um valor seja bem sucedida, a execução do algoritmo é dividida em rondas, numeradas através de boletins e presididas por um líder pré-determinado. O boletim atribuído a cada ronda é único, assim como o seu líder. Convém salientar que, ao longo deste trabalho iremos utilizar o conceito de ronda e de boletim como sinónimos.

A eleição do líder, num sistema distribuído assíncrono, é uma tarefa difícil, pois esta questão reconduz-se ao problema do consenso distribuído, no sentido em que todos os participantes devem acordar quanto ao líder. A forma como o líder é eleito não nos vai aqui ocupar. No entanto, convém frisar que para haver progresso no *Paxos* é imprescindível designar um líder único capaz de emitir propostas. Caso contrário, num cenário em que

²O seu autor descreve o algoritmo através de uma alegoria muito cativante sobre o funcionamento do parlamento da ilha grega de *Paxos*.

³Em ([Lamport, 2001](#)), Lamport designa o líder de *distinguished proposer*.

vários líderes tentam, continuamente, propor valores com boletins sucessivos, nunca se alcançará um acordo quanto ao valor escolhido.

Assim, a nomeação de um único líder e a correção dos intervenientes são suficientes para que o progresso do algoritmo seja garantido. Note-se que a propriedade de *safety* é salvaguardada, independentemente, do sucesso ou insucesso da eleição de um único líder (Lamport, 2001).

A escolha de um valor tem início com o envio de uma mensagem do líder a um conjunto de participantes. Mas que participantes? Estes participantes são membros de um ou mais *quóruns*. A noção de quórum é um conceito abstrato que define os participantes que têm legitimidade para executar cada fase do protocolo. Na verdade, o valor tem-se por escolhido somente quando um número suficiente de membros de um quórum votam no mesmo boletim e respetivo valor. O critério mais utilizado na determinação desse “número suficiente” é a maioria (Vukolic, 2012), no entanto, existem muitos outros (Whittaker et al., 2021; Vukolic, 2010).

A fim de garantir a consistência, i.e., que apenas um único valor pode ser escolhido (Lamport, 2004), o Paxos também obriga a que todos os quóruns se intersectem, independentemente da ronda ou fase do mesmo. Com o critério da maioria essa condição é garantida, pois quaisquer duas maiorias têm pelo menos um elemento em comum. Desta forma assegura-se que pelo menos um membro que participa na escolha do valor conhece a versão mais atualizada do estado do sistema. Por esse motivo, doravante, iremos assumir um quórum como uma maioria, a menos que seja referido expressamente o contrário. Convém salientar que o quórum é uma representação dinâmica ou variável, ou seja, os participantes que integram o quórum nas diferentes fases deste protocolo podem ser distintos, sem que isso comprometa a sua correção. O termo correção é aqui entendido como salvaguardada das propriedades de *safety* e *liveness* do protocolo (Lamport, 1977).

O Paxos é dividido em duas etapas: a *Fase de Preparação* e a *Fase de Aceitação*. Na primeira fase, o líder envia uma mensagem - *Prepare Request* - a um conjunto de participantes, propondo um boletim b e solicita uma resposta. Os participantes que recebem esta missiva respondem, prometendo não aceitar propostas inferiores a b e indicam o maior boletim, menor do que b , em que votaram anteriormente. Desta forma, o líder fica a conhecer os valores e as rondas anteriormente votados pelos participantes.

No entanto, o participante deve ser criterioso com os *Prepare Requests* a que responde. Imaginemos a situação em que o participante recebe um *Prepare Request* do líder com o boletim b , mas, entretanto, já havia respondido a um *Prepare Request* com boletim maior do que b , prometendo não aceitar propostas com boletins inferiores. Neste caso, o participante não irá responder à proposta com boletim b sob pena de violar a promessa feita anteriormente. Contudo, o participante pode sempre responder a *Prepare Requests* com boletins maiores do que aqueles com que já se comprometeu. Por conseguinte, cada participante terá apenas de guardar o maior boletim a que respondeu nesta fase.

A segunda etapa do protocolo inicia-se assim que o líder recebe, de um quórum, as respostas ao seu *Prepare Request*. A fim de garantir a consistência, o líder terá de encontrar, entre essas respostas, o maior boletim e correspondente valor votados anteriormente por alguns dos participantes. Conhecida esta informação, o líder envia uma nova mensagem, com o boletim b , - *Accept Request* - aos participantes apresentando a votação o valor antes aprendido, caso este exista. No caso de ninguém ter ainda votado, o líder pode propor qualquer valor.

Se o participante receber a mensagem anterior aceita-a, a menos que, entretanto, se tenha comprometido com um boletim maior do que b .

Acresce que, não obstante múltiplas propostas poderem ser escolhidas, isto é, poderem ser aceites por um quórum, é preciso garantir que todas as propostas escolhidas têm o mesmo valor. Desta feita, se uma proposta com valor v é escolhida, então qualquer proposta escolhida com boletim superior deve ter mesmo o valor v associado. A condição anterior implica que se uma proposta com o valor v for escolhida, as propostas posteriores emitidas têm de ter esse mesmo valor, de modo a respeitar as propriedades da consistência, supra referida, e a da *Validade do consenso* (ver Secção 2.2).

Ora, Lamport ([Lamport, 2001](#)) concluiu que, invariavelmente, a consistência é sempre respeitada se, para qualquer proposta com valor v e boletim b , há um quórum, Q , tal que:

- a) nenhum participante pertencente a Q votou em qualquer proposta com boletim menor do que b , ou
- b) v é o valor da proposta com maior boletim, entre todas as propostas com boletins inferiores a b , votada por algum ou alguns membros de Q .

A fim de respeitar o referido invariante, o líder terá de conhecer as propostas menores do que a sua que, porventura, foram anteriormente escolhidas por um quórum. Daí a necessidade de se executar a *Fase de Preparação* deste protocolo.

Assim, desde que observadas todas as restrições referidas, um participante que não interveio na *Fase de Preparação* de determinada ronda pode aceitar o seu correspondente *Accept Request*, pois essa votação em nada afetará a consistência do protocolo.

Falta mencionar como os *Learners* conhecem os valores que foram escolhidos pelo quórum. Existem várias alternativas. Uma possibilidade é um participante comunicar o valor que foi escolhido a um determinado *Learner* e este, por sua vez, propagar a informação para os demais. Mas esta solução é pouco fiável, dado que o *Learner* que conhece o valor pode, entretanto, falhar. Outra alternativa é os participantes comunicarem os valores escolhidos a um conjunto de *Learners*, o que oferece mais garantias de que o valor é aprendido por todos. Numa terceira hipótese, cabe ao líder a responsabilidade de notificar os *Learners*.

Em suma, o *Paxos* descreve o seguinte algoritmo:

1. Fase 1 - Preparação

- a) O líder envia uma proposta, com o boletim b , $M1A(b)$, para um conjunto de participantes - *Prepare Request*.
- b) O participante, a , recebe a mensagem anterior e verifica se b é maior do que qualquer outro boletim com que já se tenha comprometido. Em caso afirmativo, responde com a mensagem $M1B(b, a, mbal, mval)$, prometendo não aceitar nenhum *Prepare Request* com boletim inferior a b e indica qual a maior proposta em que votou anteriormente, $mbal$, e qual o seu valor associado, $mval$, caso exista.

2. Fase 2 - Aceitação

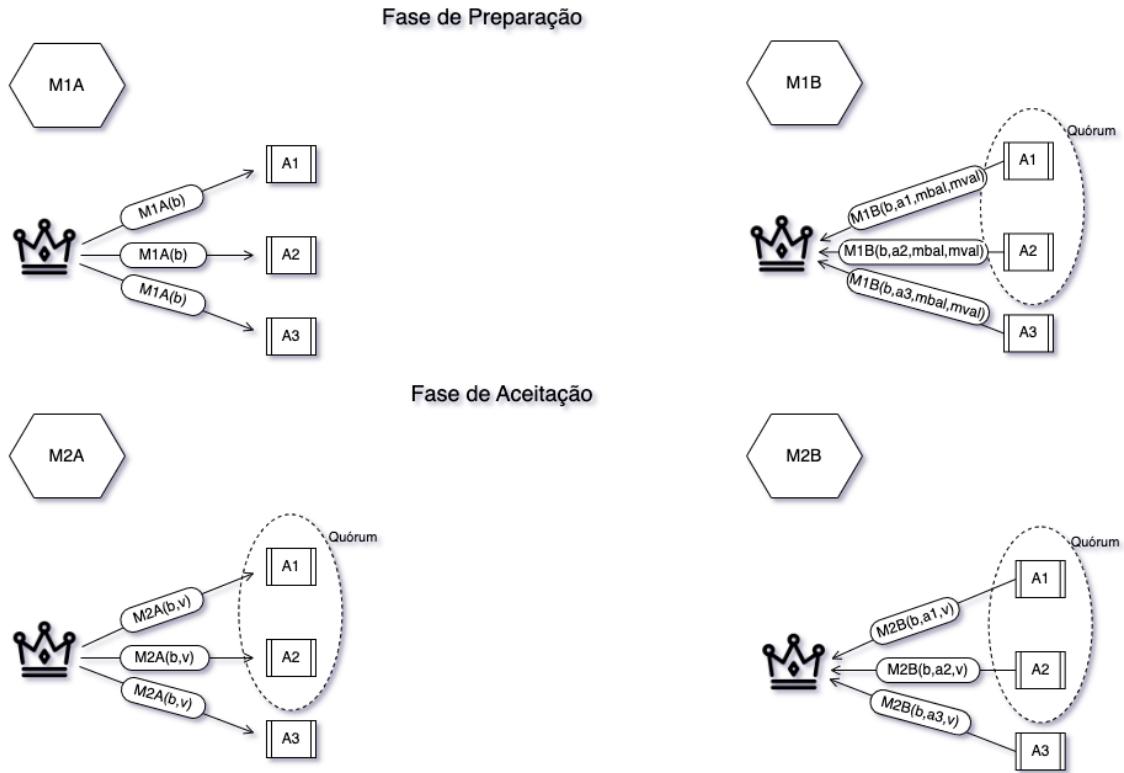


Figura 3: Etapas do protocolo *Paxos*.

- Se o líder receber a resposta ao seu *Prepare Request* de um quórum, então envia um *Accept Request*, $M2A(b, v)$, com o boletim b e o valor v , igual ao constante do maior boletim que foi anteriormente votado por algum desses participantes, ou qualquer valor caso ninguém tenha ainda votado.
- Se o participante receber a mensagem anterior terá de aceitar o valor proposto, v , a menos que tenha, entretanto, respondido a uma mensagem com boletim maior do que b . Esta aceitação verifica-se com o envio da mensagem $M2B(b, a, v)$, sendo a o remetente da mensagem, b o boletim em causa e v o valor aceite.

A fase de aprendizagem não é aqui relevante, pelo que não a incluímos nesta síntese. A Figura 2 ilustra os diferentes momentos do protocolo *Paxos*.

3.2 MULTI-PAXOS

O protocolo *Paxos* descreve um algoritmo que visa o acordo quanto à escolha de um único valor. No entanto, para ser útil, um sistema distribuído deve ser capaz de implementar várias execuções deste protocolo. A variante

do algoritmo original que consegue esse feito é o *Multi-Paxos*⁴. Este considera várias execuções do protocolo original para se chegar a um consenso sobre uma sequência de valores (ou comandos) (Lamport, 1998), possibilitando, portanto, a implementação de uma MER. Esta variante tem os mesmos agentes, com as mesmas funções descritas no Paxos.

Para garantir o progresso do protocolo, todos os participantes respondem a um único líder. Este é o único com a prerrogativa de emitir propostas e decidir em que posição da sequência de execução deve cada valor escolhido ser inscrito (Lamport, 2001).

À sequência de comandos a executar dá-se o nome de *log*, o qual deve ser exatamente igual para todos os intervenientes do protocolo, ver Figura 1. Conforme vimos na Secção 2.3, as MER são determinísticas, pelo que o *log* dos diversos atores é invariável, sob pena de se comprometer o funcionamento da máquina de estados.

Numa execução normal do *Multi-Paxos*, o líder inicia a fase de preparação do *Paxos* e espera pela resposta de um quórum. Se algum participante desse quórum tiver votado anteriormente, então o líder proporá o valor associado ao maior boletim anteriormente votado. Caso contrário, o líder poderá propor qualquer comando. Após ter sido processada a Fase de Aceitação, o valor escolhido é dado a conhecer aos *Learners*.

O que acontece quando se suspeita (ver Secção 2.1) que o líder falhou? Neste caso, é eleito um novo líder que tem de aprender a sequência de comandos que foi decidida até ao presente momento. Porém, podem existir lacunas na sequência aprendida, o que pode impedir o progresso da MER. Suponhamos que o novo líder aprende os comandos referentes às posições 1 a 10, 15 e 16 da sequência. Verificando que desconhece os comandos das posições 11 a 14, executa a Fase de Preparação para estes e para os das posições superiores à 16. Agora imaginemos que nas referidas instâncias do *Paxos*, o líder consegue chegar a um consenso quanto aos comandos para as posições 11 e 17. Ora, existe uma sequência de comandos - do 1 ao 11 - que pode ser executada, mas, apesar dos comandos das posições 16 e 17 serem conhecidos, as posições 12, 13 e 14 estão ainda vazias. O líder pode sempre tentar propor novamente os comandos para as posições que ainda não foram preenchidas. Contudo, essa opção pode ser morosa, até porque pode não haver comandos para serem inseridos nessas posições. Nestas situações, o líder pode propor um comando especial, *no-op*, para as posições a preencher, o qual deixa o estado inalterado, mas permite que haja progresso. Caso o comando *no-op* seja escolhido para as posições 12, 13 e 14, todos os comandos escolhidos podem ser executados, inclusive os comandos das posições 16 e 17.

Mas como surgem, exatamente, estas lacunas? Ora vejamos. O líder pode iniciar a Fase de Preparação do protocolo *Paxos* para novas instâncias, sem terem sido escolhidos todos os comandos precedentes. Conforme referimos, apesar de não conhecer os comandos para as posições 11 a 14, o líder continuou a iniciar novas iterações do *Paxos*, designadamente para a posição 17. Porque cada iteração é uma instância do *Paxos* completamente separada é seguro continuar com novas iterações enquanto se aprende as antigas. Todavia, como o sistema é assíncrono e a ordem das mensagens não está garantida, as diferentes instâncias do *Paxos* podem não se completar ordenadamente. Além disso, as mensagens podem ser perdidas. E, apesar de o líder poder retransmiti-las, o mesmo pode falhar antes da aprovação de um comando proposto, passando, assim, a

⁴O algoritmo é designado em (Lamport, 1998) como *multi-decree parliament*.

existir uma lacuna na sequência de comandos. Note-se que, mesmo na eventualidade de falhas, este protocolo garante que, no máximo, apenas um comando é escolhido para cada posição.

Vimos que, numa fase inicial, o líder tem de executar a Fase de Preparação para todas as instâncias que quer aprender e para as novas instâncias. Ora, esta operação pode revelar-se bastante pesada, tendo Lamport sugerido uma otimização (Lamport, 2001). O líder executa uma única vez a Fase de Preparação, enviando um único *Prepare Request* para aprender todos os valores até então escolhidos. Depois de conhecer o histórico de valores escolhidos, o líder pode omitir a Fase de Preparação nas instâncias subsequentes do *Paxos*. Como a mudança de líder deve ser algo esporádico, executar a esta fase não representa um problema.

De forma genérica este protocolo tem os passos que, de seguida, se descrevem:

1. Fase 1 - Preparação

- a) O líder envia para um conjunto de participantes uma mensagem $M1A(pbal, from)$, com o boletim $pbal$ e remetente $from$.
- b) O participante que recebe a mensagem anterior verifica se o $pbal$ é maior do que qualquer outro boletim com que já se tenha comprometido. Em caso afirmativo, responde com a mensagem $M1B(pbal, from, voted)$, prometendo não aceitar nenhuma mensagem com boletim inferior a $pbal$. Ademais, indica ainda os seus votos anteriores, $aVoted$, caso existam.

2. Fase 2 - Aceitação

- a) Se o líder receber a resposta à sua $M1A$ de um quórum, então envia uma mensagem $M2A(pbal, from, propSV)$ com o boletim $pbal$, remetente $from$ e o conjunto de pares ($Slot, Valor$) propostos, $propSV$. A variável $propSV$ contém, obrigatoriamente, os $slots$ e respetivos valores anteriormente votados pelos elementos do quórum, bem como $slots$ ainda não votados.
- b) Se o participante receber a mensagem anterior tem de a aceitar, a menos que, entretanto, tenha respondido a uma mensagem com boletim maior do que $pbal$. Esta aceitação verifica-se com o envio da mensagem $M2B(pbal, from, propSV)$.

Na prática, o líder somente execute a Fase de Preparação uma única vez, implementando-se a otimização anteriormente referida. Neste caso, os passos consistem, grosso modo, nos descritos em (Konczak et al., 2021). Embora, geralmente, a aplicação do protocolo integre esta otimização, optamos por não a detalhar pois a nossa especificação não a contempla, dado que foi baseada na formalização descrita no artigo (Chand et al., 2016), única deste algoritmo de que temos conhecimento.

3.3 OUTRAS VARIANTES

Após a publicação do protocolo *Paxos*, novos algoritmos tentaram aperfeiçoá-lo. Na Figura 4 podemos observar um cronograma de alguns protocolos que derivam do *Paxos*. De seguida descrevemos, muito sucintamente, alguns desses protocolos.

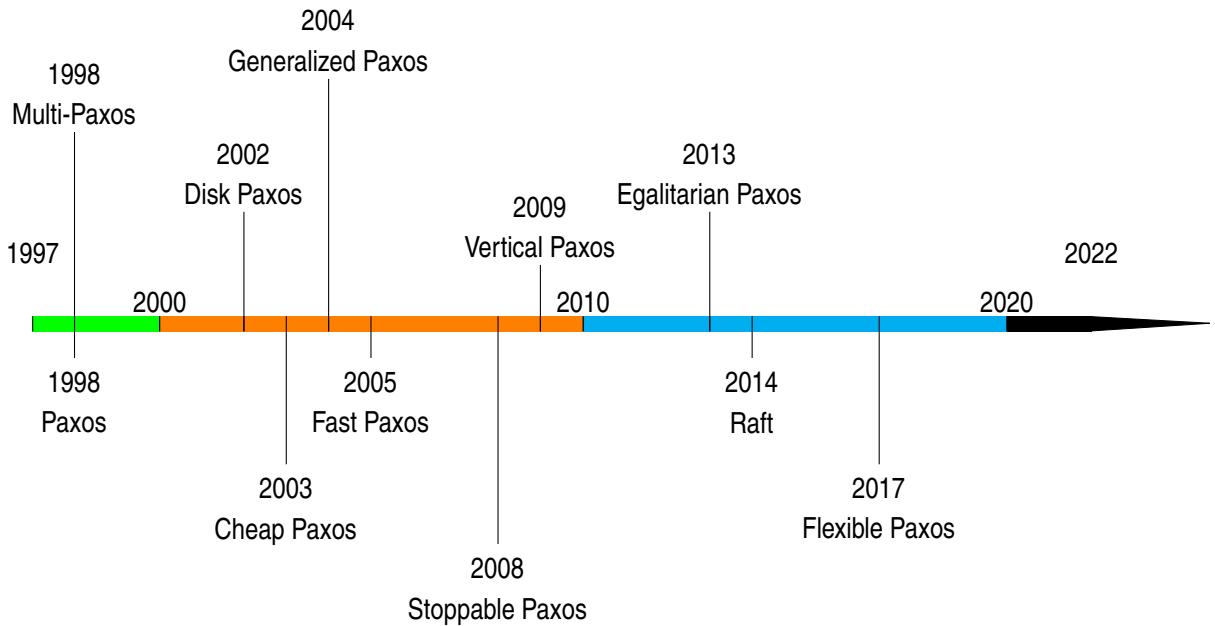


Figura 4: Cronograma do protocolo *Paxos* e suas variantes.

DISK PAXOS *Disk Paxos* é uma variante do protocolo *Paxos* que usa discos como componentes replicados para obter tolerância a falhas. De acordo com esta variante, os participantes são substituídos por discos que permitem operações de leitura e escrita para guardar o estado do protocolo. Há progresso caso o sistema esteja estável e se consiga ler e escrever na maioria dos discos (Gafni and Lamport, 2003).

CHEAP PAXOS Este algoritmo executa o protocolo *Multi-Paxos* com um quórum fixo de $f + 1$ participantes. Além destes, esta variante dispõe de f participantes auxiliares, (*cheap acceptors*), que são utilizados quando os primeiros falham. No caso de haver uma suspeita de falha de um dos participantes é ativado um *cheap acceptor*, de modo a manter-se o número necessário de membros do quórum. Assim, após a reconfiguração do quórum, substituindo-se o participante suspeito por um suplente, este pode continuar a processar normalmente (Lamport and Massa, 2004).

GENERALIZED PAXOS O *Generalized Paxos* permite que comandos independentes sejam executados em qualquer ordem. Assim, em vez de se ordenar uma sequência de comandos, é suficiente ter um conjunto parcialmente ordenado de comandos no qual apenas comandos dependentes são garantidamente ordenados.

Este conjunto parcialmente ordenado designa-se de *command history*. Desta feita, a abordagem da máquina de estados passa a ser generalizada, optando-se por um histórico de comandos em detrimento de uma sequência (Lamport, 2005).

FAST PAXOS O *Paxos* necessita, no mínimo, de três trocas de mensagens para que um valor seja escolhido. O *Fast Paxos* reduz esse número para duas trocas de mensagens quando não há colisão. No caso de haver colisão, i.e., propostas concorrentes de dois valores diferentes para o mesmo *slot*, haverá três trocas de mensagens. Nesta variante, os participantes enviam as suas propostas diretamente para os demais, desprezando o líder. A desvantagem é que são necessários $3f + 1$ participantes para suportar f falhas (Lamport, 2006).

STOPPABLE PAXOS O *Stoppable Paxos* implementa uma **MER** interruptiva, ver Secção 2.4. Este protocolo caracteriza-se por interromper o processamento da **MER** sempre que é necessário executar a reconfiguração do sistema. Depois de completa a reconfiguração é reiniciada a execução da **MER** (Malkhi et al., 2008).

EGALITARIAN PAXOS Esta variante tenta contornar a exigência de um líder único, permitindo que todas as réplicas recebam pedidos dos clientes e proponham comandos. Acresce que, foi o primeiro protocolo a utilizar uma maioria simples ($2f + 1$ para suportar f falhas) e a necessitar, em regra, de apenas uma troca de mensagens (um envio e uma receção) para escolher um valor (Moraru et al., 2013).

RAFT Este protocolo é semelhante ao *Paxos*, tendo como principal diferença a abordagem adotada na eleição do líder. De acordo com o seu autor, Diego Ongaro, este protocolo é mais inteligível, tendo sido aplicadas técnicas específicas que ajudam na sua compreensão, designadamente a separação dos mecanismos de eleição do líder, de replicação de *logs* e da verificação da propriedade de *safety* (Ongaro and Ousterhout, 2014; Howard and Mortier, 2020).

FLEXIBLE PAXOS O *Flexible Paxos* opta pela noção de quórum flexível. Em concreto, este algoritmo enfraquece a condição de que os diferentes quóruns têm de se intersectar, defendendo que a intersecção de quóruns apenas é necessária entre diferentes rondas do algoritmo (Howard et al., 2017).

3.4 VERTICAL PAXOS

Leslie Lamport, Dahlia Malkhi e Lidong Zhou especificaram duas variantes deste protocolo de forma genérica, na linguagem de especificação formal *PlusCal* (Lamport et al., 2009b).

O protocolo de reconfiguração vertical *Vertical Paxos* é executado por cinco atores, que na prática podem não ser necessariamente disjuntos. A saber:

- *clients* - solicitam que um valor (ou comando) seja inserido na sequência de comandos da **MER**;

- *leaders* - gerem a execução do consenso, através da troca de mensagens com os participantes e o *master*;
- *acceptors* - participantes que votam num comando, anteriormente proposto;
- *learners* - aprendem o comando que foi escolhido em cada uma das instâncias da MER e executam-no na respetiva ordem.
- *master* - inicia e controla a reconfiguração, validando-a.

Esta técnica de reconfiguração introduz duas novidades (Lamport et al., 2009a,b):

- A distinção entre quóruns de leitura (*read quorums*) e quóruns de escrita (*write quorums*).
- A introdução do mestre da configuração (*master* ou *auxiliar configuration master*).

A distinção de diferentes quóruns, leitura e escrita, prende-se com a fase do protocolo de consenso em que cada um dos quóruns intervém. Por um lado, o quórum de leitura refere-se ao conjunto de participantes que executa a primeira fase do protocolo, na qual o líder aprende o estado das configurações anteriores e em que os elementos do quórum se comprometem a desconsiderar boletins com números inferiores ao atual. Por outro lado, o quórum de escrita é composto pelos participantes que, efetivamente, votam no valor proposto pelo líder.

De notar que os quóruns de escrita têm de se intersetar com os quóruns de leitura pertencentes à mesma ronda. Ademais, para que se mantenha a correção do protocolo, o líder terá de contactar com os quóruns de escrita da ronda atual, bem como com os quóruns de leitura de uma ou mais rondas anteriores. Estas imposições fazem com que o estado dos quóruns anteriores seja aprendido pelos quóruns sucessores, operação que designamos de transferência de estado.

O caso mais interessante será manter o quórum de leitura com apenas um elemento, o que implica que no quórum de escrita participem todos os processos da configuração. À partida, mantendo-se os quóruns de leitura mais pequenos o protocolo será mais eficiente. Contudo, diminuir o quórum de leitura implica aumentar o quórum de escrita.

Talvez a maior inovação deste protocolo seja a criação do mestre da configuração, separando-se, assim, as decisões que cabem no domínio do algoritmo de reconfiguração das que cabem na execução normal do sistema. O mestre da configuração determina e armazena cada configuração, elegendo o líder e os participantes de cada ronda, atuando à margem e em simultâneo com o algoritmo de consenso. Daí que se diga que a reconfiguração ocorre “verticalmente” através do incremento dos boletins que identificam cada uma das configurações, facto que inspirou o nome do protocolo (Lamport et al., 2009b).

O mestre da configuração somente é chamado a intervir quando há a necessidade de reconfigurar o sistema. Assim, a sua participação deve ser pouco frequente, utilizando-se pouco poder de processamento para o efeito. Na prática, o mestre da configuração é um serviço externo constituído por vários processos cuja responsabilidade prende-se única e exclusivamente com a reconfiguração do sistema. Todavia, não nos vamos aqui ocupar com a forma como o mesmo é implementado, nem com as suas eventuais falhas. Assumimos que este é fiável.

Tal como no protocolo *Paxos*, também no *Vertical Paxos* temos de preservar o invariante de que dois participantes diferentes não votam em dois valores distintos para o mesmo boletim. Neste protocolo isso é conseguido com a ajuda do mestre da configuração que comanda todo o processo de reconfiguração e que garante que existe apenas um líder em cada ronda, o qual, por sua vez, informa os membros do quórum qual o valor que estes podem escolher.

Em traços gerais e de forma sucinta podemos descrever o protocolo de reconfiguração vertical através dos seguintes passos:

1. O mestre da configuração determina o líder, a configuração do sistema e o seu boletim;
2. O líder inicia a Fase de Preparação do protocolo *Paxos*, enviando um *Prepare Request* ao quórum de leitura da configuração que coordena, bem como aos quórums de leitura das configurações das rondas anteriores;
3. Os membros desses quórums de leitura indicam os comandos por si votados em boletins precedentes e, simultaneamente, comprometem-se em não votar em boletins inferiores ao da nova configuração;
4. Recebidas as respostas, o líder está em condições de dar início à segunda fase, enviando um *Accept Request*, com o mesmo boletim, ao quórum de escrita da configuração atual.
5. Os elementos do quórum de escrita que receberem a anterior mensagem têm de a aceitar, a menos que, entretanto, tenham respondido a outra com boletim superior.
6. O líder notifica o mestre da configuração de que a transferência de estado, referente àquele boletim, está completa.
7. A nova configuração é considerada pronta para executar normalmente.

De resto, convém notar que quando não existe estado a transferir, o líder aguarda que algum cliente solicite a execução de um comando, após o que propõe esse valor ao seu quórum de escrita. Neste caso, a ativação da configuração por parte do mestre da configuração ocorre sem que se execute a Fase de Aceitação, visto que não há nenhum estado prévio para aceitar nem transferir.

Ao contrário do que acontecia no protocolo *Paxos*, onde o conjunto de participantes é estático, no *Vertical Paxos* a configuração do sistema é alterada sempre que muda a ronda, pelo que o líder tem de comunicar com os participantes das rondas anteriores. Contudo, as duas variantes deste algoritmo de reconfiguração tratam esta comunicação de forma diferente.

3.4.1 *Vertical Paxos I*

No *Vertical Paxos I*, quando se inicia o processo de reconfiguração, a nova configuração fica imediatamente ativa em simultâneo com a precedente, ou seja, podemos ter várias configurações ativas (Lamport et al., 2009b).

Com efeito, esta variante permite que as configurações precedentes se mantenham em processamento até a reconfiguração se encontrar concluída, passando a nova configuração a ser designada completa. Note-se que uma configuração completa está necessariamente ativa enquanto não existir uma configuração completa posterior.

No entanto, caso o histórico de configurações seja consideravelmente elevado, não é nada eficiente contactar os quóruns de leitura de todas as rondas anteriores sempre que se inicia o mecanismo de reconfiguração.

De forma a otimizar o processo de transferência de estado, uma configuração é considerada obsoleta, deixando de ser necessário trocar mensagens com os seus membros, quando os comandos executados por esta forem conhecidos por uma configuração completa com boletim superior. Este processo de transferência de estado é gerido pelo líder de cada configuração, que terá de conhecer os comandos armazenados nos quóruns de leitura das configurações precedentes ainda ativas para os comunicar ao seu quórum de escrita. Quando o líder sabe que o quórum de escrita fez a transferência de estado, solicita ao mestre da configuração que considere o algoritmo de reconfiguração completo. Caso o mestre da configuração aprove o pedido anterior do líder, na eventualidade de novas reconfigurações, o mestre somente informará os futuros líderes das configurações a partir da completa mais recente.

De acordo com este algoritmo pode acontecer o seguinte cenário: a ronda $b + 1$ é iniciada, contudo, o seu líder falha antes da transferência de estado das rondas anteriores estar completa. Então, o mestre da configuração começa uma nova ronda, $b + 2$, mas o seu líder também falha antes que se complete a transferência de estado. Agora imaginemos que esta situação é recorrente e que estamos já na ronda $b + 50$. Pois bem, de acordo com o *Vertical Paxos I*, o líder da ronda atual tem de comunicar com todas as configurações anteriores ativas, independentemente da grandeza do seu número. Ora, este processo pode ser moroso, pelo que os seus criadores pensaram numa otimização, *Vertical Paxos II*, que explicaremos na Secção 3.4.2.

Em suma, quando há estado a transferir, o *Vertical Paxos I* comprehende as seguintes etapas:

i) Fase 0 - Início da Reconfiguração

- O mestre da configuração envia ao líder uma mensagem $MsgNewBallot(b, completeBal)$, indicando a ronda, b , e a configuração completa mais recente, $completeBal$.

ii) Fase 1 - Preparação

- o novo líder envia um *Prepare Request* para os quórum de leitura das configurações anteriores ativas - mensagem $M1A(b, prevbal)$. A variável $prevbal$ indica o boletim dos quóruns de leitura a que a mensagem se destina;
- os membros desses quóruns respondem, caso não tenham já respondido a uma mensagem com boletim superior, e comprometem-se a não aceitar nenhum boletim inferior ao atual. Desta forma o líder aprende os valores que foram votados nas rondas anteriores - mensagem $M1B(b, acceptor, mbal, mval)$. O *acceptor* refere-se ao remetente da mensagem, o *mbal* ao maior boletim anteriormente votado e o *mval* ao seu valor.

iii) Fase 2 - Aceitação

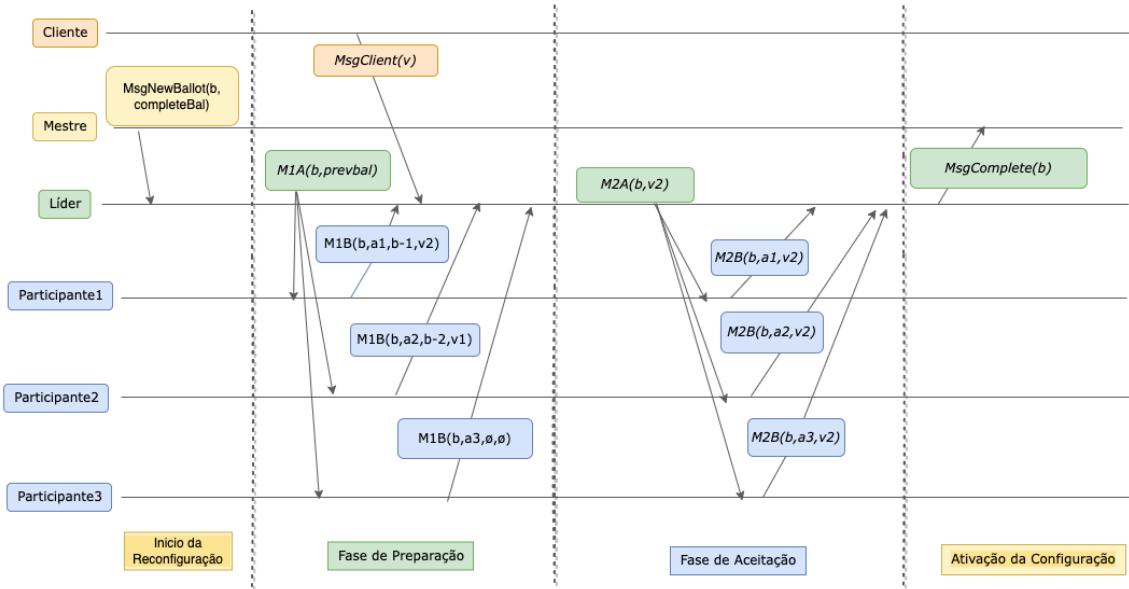


Figura 5: Sequência de fases no protocolo *Vertical Paxos I* quando existe estado anterior a transferir.

- O líder envia a um seu quórum de escrita um *Accept Request* com o estado do sistema - mensagem `M2A(b, v2)`, onde `v2` é o valor a ser transferido para a nova configuração.
- O líder recebe as mensagens de aceitação por parte dos membros do quórum de escrita - mensagem `M2B(b, v2)`.

iv) Fase 3 - Ativação da Configuração

- O líder envia uma mensagem `MsgComplete(b)` ao mestre da configuração, comunicando que a transferência de estado está completa.

Esta sequência de passos pode ser observada na Figura 5, onde o decurso do tempo é representado da esquerda para a direita.

Por último, caso não haja nenhum valor votado anteriormente, a Fase de Ativação da configuração sucede à Fase de Preparação. Conforme se pode observar na Figura 6, depois de realizada a Fase de Preparação, como nenhum valor foi anteriormente votado, o líder envia uma mensagem ao mestre da configuração solicitando que a configuração relativa ao boletim `b` se tenha por completa - `MsgComplete(b)`. Posteriormente, é levada a cabo a Fase de Aceitação, sendo o valor então proposto, `v`, o constante de um pedido de um cliente.

3.4.2 Vertical Paxos II

A segunda variante do *Vertical Paxos* impede que haja dependência de tantas configurações anteriores pois assegura que, em qualquer momento, há somente uma configuração ativa (Lamport et al., 2009b). A nova

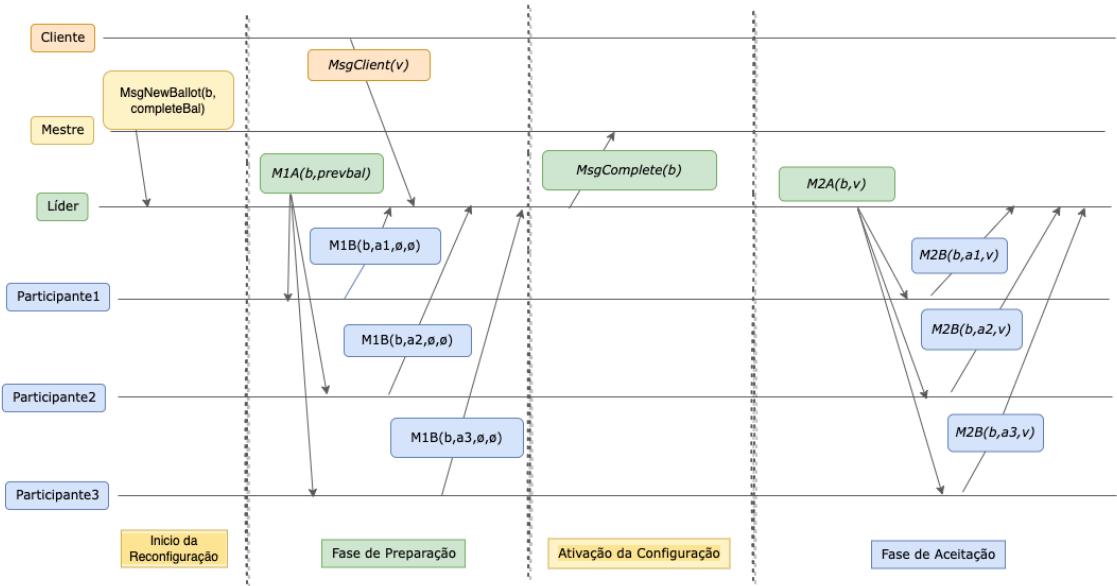


Figura 6: Sequência de fases do protocolo *Vertical Paxos I* quando não existe estado anterior a transferir.

configuração permanece inativa até que o novo líder notifique o mestre da configuração de que transferência de estado foi integralmente efetuada e este último autorize a ativação da nova configuração (Lamport et al., 2009a).

Nesta versão, o mestre da configuração inicia o processo de reconfiguração, elegendo o líder e a nova configuração do sistema, tal como acontece no *Vertical Paxos I*. Contudo, informa o novo líder de que este apenas terá de comunicar com a única configuração que ainda se encontra ativa. Esta variante não distingue entre configuração ativa e completa porque uma configuração ativa está necessariamente completa. Agora, o novo líder tem somente de conhecer uma configuração que não a sua, diminuindo-se assim a troca de mensagens. Depois de ter contactado os quóruns de leitura anteriores e de conhecer o estado atual do sistema, o novo líder propõe a transferência de estado a um seu quórum de escrita e aguarda a confirmação de que a mesma foi bem sucedida. Se tudo correr conforme o esperado, o líder informa o mestre da configuração de que a nova configuração tem já o estado atualizado e pede para que a mesma seja ativada. Depois de receber a mensagem anterior, o mestre ativa a nova configuração e, simultaneamente, desativa a precedente.

No caso de existir estado a transferir, esta variante comprehende as seguintes etapas, as quais podem ser observadas na Figura 7:

i) Fase 0 - Início da Reconfiguração

- O mestre da configuração envia ao líder uma mensagem `MsgNewBallot(b, prevbal)`, indicando a ronda, b , e o boletim da configuração ativa, $prevbal$.

ii) Fase 1 - Preparação

- o novo líder envia um *Prepare Request* para os quóruns de leitura da configuração ativa - mensagem `M1A(b, prevbal)`;

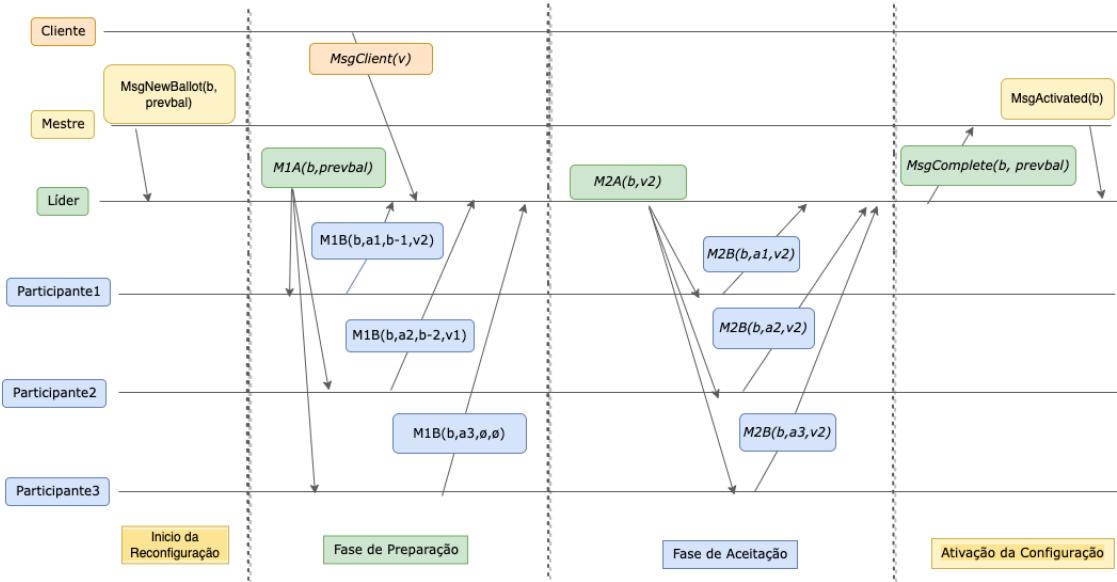


Figura 7: Sequência de fases do protocolo *Vertical Paxos II* quando existe estado anterior a transferir.

- os membros desses quóruns respondem, caso não tenham já respondido a uma mensagem com boletim superior, e comprometem-se a não aceitar nenhum boletim inferior ao atual. Desta forma o líder aprende os valores que foram votados nas rondas anteriores - mensagem `M1B(b, acceptor, mbal, mval)`. O `acceptor` representa o remetente da mensagem, o `mbal` o maior boletim anteriormente votado e o `mval` o seu valor.

iii) Fase 2 - Aceitação

- O líder envia a um seu quórum de escrita um `Accept Request` com o estado do sistema - mensagem `M2A(b, v2)`, onde `v2` é o valor a ser transferido para a nova configuração.
- O líder recebe as mensagens de aceitação por parte dos membros do quórum de escrita - mensagem `M2B(b, v2)`.

iv) Fase 3 - Ativação da Configuração

- O líder envia uma mensagem `MsgComplete(b, prevbal)` ao mestre da configuração, comunicando que a transferência de estado está completa.
- Depois de receber a mensagem anterior, o mestre da configuração notifica o líder de que a sua configuração está agora ativa, enviando-lhe uma mensagem `MsgActivated(b)`. Note-se que esta mensagem não existe na primeira variante do protocolo.

Finalmente, conforme referimos anteriormente, na eventualidade de não existir nenhum valor votado na ronda anterior, a Fase de Aceitação não terá lugar, passando-se imediatamente para a Fase de Ativação da nova con-

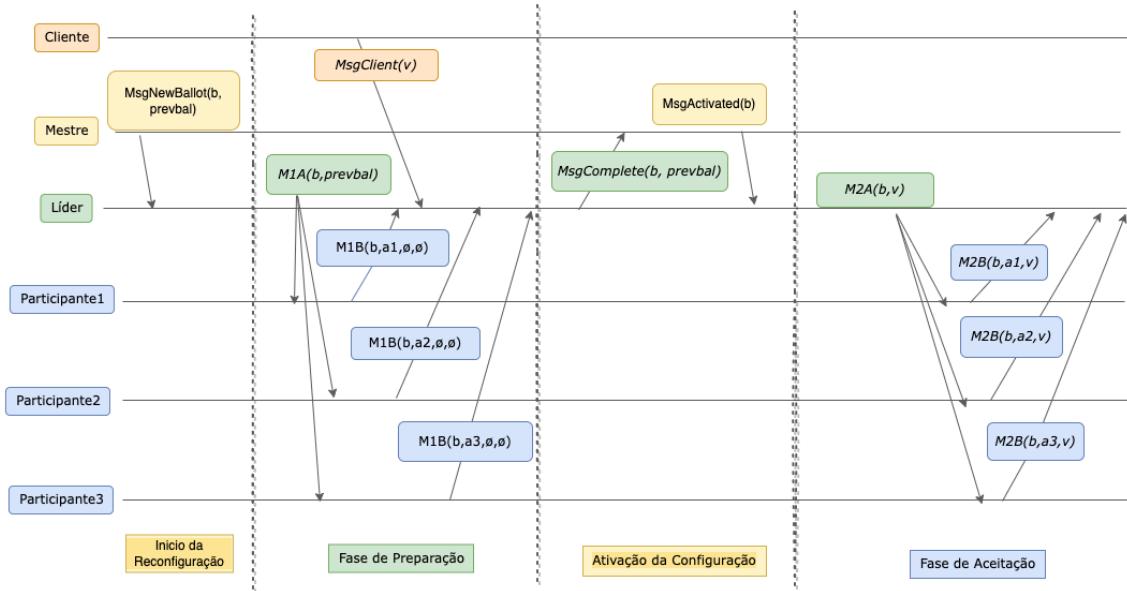


Figura 8: Sequência de fases do protocolo *Vertical Paxos II* quando não existe estado anterior a transferir.

figuração, conforme se ilustra na Figura 8. Neste caso, só depois de o líder receber a mensagem *MsgActivated* é que este inicia a Fase de Aceitação.

4

ALLOY

Nas últimas décadas, temos assistido a uma significativa evolução dos *model checkers* (Clarke et al., 2018), a qual tem tornado a análise automática de modelos de software cada vez mais eficiente. A especificação ou modelação formal de sistemas de software pode ser vista como um contrato de fiabilidade, na medida em que descreve a estrutura e o comportamento esperados do sistema. Consoante a complexidade do sistema, a sua modelação, ainda que parcial, pode ser crucial para o seu conhecimento, pois obriga o programador a pensar abstrata e matematicamente (Lamport). Acresce que, linguagens de especificação formal de alto nível, como o Alloy (Jackson, 2012) ou o TLA+ (Lamport, 2003), são complementadas com poderosas ferramentas que permitem a análise automática de sistemas complexos sem grande esforço. O nível de maturidade destas ferramentas está de tal forma consolidado que as mesmas são usadas no contexto industrial (Newcombe et al., 2015; Cunha and Macedo, 2018).

O Alloy é uma linguagem de modelação desenvolvida pelo MIT, com a coordenação de Daniel Jackson, que permite capturar as abstrações de *software* de maneira simples e sucinta. Esta linguagem, cuja predecessora é o Nitpick (Jackson, 1996), foi fortemente influenciada pela linguagem Z (da qual herdou a notação da teoria dos conjuntos e lógica de primeira ordem) e pela notação da modelação de objetos (onde retirou a classificação e caracterização dos mesmos) (Jackson, 2012).

Ao conjugar a semântica da lógica de primeira ordem, álgebra relacional e hierarquia e classificação de objetos, a linguagem Alloy mostra-se muito flexível, permitindo conceber, formal e abstratamente, o modelo, bem como introduzir-lhe restrições e operações mais complexas que descrevam como este se comporta dinamicamente (Jackson, 2002).

Acresce que, o Alloy tem integrado uma ferramenta de visualização, validação e verificação da especificação, totalmente automática, o Analyzer (cujo *model finder engine* é o Kodkod (Torlak and Jackson, 2007)), capaz de representar graficamente as suas falhas (Jackson, 2012). Este é usado para depurar e analisar as especificações, revelando-se muito útil na deteção de erros subtils e casos especiais. Esta ferramenta possui vários mecanismos de exploração e validação do modelo, nomeadamente a possibilidade de destacar e esconder graficamente as relações, percorrer as diversas instâncias e avaliar as relações num certo passo da execução, tornando a compreensão do comportamento do modelo muito mais intuitiva e amigável (Brunel et al., 2021; alloy, 2011).

O Alloy oferece um novo tipo de linguagem de especificação e análise automática graças, fundamentalmente, às seguintes características:

- lógica relacional - o *Alloy* combina os quantificadores da lógica de primeira ordem com os operadores da teoria dos conjuntos e do cálculo relacional, tendo apenas um único tipo de dados - as relações (Jackson, 2019). As funções são tratadas como relações binárias e os conjuntos e os escalares são considerados relações unárias;
- análise de pequeno escopo - o desenvolvedor determina o escopo da análise dado que, na maioria dos casos, não é possível verificar a totalidade dos estados do sistema. Esta opção tem por base a teoria da *small scope hypothesis*, a qual afirma que a maioria dos erros pode ser capturada através de um escopo pequeno (Jackson, 2019);
- transposição para **SAT** ou para **SMV** - o *Alloy* não realiza uma pesquisa explícita. Em vez disso, faz uma representação simbólica dos estados, traduzindo o modelo para **SAT** ou para **SMV**, consoante esteja a efetuar verificação automática limitada (*bounded model checking*) ou ilimitada (*unbounded model checking*) (Brunel et al., 2018).
- *model finder* - a análise automática desta linguagem tem por base um mecanismo de procura de modelos que satisfaçam determinada fórmula. Em vez de procurar construir uma prova de que determinada asserção é válida, o *Analyzer* tenta encontrar um contra-exemplo (Jackson, 2012).
- verificação exaustiva - o *Alloy* verifica automática e exaustivamente todas as possíveis configurações num universo pré-definido ou indicado pelo utilizador.
- representação gráfica - o seu *Analyzer* é uma mais-valia para perceber o modelo. As suas características de visualização e propriedade de depuração da especificação fazem deste interface uma ferramenta muito útil e amigável para o utilizador.

Além disso, esta linguagem de modelação distingue-se das demais por permitir que os comandos de análise **run**, para verificar a “satisfabilidade”⁵, e **check**, para verificar a validade⁶ de uma fórmula, possam ser incluídos na especificação, juntamente com as assinaturas, axiomas e predicados, etc..

Assim, ao agregar os processos de especificação e de verificação do modelo, o *Alloy* permite conhecer as propriedades e o comportamento do mesmo, o que se revela crucial para a sua total compreensão, quer seja antes ou depois da sua implementação.

Atualmente, o *Alloy* encontra-se na sua 6.^a versão (Brunel et al., 2021). Esta versão incorporou as extensões e melhorias previamente implementadas no *Electrum* (Macedo et al., 2016; Brunel et al., 2018). Em relação às versões anteriores, a versão 6 tem muitas funcionalidades novas (Brunel et al., 2021), designadamente:

- declarar conjuntos e relações mutáveis;
- optar entre verificação automática limitada ou ilimitada;
- especificar propriedades com lógica temporal, incorporando diversos conectivos temporais;

⁵Uma fórmula diz-se satisfazível se a mesma se verifica para alguma valoração.

⁶Uma fórmula é válida se a mesma se verifica para qualquer atribuição. Uma fórmula válida é designada de tautologia.

- escolher diferentes estratégias de *solving*;
- novas funcionalidades de exploração das instâncias.

O presente capítulo apresenta a linguagem *Alloy* através de uma abstração de alto nível do algoritmo *Paxos* e explora a sua ferramenta *Analyzer*, destacando as suas principais características.

4.1 LINGUAGEM ALLOY

Neste capítulo, explicaremos a sintaxe e a semântica da linguagem *Alloy* através da especificação de uma abstração do algoritmo *Paxos*. O objetivo é apresentar os conceitos base do *Alloy*, nomeadamente as assinaturas, os axiomas, os predicados, as funções e os comandos de análise.

A especificação do *Paxos* apresentada tem como elementos centrais os participantes que cooperam na obtenção de um acordo e os boletins que identificam cada ronda de votação. Os referidos participantes votam em boletins, no máximo um voto por ronda, sendo que um valor é escolhido quando todos os membros de um quórum votam num valor associado a um determinado boletim.

4.1.1 Lógica Relacional

O *Alloy*, conforme já mencionamos, utiliza a lógica relacional, que permite:

1. usar quantificadores:

- **no** – nenhum;
- **lone** – no máximo um;
- **one** – um;
- **some** – algum;
- **all** – todos.

2. usar operadores da lógica proposicional:

- **&&** ou **and** – Conjunção;
- **||** ou **or** – Disjunção;
- **=>** ou **implies** – Implicação;
- **!** ou **not** – Negação;
- **<=>** ou **iff** – Equivalência.

3. usar operadores relacionais:

- **+** – União – a expressão $x + y$ significa a junção dos elementos das relações x e y ;

- $\&$ – Interseção - a expressão $x \& y$ corresponde aos elementos que se repetem nas relações x e y ;
- $-$ – Diferença – a expressão $x - y$ extrai todos os elementos que pertencem à relação x , mas não pertencem a y ;
- $.$ – Composição – se o último elemento de um tuplo de x for igual ao primeiro elemento de um tuplo de y , então $x.y$ conterá um tuplo resultante da concatenação dos dois, omitindo os elementos que tornaram possível a composição;
- $[]$ – Box Join – é semanticamente equivalente à composição, mas processa os argumentos por ordem inversa, por exemplo, $x[y] \Leftrightarrow y.x$ e $x.y[z] \Leftrightarrow z.(x.y)$;
- \rightarrow – Produto Cartesiano – $x \rightarrow y$ corresponde ao conjunto de todos os tuplos que resultam da concatenação de um tuplo de x com um tuplo de y ;
- $<:$ – Restrição de Domínio – a expressão $x <: y$, onde x é um conjunto e y é uma relação, contém os tuplos de y cujos primeiros elementos pertencem a x ;
- $:>$ – Restrição do Contra-Domínio – a expressão $x :> y$, onde y é um conjunto e x é uma relação, contém os tuplos de x cujos últimos elementos pertencem a y ;
- $++$ – Override – $x ++ y$ significa que os elementos da relação y sobrescrevem os da relação x , ou seja, os tuplos de x cujo primeiro elemento seja também o primeiro elemento de um tuplo de y serão substituídos pelo segundo. Acresce que, os tuplos de y cujo primeiro elemento não corresponda ao primeiro elemento de um tuplo de x também farão parte de $x ++ y$;
- \sim – Conversa ou Transposta – inverte a ordem dos átomos de cada tuplo, ou seja, $x. \sim y \Leftrightarrow y.x$;
- \wedge – Fecho Transitivo – \wedge_x equivale à menor relação que contém x e é transitiva, i.e., se a relação contém o tuplo $a \rightarrow b$ e o tuplo $b \rightarrow c$, então também contém o tuplo $a \rightarrow c$;
- $*$ – Fecho Transitivo-Reflexivo – $*_x + \text{id}_n$, onde id_n é a relação de identidade;
- in – Contido – uma relação está contida noutra se são iguais ou se os elementos da primeira relação são um subconjunto da segunda;
- $=$ – Igualdade – uma relação é igual a outra se ambas têm exatamente os mesmos elementos;

Além disso, a última versão do *Alloy* introduziu a noção de tempo, futuro e passado, através dos seguintes operadores temporais:

- **`always`** p - doravante, p é sempre verdadeira;
- **`eventually`** p - inevitavelmente, p será verdadeira;
- **`after`** p - no próximo estado, p será verdadeira;
- **`historically`** p - até então, p foi sempre verdadeira;

- **once** p - p foi, alguma vez, verdadeira;
- **before** p - no estado imediatamente anterior, p foi verdadeira;

Convém notar que, enquanto o operador **after** aplica-se a uma fórmula p , o operador **'** aplica-se a uma expressão relacional p , denotando p' o seu valor no estado seguinte.

4.1.2 Assinaturas e Relações

Ao desenvolver um modelo em *Alloy*, começamos por declarar e caracterizar os tipos que o compõem. Estes são declarados através da palavra-chave **sig**. Uma assinatura é um conjunto que agrupa elementos de um domínio (também designados átomos) e que pode ser visto como um tipo ou classe da especificação, conforme Especificação em 4.1. Convém referir que um átomo, neste contexto, significa uma entidade primitiva indivisível, imutável e não-interpretável, i.e., sem quaisquer propriedades associadas (Jackson, 2012).

Quando é necessário declarar um subconjunto pertencente a outra assinatura, ou seja, estabelecer uma classificação hierárquica entre conjuntos, usamos as palavras **in** ou **extends** depois da assinatura do tipo, seguido do nome da assinatura progenitora, como, por exemplo, na declaração **sig** `negativeBallot extends Ballot {}`. Apesar de ambas as palavras, **in** e **extends**, caracterizarem subconjuntos de uma assinatura, somente esta última obriga a que os respetivos subconjuntos sejam disjuntos.

As assinaturas também podem ser abstratas, desde que a palavra-chave **abstract** conste antes da assinatura, o que impede que as mesmas sejam instanciadas. Por último, no que se refere às assinaturas, a atual versão do *Alloy* permite especificar uma assinatura mutável, basta que esta seja precedida da palavra **var**, caso contrário é considerada estática (Brunel et al., 2021).

Em segundo lugar, temos que abordar o conceito de *fields* do *Alloy*, o qual traduziremos por **relações** por quanto representam puras relações matemáticas entre os diferentes conjuntos. As relações são declaradas na própria assinatura e relacionam o conjunto da assinatura com outros conjuntos do sistema. Tal como as assinaturas, as relações podem ser mutáveis, desde que precedidas da palavra-chave **var**.

Acresce que, estas são qualificadas quanto à sua multiplicidade. Conforme podemos observar na relação `maxBal`, ver Especificação 4.1, a declaração de uma relação consiste no seu nome seguida por dois pontos e o conjunto de destino, opcionalmente precedido por uma declaração de multiplicidade. No caso em concreto, temos uma relação mutável `maxBal` que pode ser lida como “*Um participante guarda no máximo um boletim, o qual corresponde ao maior que já conheceu*”. No que se refere a relações estáticas temos a relação `quorum`, que estabelece um conjunto de participantes que pertence a cada `Quorum`, conforme Especificação em 4.1. Ademais, as relações declaradas numa assinatura de hierarquia superior podem ser usadas pelas assinaturas descendentes e vice-versa (Jackson, 2012).

```
sig Acceptor {
    var votes : Ballot -> lone Value,
    var maxBal : lone Ballot
}
sig Quorum {
```

```

    quorum : some Acceptor
}
sig Value {}
sig Ballot {}

```

Extrato da Especificação 4.1: Sintaxe das assinaturas e relações em *Alloy*.

4.1.3 Módulos

A linguagem *Alloy* permite a importação de módulos, de forma semelhante às linguagens de programação. Assim, é possível criar um módulo com declarações, funções, predicados, etc., e depois importá-lo noutro módulo. No nosso caso, importaremos um módulo nativo do *Alloy*, que se encontra na diretoria `util`, e que se chama `ordering`. Este módulo confere uma ordem a uma dada assinatura passada como parâmetro. Assim, parametrizamos o módulo com a assinatura `Ballot` para que estes sejam ordenados e comparáveis, de modo a garantir a correção do protocolo. Acresce que, as funções `nexts`, `prevs`, `first` do referido módulo e mencionadas na especificação, permitem percorrer os átomos de `Ballot`. Por seu turno, os predicados `gte`, `lte` permitem comparar átomos da referida assinatura.

```

open util/ordering[Ballot]

```

Extrato da Especificação 4.2: Exemplo de importação de módulos em *Alloy*.

4.1.4 Axiomas, Predicados, Funções e Asserções

Na linguagem *Alloy* todas as restrições são estruturadas em parágrafos. Os axiomas são utilizados para descrever restrições ou propriedades do modelo que têm de ser satisfeitas. Estes são declarados através da palavra-chave `fact` seguido do nome dado ao axioma. A nossa especificação apresenta três axiomas: `Quorums`, `Init` e `Next`, conforme se pode observar na Especificação 4.3. A primeira obriga a que quaisquer dois quórum se intersectem. A restrição `Init` refere-se ao estado inicial do sistema, obrigando a que não haja votos, nem um valor associado ao `maxBal` de cada participante. Além disso, o `Next` declara as possíveis transições alternativas do sistema. Note-se que este axioma, ao contrário dos anteriores, contém a palavra `always`, que indica continuidade e persistência.

```

fact Quorums {
    all disj q1, q2: Quorum | some q1.quorum & q2.quorum
}
fact Init {
    no votes
    no maxBal
}

```

```

fact Next {
    always (
        (some a : Acceptor, b : Ballot, v : Value | voteFor[a,b,v]) or
        (some a : Acceptor, b : Ballot | increaseMaxBax[a,b]) or
        nop)
}

```

Extrato da Especificação 4.3: Exemplo de axiomas em Alloy.

Os predicados definem fórmulas que podem ser verdadeiras ou falsas. Estes também podem ser parametrizados e invocados em axiomas, asserções e funções. No *Alloy* é comum usar-se predicados para especificar os diferentes eventos de um sistema. Apesar de ser indiferente a ordem pela qual a estrutura destes predicados é organizada, geralmente a mesma divide-se em três partes: a guarda ou pré-condição, efeitos ou pós-condição e condições de enquadramento (*frame conditions*). A guarda estabelece as propriedades que têm de se verificar para que o evento possa ocorrer, a pós-condição refere-se aos efeitos que se pretendem obter e, finalmente, as condições de enquadramento referem-se às relações do sistema que permanecem inalteradas.

No nosso modelo temos seis predicados, ver Especificação 4.4, sendo que uns descrevem eventos do sistema e outros não. Os predicados `nop`, `increaseMaxBal` e `voteFor` descrevem eventos no sistema, especificando como são alteradas as relações `maxBal` e `votes`, o que se consegue com o operador `'`. Por exemplo, no predíaco `increaseMaxBal`, ver Especificação 4.4, se o participante passado como parâmetro não tiver ainda um boletim associado ao seu `maxBal` ou se o que lá estiver for menor do que boletim passado como parâmetro, então, no estado imediatamente seguinte, o seu `maxBal` será atualizado, permanecendo a sua relação `votes` e o `maxBal` dos demais participantes inalterados. Os restantes predicados limitam-se a descrever fórmulas auxiliares que serão usados num contexto de restrição ou diretamente numa asserção.

Em concreto, os predicados presentes na nossa especificação (conforme Especificação 4.4) descrevem os seguintes elementos do modelo:

- `nop` - um evento onde as relações do sistema, no estado imediatamente seguinte, se mantêm inalteradas;
- `increaseMaxBal` - um evento correspondente à atualização do valor do `maxBal` do participante;
- `showsSafeAt` - um predíaco que valida se todas as condições estão reunidas para que determinado quórum possa votar num boletim e correspondente valor associado.
- `voteFor` - um evento que especifica as restrições necessárias para que um participante *a* possa votar no valor *v* associado ao boletim *b*;
- `oneVotePerBallot` - um predíaco que verifica se cada participante votou no máximo uma vez em cada boletim;
- `safeAt` - um predíaco que verifica que nenhum outro valor que não *v* foi ou poderá ser escolhido num boletim menor do que o boletim *b*;

- `chosenAt` - um predicado que verifica se algum quórum escolheu o valor v , chegando, portanto, a um consenso.

```

pred nop {
    votes' = votes
    maxBal' = maxBal
}

pred increaseMaxBal[a: Acceptor, b: Ballot] {
    no a.maxBal or gt[b, a.maxBal]
    maxBal' = maxBal ++ a->b
    votes' = votes
}

pred showsSafeAt[q: Quorum, b: Ballot, v: Value]{
    all a: q.quorum | gte[a.maxBal, b] and some a.maxBal
    b = first or some c : prevs[b] {
        some a : q.quorum | a.votes[c] = v
        all d : c.nexts & b.prevs, a : q.quorum | no a.votes[d]
    }
}
pred voteFor(a: Acceptor, b: Ballot, v: Value){
    no a.maxBal or lte[a.maxBal,b]
    no a.votes[b]
    all c: Acceptor - a | no c.votes[b] = v
    some q: Quorum | showsSafeAt[q,b,v]
    votes' = votes + a->b->v
    maxBal' = maxBal ++ a->b
}
pred oneVotePerBallot{
    all a: Acceptor, b: Ballot, v: Value | lone (b->v & a.votes)
}

pred safeAt[b: Ballot, v: Value] {
    all c: prevs[b] | some q: Quorum | all a: q.quorum {
        c->v in a.votes or
        (some a.maxBal and gt[a.maxBal,c]
        and no a.votes[c])
    }
}

pred chosenAt[b: Ballot, v: Value] {
    some q: Quorum | all a: q.quorum | b->v in a.votes
}

```

Extrato da Especificação 4.4: Exemplo de predicados em Alloy.

As funções são declaradas através da palavra-chave `fun`, seguida pelo nome da função, uma lista opcional de parâmetros, expressão relacional de retorno, com aridade arbitrária, e uma expressão entre colchetes, tal como se pode observar na Especificação 4.5. As funções servem para definir relações auxiliares da especificação e,

em particular, as funções sem parâmetros podem ser utilizadas na definição do *theme*, do *Alloy Analyzer*, para destacar visualmente alguns elementos.

A função `chosenValue`, descrita na Especificação 4.5, retorna um valor, caso haja consenso.

```
fun chosenValue : lone Value {
    {v:Value | some b: Ballot | chosenAt [b,v]}
}
```

Extrato da Especificação 4.5: Exemplo de funções em *Alloy*.

Uma asserção é uma afirmação que se julga verdadeira porque deriva dos factos definidos na especificação. As asserções são precedidas da palavra `assert` e verificadas pelo *Alloy Analyzer* através do comando `check`, como se pode ver na Especificação 4.6. No caso da mesma não se verificar significa que estamos perante uma formulação errada ou um erro na Especificação (Jackson, 2012).

No caso da nossa especificação, para garantirmos que a mesma se comporta conforme o esperado fixamos que um participante somente pode votar uma única vez em cada boletim, respeitando a propriedade de *Integridade do consenso*, ver Secção 2.2. Além disso, quisemos garantir que a nossa especificação não deixa escolher um valor arbitrário. Por conseguinte, se os participantes de um quórum votaram em determinado valor v associado a um boletim b , então, necessariamente, nenhum outro valor, além de v , foi ou pode ser escolhido em qualquer boletim menor do que b .

Por último, verificamos se apenas é possível escolher no máximo um valor, de modo a salvaguardar a propriedade de *Acordo*.

```
assert ConsistencyInVotes {
    always oneVotePerBallot
}
check ConsistencyInVotes
assert SafeAt {
    all a : Acceptor, b : Ballot, v : Value |
    always (b->v in a.votes implies safeAt [b,v])
}
check SafeAt
assert ChosenValue {
    always lone chosenValue
}
check ChosenValue
```

Extrato da Especificação 4.6: Exemplo de asserções em *Alloy*.

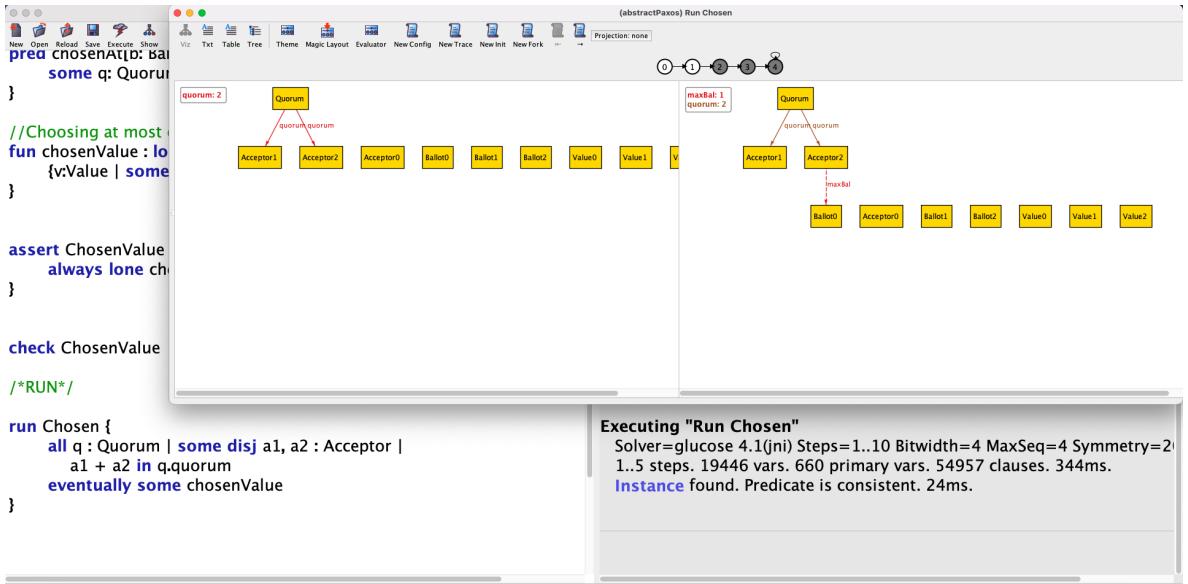


Figura 9: Instância resultante do comando *run Chosen*.

4.2 ALLOY ANALYZER

Depois de completa a especificação formal é necessário fazer a sua análise e verificação. O *Alloy Analyzer*, doravante *Analyzer*, permite ao utilizador analisar, depurar, validar e verificar o modelo, através de diferentes mecanismos, incluindo uma exploração interativa do mesmo semelhante à simulação (Brunel et al., 2019).

A validação e verificação do modelo é conseguida com a introdução dos comandos de análise `run` e `check` na especificação, exemplificados na Especificação 4.7. A estrutura do comando `run` consiste na palavra-chave `run`, opcionalmente seguida de um nome, um predicado e um escopo. O comando `check` tem a mesma estrutura, à excepção da palavra-chave e o facto de mesmo estar vinculado a uma asserção (Jackson, 2012).

O comando `run` ordena que seja avaliada a “satisfabilidade” de uma fórmula e devolve uma possível instância consistente com as restrições impostas, a qual pode depois ser visualizada, conforme podemos observar na Figura 9. Esta funcionalidade revela-se muito útil porque permite ao utilizador validar a instância e facilmente verificar se esta descreve um comportamento expectável do sistema. Convém notar que o termo instância, no caso de uma especificação com elementos mutáveis e/ou fórmulas temporais, significa um traço de execução válido, i.e., uma sequência infinita de estados, em que cada estado é uma valoração das assinaturas e relações declaradas. A sequência infinita de estados é representada através do laço do último passo para qualquer outro passo do traço, incluindo o próprio. Quando não temos elementos variáveis, temos uma representação estática do modelo em que a sequência infinita de estados consiste num laço para o primeiro e único estado (alloy, 2011).

Por outro lado, o comando `check` obriga o *Analyzer* a procurar um contra-exemplo da asserção que se quer verificar. Entretanto, caso não se encontre um contra-exemplo não há garantia de que, efetivamente, a fórmula seja válida num escopo mais alargado. De ressaltar que, quando estamos no âmbito de uma verificação limitada

The screenshot shows the Alloy Analyzer interface. On the left, there is a code editor with a tab bar containing 'New', 'Open', 'Reload', 'Save', 'Execute', and 'Show'. Below the tabs, the code editor displays a model specification in Alloy language. The code includes sections for signatures, facts, and predicates. On the right, there is a log window titled 'Alloy Analyzer [what is new] [spec] 6.1.0 built 2021-11-03T15:25:43.736Z'. The log shows four commands being executed: 'Check consistencyInVotes', 'Check SafeAt', 'Check ChosenValue', and 'Run Chosen'. Each command has associated solver statistics and a status message indicating no counterexample was found.

```

New Open Reload Save Execute Show
open util/ordering[Ballot]

/*SIGNATURES*/
sig Acceptor {
    var votes : Ballot -> lone Value,
    var maxBal : lone Ballot
}
sig Quorum {
    quorum : some Acceptor
}
sig Value {}
sig Ballot {}

/*FACTS*/
fact Quorums {
    all disj q1, q2 : Quorum | some q1.quorum & q2.quorum
}

fact Init {
    no votes
    no maxBal
}

fact Next {
    always (some a: Acceptor, b: Ballot, v:value {
        IncreaseMaxBal[a,b] or VoteFor[a, b, v] or Nop
    })
}

/*PREDICATES*/

```

Figura 10: Editor da linguagem *Alloy*.

(*bounded model checking*), o *Analyzer* procura contra-exemplos de comprimento crescente e garante o retorno do menor, caso o mesmo exista. Conforme podemos observar na Figura 10, nenhum dos comandos `check` apresentaram um contra-exemplo.

Um comando `run` com o corpo vazio faz com que o *Analyzer* procure uma qualquer instância consistente com os factos. Todavia para obter uma instância mais representativa do modelo, associamos ao comando `run Chosen`, (ver Especificação 4.7), duas restrições adicionais: 1) os quóruns devem ter pelo menos dois participantes e 2) inevitavelmente, um valor tem de ser escolhido.

```

check ChosenValue
run Chosen {
    all q : Quorum | some disj al, a2 : Acceptor |
        a1 + a2 in q.quorum
    eventually some chosenValue
}

```

Extrato da Especificação 4.7: Exemplo dos comandos *check* e *run*.

Assim, executado o referido comando, o *Analyzer* gerou uma instância composta por quatro passos, os quais estão representados na Figura 11. O estado inicial é composto por dois quóruns, `Quorum0` e `Quorum1`, dois participantes, `Acceptor1` e `Acceptor2`, e dois boletins, `Ballot0` e `Ballot1`, sendo que os dois participantes pertencem a ambos os quóruns. No estado seguinte, o `Acceptor2` incrementou o seu `maxBal` com o `Ballot0`.

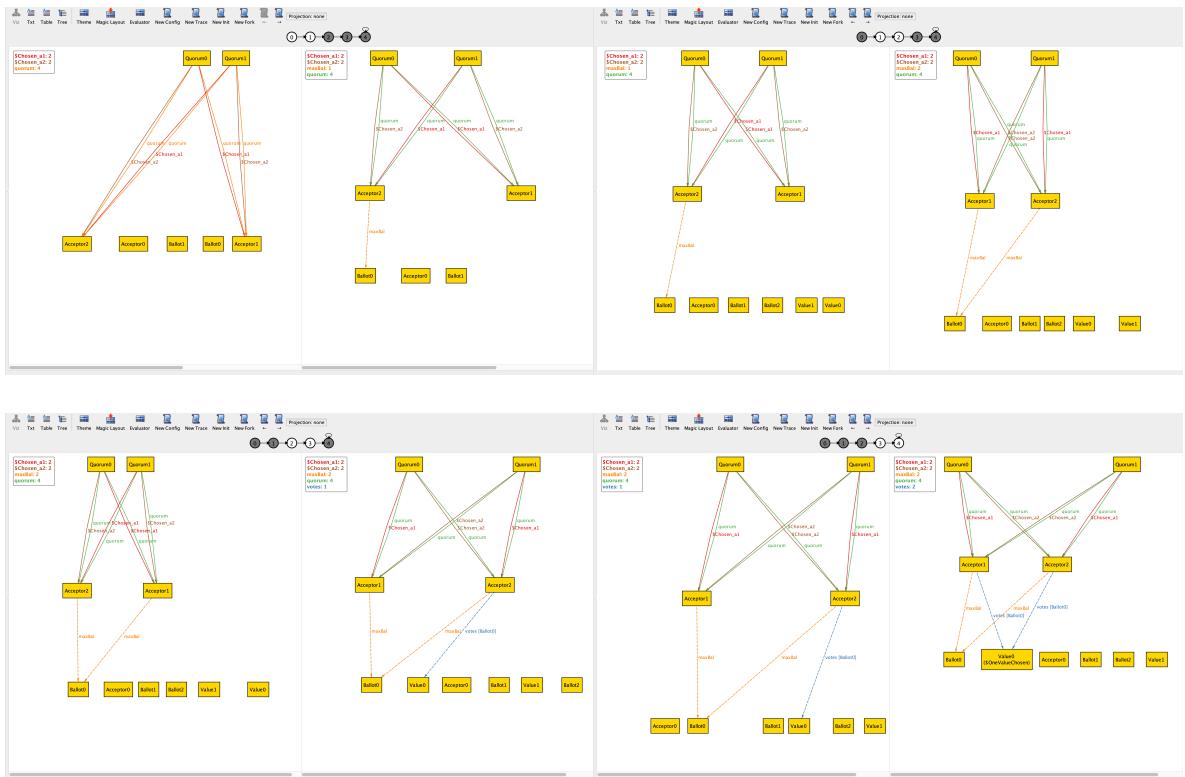


Figura 11: Traços da instância do nosso modelo.

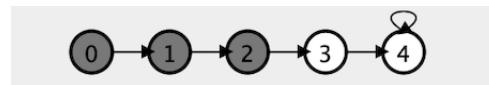


Figura 12: Representação sucinta do traço.

De seguida, o `Acceptor1` executou a mesma operação. O próximo passo mostra a votação do `Acceptor2` no `Ballot0`, cujo valor associado é o `Value0`. Finalmente, no quarto estado é escolhido um valor, pois o `Acceptor1` votou também no `Ballot0` e `Value0`.

Uma representação sucinta do traço de execução é apresentada logo abaixo da barra de ferramentas do Analyzer, ver Figura 12. No caso concreto, a instância consiste em cinco estados, com um laço (*loop*) no último estado. Embora todos os traços sejam infinitos, a ferramenta apenas devolve traços que podem ser representados finitamente, nomeadamente que podem ser descritos com um laço do último estado para um dos estados anteriores.

Em cada momento, o *Analyzer* destaca uma das suas transições entre estados, dividindo a sua janela em estado atual, representada no seu lado esquerdo, e o estado imediatamente seguinte, reproduzido no seu lado direito, conforme se pode ver na Figura 11. Na visualização de um estado podemos ver retângulos amarelos, que identificam os átomos das assinaturas a que pertencem, e setas que indicam as relações que unem os referidos átomos. De notar que os elementos mutáveis são representados a tracejado.

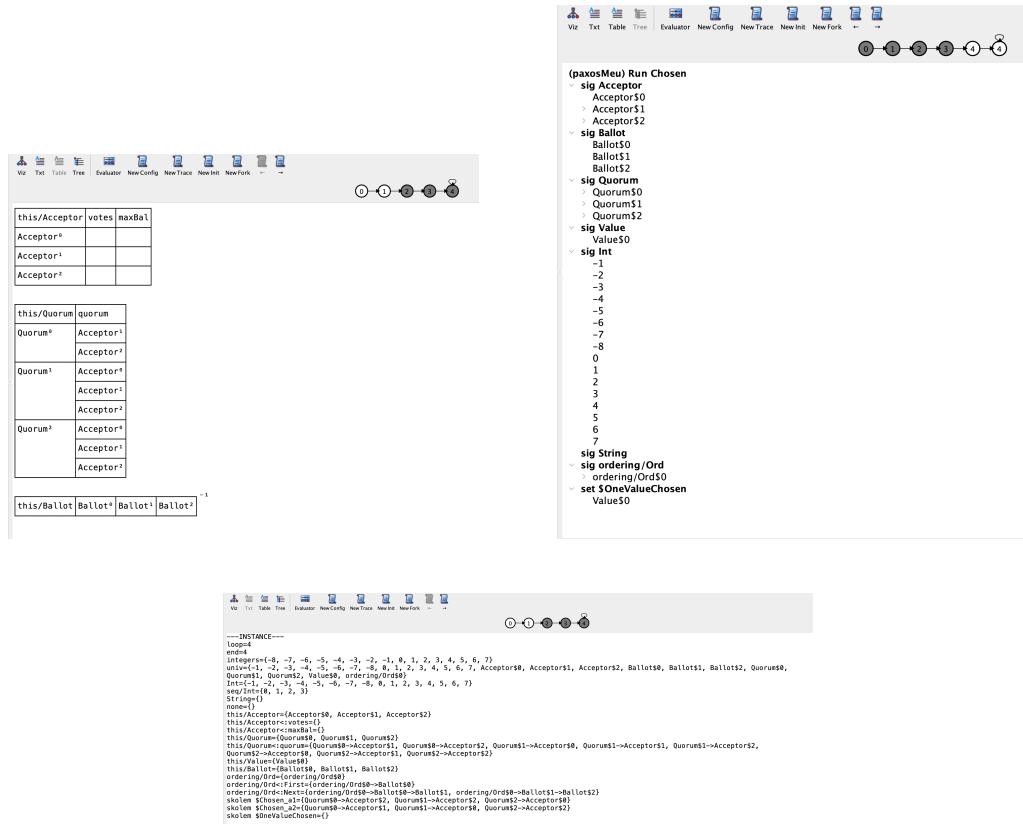


Figura 13: Formas alternativas de representação de uma instância.

4.2.1 Funcionalidades

O *Analyzer*, além de ser um *model checker*, verificando automaticamente as propriedades da especificação, também pretende ser um instrumento útil para garantir a correção da mesma. De seguida, explicaremos, sucintamente, as suas diversas funcionalidades.

Os primeiros quatro ícones da barra de ferramentas do *Analyzer*, da esquerda para a direita, referem-se ao modo de visualização das instâncias:

- *Viz* - representação gráfica já apresentada, conforme a Figura 11;
- *Txt* - representação textual, conforme parte inferior da Figura 13;
- *Table* - representação em tabelas, conforme canto superior esquerdo da Figura 13;
- *Tree* - representação em árvore, conforme canto superior direito da Figura 13.

A funcionalidade seguinte designa-se *Theme* e permite que o utilizador personalize a visualização das instâncias. Por exemplo, pode ser alterada a cor e/ou a forma dos átomos e relações, podendo-se realçar a informação relevante e esconder a considerada supérflua. Quando o botão *Theme* é pressionado, aparece do

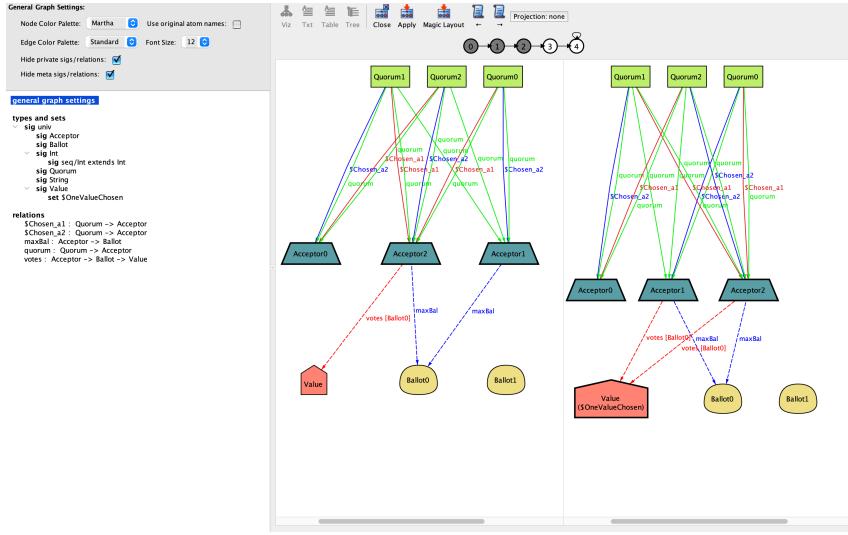


Figura 14: Personalização de um tema para o nosso modelo.

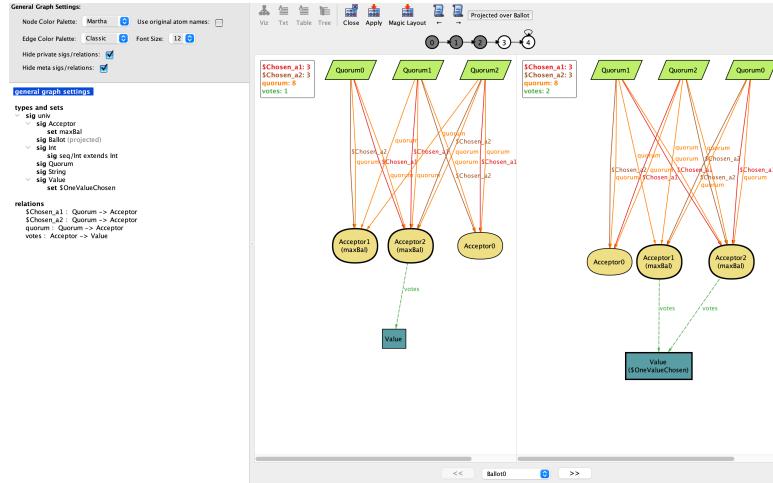


Figura 15: Aplicação da funcionalidade *Magic Layout* à especificação do *Paxos*.

lado esquerdo do visualizador várias opções. Basta selecionar a assinatura ou relação que pretendemos alterar com as opções do menu e selecionar `Apply`, seguido de `Close`, para que a instância fique com uma imagem renovada, conforme se ilustra na Figura 14. Observe-se que as configurações do tema podem ser redefinidas e também guardadas para uso posterior. Esta opção pode ser fundamental para compreender o comportamento do nosso modelo quando lidamos com sistemas complexos, visto que permite desconstruir o modelo através do destaque e da omissão de informação.

No entanto, em vez de customizar um tema manualmente, é possível obtê-lo de forma automática. Para isso basta clicar no botão `Magic Layout`, o qual escolhe as opções de visualização para os diferentes elementos da especificação (Rayside et al., 2007). Por vezes, o resultado é suficiente para ajudar a compreender o modelo, permitindo economizar tempo e trabalho. O resultado do `Magic Layout` para o nosso modelo é apresentado na Figura 15.

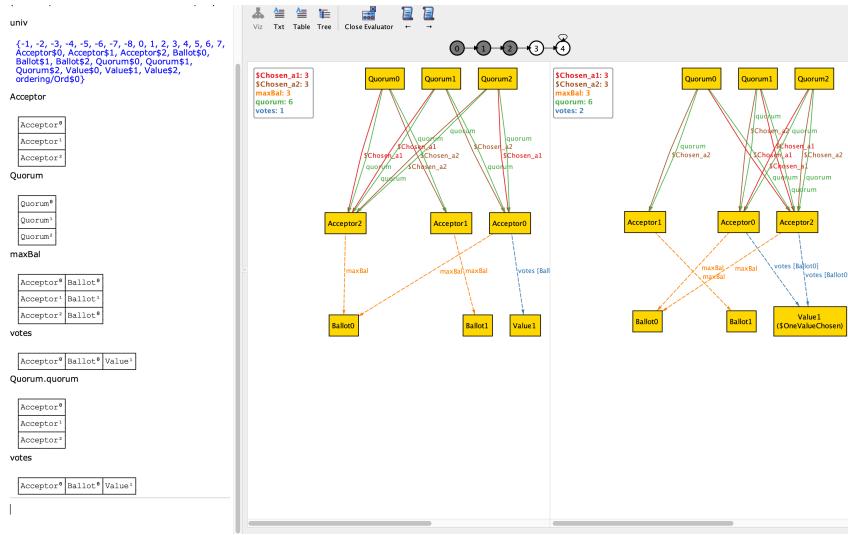


Figura 16: Exemplo de utilização do *Evaluator*.

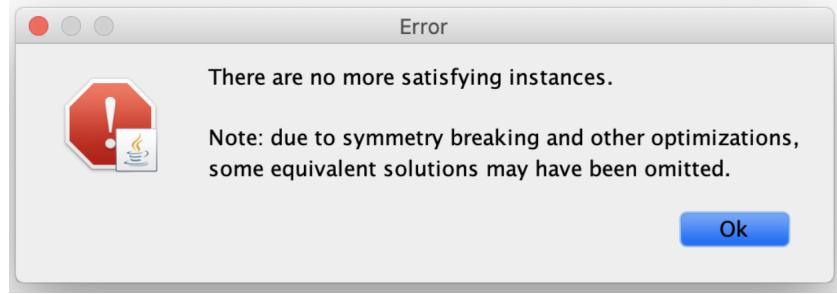


Figura 17: Aviso de inexistência de mais instâncias satisfatórias.

Ademais, o *Analyzer* oferece ainda um instrumento de depuração da especificação, o *Evaluator*. Este permite calcular o valor de qualquer expressão relacional, apresentando-o na forma de tabela, conforme se pode ver na Figura 16. Para obter o valor de uma expressão basta digitá-la no terminal do *Evaluator*, sendo a mesma avaliada no estado atual.

Na barra de ferramentas podemos ainda encontrar quatro botões que servem para alterar a instância gerada. Em primeiro lugar, temos a possibilidade de obter uma nova instância através do botão *New Config*. No caso de ser pedida uma nova configuração, os valores que vão ser alterados são os relativos aos elementos e relações imutáveis, pois são estes que constituem a configuração de um sistema. Em segundo lugar, o botão *New Trace* apresenta um novo traço de execução, mantendo a mesma configuração da instância. Outra alternativa é solicitar um valor inicial diferente para conjuntos e relações mutáveis através da funcionalidade *New Init*. Por último, temos o botão *New Fork* que nos permite obter um traço diferente a partir do passo atual. Esta opção mantém o estado atual, mas altera o(s) estado(s) subsequente(s). Caso não seja possível realizar alguma destas operações o sistema indica a inexistência de novas instâncias através da mensagem que pode ser observada na Figura 17.



Figura 18: Barra de ferramentas do *Alloy Analyzer*.

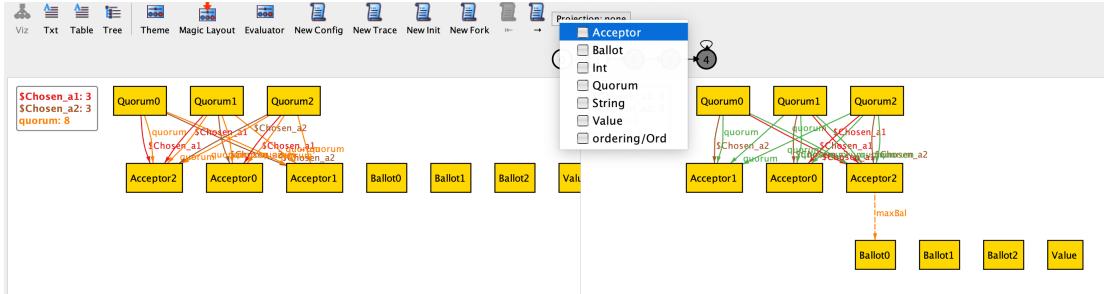


Figura 19: Funcionalidade de projeção de um conjunto.

Quando duas instâncias diferem somente nos nomes dos átomos são, na verdade, isomórficas, pelo que, mostrar ambas ao utilizador apenas dificultaria o entendimento da especificação. Para evitar isso, e também para acelerar a análise, o *Analyzer* implementa um poderoso mecanismo de quebra de simetria, *symmetry breaking*, que exclui da pesquisa as instâncias simétricas (Jackson et al., 1998). Esta técnica agrupa possíveis soluções em classes, sendo que duas soluções pertencem à mesma classe se forem permutáveis. Desta feita, o *solver* precisa de examinar somente uma única solução de cada classe, sendo que quando temos poucas classes que abarcam muitas soluções, o desempenho do *Analyzer* melhora (Jackson, 2012).

Para navegar entre os diferentes estados de um traço de execução basta pressionar os botões \leftarrow e \rightarrow , ver Figura 18, ou clicar diretamente no ícone do traço, ver Figura 12.

Por último, o utilizador tem ainda a possibilidade de projetar a representação gráfica da instância sobre determinado conjunto, ver Figura 19. Por exemplo, se o utilizador optar por projetar o conjunto *Ballot*, o programa construirá várias instâncias (uma para cada elemento de *Ballot*), permitindo que o utilizador percorra cada uma individualmente. Quando escolhida corretamente, a projeção pode tornar o resultado muito mais fácil de compreender. Mais informações sobre como esta funcionalidade podem ser encontradas em (Rayside et al., 2007).

4.2.2 Relações Auxiliares e o Analyzer

Um instrumento fundamental na exploração de instâncias é a possibilidade de personalização de um tema para ajudar a compreender e depurar o modelo. Além da construção de um tema, também é possível introduzir novas propriedades de visualização com a criação de relações auxiliares. Conforme foi já referido, as funções sem parâmetros têm a finalidade adicional de serem captadas pela funcionalidade *Theme*, podendo ser atribuída uma cor e forma distintas ao seu conjunto de retorno. Acresce que, uma vez que estas relações auxiliares são introduzidas no *Analyzer* após a verificação da especificação, não prejudicam o desempenho do programa (Brunel et al., 2021).

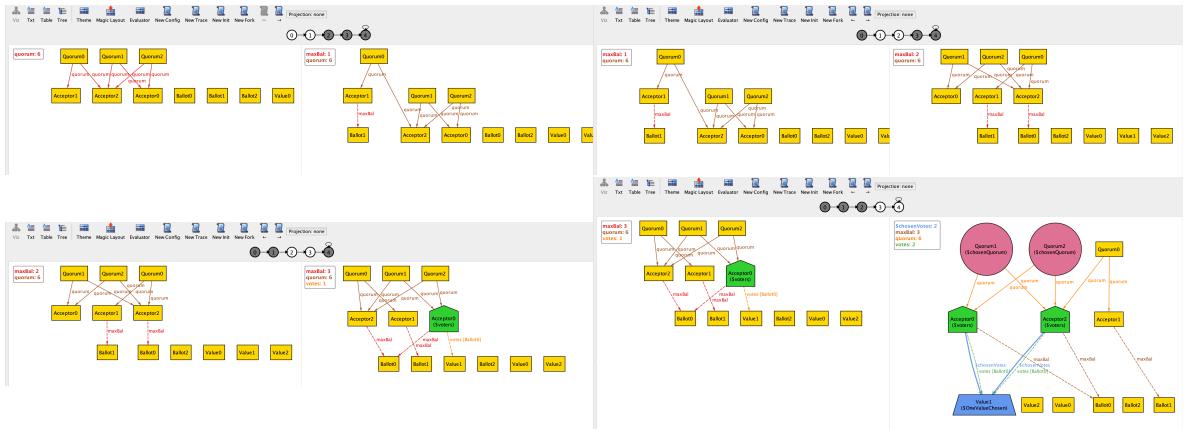


Figura 20: Destaque de relações auxiliares no Alloy Analyzer.

Assim, definimos quatro relações auxiliares que nos ajudam a destacar: os participantes que votaram em algum boletim; o valor escolhido; o quórum cujos elementos chegaram a um acordo e os participantes que votaram no valor escolhido, ver Especificação 4.8.

```

fun voters : set Acceptor{
    votes.Value.Ballot
}

fun chosenQuorum : lone Quorum{
    {q:Quorum | all a : q.quorum | some b:Ballot, v:Value | b->v in a.votes}
}

fun chosenVotes : Acceptor -> Value {
    {a : Acceptor , v:Value | some q:Quorum, b:Ballot |
        a in q.quorum and chosenAt[b,v] and q in chosenQuorum}
}

```

Extrato da Especificação 4.8: Relações auxiliares captadas pela funcionalidade *Theme*.

Com estas funções adicionais, a nossa especificação passou a ter a representação gráfica constante da Figura 20. Podemos observar que na quarta transição da execução temos as referidas relações auxiliares diferenciadas das restantes. Assim, conseguimos facilmente identificar qual o valor escolhido, quais os participantes que nele votaram e a que quórum os mesmos pertencem. Para definir as cores e as formas a atribuir basta explorar a funcionalidade *Theme* anteriormente explicada.

Convém notar que todas estas relações auxiliares têm como prefixo o símbolo $\$$. Os rótulos destas relações podem ser interpretados como arestas adicionais, como é o caso de $\$chosenVotes$, ou como sub-assinaturas de um conjunto, como acontece com $\$voters$, $\$OneValueChosen$ e $\$chosenQuorum$ que são, respetivamente, sub-assinaturas dos conjuntos $Acceptor$, $Value$ e $Quorum$, ver Figura 20.

Por fim, convém mencionar que estas funções auxiliares também podem ser mencionadas no *Evaluator*, podendo ser úteis para depurar a especificação.

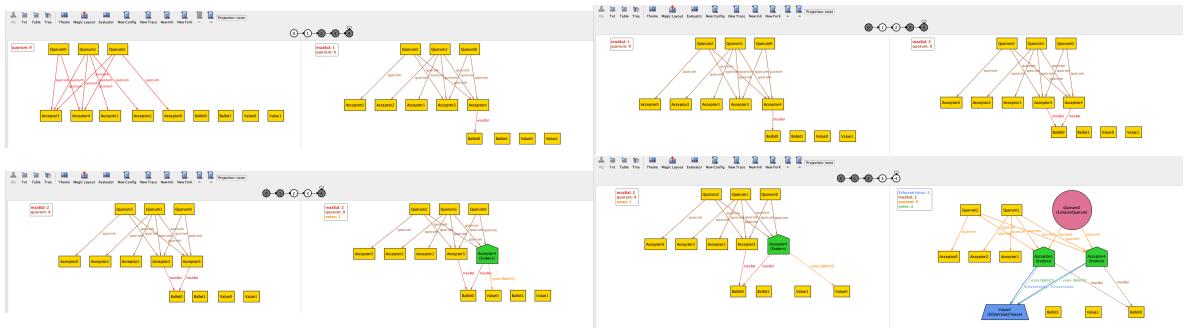


Figura 21: Transições da instância em que definimos o escopo das assinaturas.

4.2.3 Verificação Automática Limitada e Ilimitada

Atualmente, o *Analyzer* permite que o processo de verificação automática atue sobre um escopo limitado ou ilimitado. Por norma, os comandos de verificação `check` e `run` têm um escopo implicitamente predefinido (Jackson, 2012). Conforme referimos anteriormente, a linguagem *Alloy* segue o princípio da análise de pequeno escopo, pelo que sempre que não restrirmos o escopo do traço ou das assinaturas, o *Analyzer* assume, por defeito, 3 átomos por assinatura e, no máximo, 10 passos.

Todavia, estes limites superiores podem ser alterados, bastando defini-los explicitamente. Ademais, para tornar a verificação decidível, deve-se especificar o escopo de todas as assinaturas, sendo que ambos os comandos `run` e `check` admitem esta opção.

Assim, a título de exemplo, limitamos o escopo do comando `run` a 2 elementos de cada conjunto da especificação, com exceção dos conjuntos `Quorum` e `Acceptor`, que fixamos exatamente em 3 e em 5 elementos, respetivamente. Deste modo, só serão consideradas as configurações que têm **até** dois elementos de cada conjunto, com exceção do `Quorum`, que terá de ter sempre exatamente 3 elementos, e do `Acceptor`, que terá de incluir, obrigatoriamente, 5 elementos. Além disso, introduzimos a restrição de que todos os participantes têm de pertencer a, pelo menos, um `Quorum`, conforme se pode observar na Especificação 4.9. Depois de executado este comando, obtivemos uma instância com quatro transições mas cada um dos seus estados contém o número de elementos definidos, ver Figura 21. De referir, que um escopo pequeno, como o que foi apresentado, é capaz de gerar mais de 80 configurações.

```
run Chosen {
    all q : Quorum | some disj a1, a2 : Acceptor |
        a1 + a2 in q.quorum
    all a: Acceptor | some q: Quorum | a in q.quorum
    eventually some chosenValue
} for 2 but exactly 5 Acceptor, exactly 3 Quorum
```

Extrato da Especificação 4.9: Definição do escopo do comando `run`.

O limite predefinido de transições (ou passos) também pode ser alterado, conforme podemos observar na Especificação 4.10.

```
check ChosenValue for 3 but 30 steps
```

Extrato da Especificação 4.10: Definição do número de passos do comando *check*.

O comando `check ChosenValue` também não produz contra-exemplos. Conforme explicado anteriormente, estamos perante uma verificação limitada, conhecida como *bounded model checking*, que não oferece garantias de que a asserção é válida. No entanto, atendendo ao número de configurações e transições analisadas, dá-nos uma certa segurança.

Embora a análise seja sempre limitada em relação às assinaturas, podemos realizar uma verificação do tipo *unbounded model checking*, ou seja, considerando um número arbitrário de transições. Para isso é necessário escolher, no menu das opções do *Alloy*, um *solver*⁷ que suporte este tipo de verificação e usar o escopo `1 .. steps`. É possível verificar uma propriedade temporal com *unbounded model checking* desde que as assinaturas tenham o seu escopo limitado, como, por exemplo, na Especificação 4.11.

```
check ChosenValue for 3 but 1..steps
```

Extrato da Especificação 4.11: Definição de passos ilimitados.

Em suma, o *Alloy* tem dois mecanismos de análise automática: um de verificação limitada, adequado para gerar e explorar rapidamente configurações alternativas que estejam em conformidade com o modelo, e outro de verificação ilimitada, que oferece garantias ao analisar todos os estados acessíveis. Contudo, a verificação completa ou ilimitada é muito mais *pesada*, podendo falhar por falta de memória ou por excesso de tempo de execução (*runtime error*).

4.2.4 Estratégias de Decomposição

O *Alloy Analyzer* oferece ainda a possibilidade de se escolher a estratégia que o *solver* deve adotar para explorar os traços de uma instância. Existem três opções de estratégias para analisar o modelo ([alloy, 2011](#)):

- **lote** ou *batch* - analisa, num único problema, que é passado ao *solver*, todas as configurações possíveis;
- **paralela** - tira partido das dependências existentes no modelo e subdivide a especificação em pequenos problemas, *partial solutions*, um por cada configuração possível, que serão executados em paralelo pelo *solver* ([Brunel et al., 2018; Macedo et al., 2016](#));
- **híbrida** - semelhante à estratégia paralela, com a exceção de que, além das *partial solutions*, também um problema não decomposto é analisado em paralelo.

As estratégias paralela e híbrida apresentam de forma automática o número de configurações que verificam, ver Figura 22. O mesmo já não acontece com a primeira estratégia.

⁷Atualmente, os solvers *NuSMV* e *nuXmv* são suportados pelo *Alloy Analyzer*.

```

Option Decompose strategy changed to batch
Executing "Check ChosenValue"
Solver=minisat(jni) Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=batch
1..10 steps. 57518 vars. 2765 primary vars. 104639 clauses. 647ms.
No counterexample found. Assertion may be valid. 222ms.

Option Decompose strategy changed to hybrid
Executing "Check ChosenValue"
Solver=minisat(jni) Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=hybrid
11 configs. 1..10 steps. 316650 vars. 18816 primary vars. 551712 clauses. 1896ms.
No counterexample found. Assertion may be valid. 38ms.

Option Decompose strategy changed to parallel
Executing "Check ChosenValue"
Solver=minisat(jni) Steps=1..10 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=parallel
74 configs. 1..10 steps. 2503756 vars. 151965 primary vars. 4404586 clauses. 12265ms.
No counterexample found. Assertion may be valid. 66ms.

```

Figura 22: Informação sobre as estratégias de decomposição.

Analizando a Figura 22 verificamos que com a estratégia híbrida obteve 11 configurações, enquanto a estratégia paralela obteve 74 configurações. Isto acontece porque a estratégia híbrida terminou primeiro a análise do problema não decomposto e, por isso, não chega a analisar as restantes configurações.

Além disso, a estratégia de decomposição mais eficaz varia consoante o problema. Neste exemplo, a análise *batch* é muito mais rápida do que as demais. Aliás, convém referir que as estratégias híbrida e paralela, em regra, são mais lentas do que a técnica de lote quando há muitas configurações a analisar.

5

ESPECIFICAÇÃO DOS PROTOCOLOS

No que se refere às especificações dos protocolos estudados no âmbito desta dissertação, convém notar que na especificação dos protocolos de reconfiguração vertical integramos o *Paxos*. No entanto, o *Multi-Paxos* também pode ser embeddo no processo de reconfiguração, transferindo-se para a nova configuração não só um valor, mas um conjunto de valores. Por essa razão, decidimos também especificar este protocolo, dada a sua relevância e a escassez de especificações formais do mesmo. Contudo, não o integramos na especificação do protocolo de reconfiguração porque este é de tal forma complexo que as suas validações e verificações, mesmo com um escopo pequeno, ultrapassam a capacidade de tradução do *Analyzer*.

Neste capítulo iremos descrever detalhadamente a modelação do *Paxos*, muitas vezes caracterizado como um protocolo difícil de perceber, e os seus congéneres, *Multi-Paxos* e *Vertical Paxos*. As especificações, validações, verificações e temas de todos os mencionados protocolos estão disponíveis em <https://github.com/soaresCecilia/PaxosAlgorithms>.

5.1 PAXOS

Este protocolo é o ponto de partida para muitos dos protocolos de acordo distribuído existentes. Além disso, é um pilar nas técnicas de reconfiguração, as quais implicam que haja algum tipo de acordo.

5.1.1 Módulos

Na especificação do *Paxos* utilizamos o módulo nativo do *Alloy*, `ordering`, pelas razões apresentadas na Secção 4.1.3.

5.1.2 Assinaturas

Como vimos anteriormente, num algoritmo de consenso temos três classes principais de agentes: *Proposers* ou *Leaders*, *Acceptors* e *Learners*. Todavia, na especificação e implementação destes algoritmos, um processo ou participante pode atuar como um agente de várias classes. Nessa medida e por questões de simplicidade, quando desenvolvemos a especificação do *Paxos* optamos por ter apenas uma assinatura *Acceptor*, que repre-

senta todas as classes de participantes do modelo, ver Especificação 5.1. Esta assinatura tem três relações dinâmicas: `maxBal`, `maxVBal` e `maxVal`, as quais correspondem, respetivamente, ao maior boletim que o participante teve conhecimento, ao maior boletim em que o participante votou e ao valor associado a este último boletim. Em particular, a primeira relação funciona como uma barreira que impede o participante de aceitar novos boletins menores do que o seu `maxBal`.

O `Quorum` é uma assinatura composta por uma relação estática, cujo contradomínio é um conjunto de participantes, sendo que esse mesmo conjunto não pode ser vazio. Ademais, o quórum é uma abstração do conjunto de participantes cujos votos são imprescindíveis para que se alcance um consenso. Os `Quorum` são definidos no estado inicial e são imutáveis ao longo de todo o traço de execução, o mesmo acontecendo com a sua relação `acceptors`.

A terceira e quarta assinaturas referem-se ao `Value` e ao `Ballot`, ou seja, aos valores (ou comandos) que são propostos e escolhidos e ao boletim associado a cada ronda, respetivamente.

Finalmente, temos as assinaturas relativas às mensagens que são trocadas entre os participantes para que se chegue a um acordo. Assim, temos a assinatura variável e abstrata `Message`, onde definimos uma relação também variável `bal` que representa a ronda ou boletim a que a mensagem pertence. Esta assinatura representa uma classe abstrata que irá ser refinada pelos diversos tipos de mensagens que representam as várias etapas do protocolo. Ademais, como todas as mensagens têm de ter um boletim associado, declaramos esta relação na assinatura “mãe”, de modo a evitar repetições. Os tipos de mensagens possíveis, `M1A`, `M1B`, `M2A`, `M2B`, referem-se às diferentes fases do *Paxos*, anteriormente explicadas.

Convém mencionar que optamos por ter as assinaturas das mensagens variáveis, ver Especificação 5.1, bem como as suas relações, para que estas sejam criadas automaticamente pelo *Analyzer* à medida que são necessárias. Desta forma torna-se mais fácil validar interativamente as instâncias geradas. Efetivamente, a criação de mensagens de forma dinâmica permite que se simule, com o comando `New Fork`, diferentes traços de execução.

Além disso, para cada tipo de mensagem declaramos um invariante, *signature fact*, que obriga a que, a partir do momento em que a mensagem é gerada, as suas relações nunca mais se alterem, i.e., são implicitamente verdade em todos os estados futuros. Em *Alloy*, um *signature fact* tem implícito o quantificador universal `all` e o quantificador temporal `always`, pelo que, em todas as transições de estado e para todos os elementos da assinatura, este invariante projeta todas as relações (*fields*) da assinatura na variável quantificada. Assim, ter na assinatura `Message` o *signature fact* `bal = bal'` é o mesmo que escrever o axioma `always all this:`

```
Message | this.bal = this.bal'.
```

No Capítulo 7 analisamos o desempenho desta versão, designada de *dynamic messages*, bem como de outras que foram por nós desenvolvidas para otimizar os tempos de validação e verificação das diversas especificações.

```
sig Acceptor {
    var maxBal : lone Ballot,
    var maxVBal: lone Ballot,
    var maxVal: lone Value
}
```

```

sig Quorum {
    acceptors : some Acceptor
}
sig Value {}
sig Ballot {}

var abstract sig Message {
    var bal: one Ballot
} {
    bal = bal'
}

var sig M1A extends Message {}

var sig M1B extends Message {
    var acceptor: one Acceptor,
    var mbal: lone Ballot,
    var mval: lone Value
} {
    acceptor = acceptor'
    mbal = mbal'
    mval = mval'
}

var sig M2A extends Message {
    var value: one Value,
} {
    value = value'
}

var sig M2B extends Message {
    var acceptor: one Acceptor,
    var value: one Value,
} {
    acceptor = acceptor'
    value = value'
}

```

Extrato da Especificação 5.1: Assinaturas da especificação Paxos.

5.1.3 Axiomas

A especificação do *Paxos* tem três grandes axiomas, ver Especificação 5.2. O primeiro refere-se à formação dos quóruns, impondo que todos os quóruns se intersectem e que todos os participantes integrem algum dos quóruns existentes. Esta última imposição não consta do protocolo original e foi por nós declarada apenas por razões de conveniência na depuração do modelo. O segundo axioma determina que nenhuma das assinaturas ou relações variáveis podem existir no estado inicial. Finalmente, temos um axioma que determina o conjunto de ações (ou eventos) alternativos válidos sob os quais o estado do sistema pode evoluir. Em *Alloy*, a noção de estado corresponde ao valor atribuído a cada relação em cada passo do traço de execução (Brunel et al., 2021).

Já o conceito de ação está intrinsecamente relacionado com uma possível transição do estado atual para um estado seguinte. Por outras palavras, uma ação pode ser caracterizada como uma relação entre dois estados consecutivos, o atual e o seguinte, cujos efeitos no estado seguinte estão devidamente pormenorizados ([Lamport](#)).

De resto, convém salientar que o axioma `Next` utiliza a noção temporal de sempre, `always`, pelo que se verifica ao longo de todo o traço de execução, ao passo que os axiomas anteriores condicionam, somente, o estado inicial do sistema. Todavia, como os quóruns são tipos estáticos, a sua formação manter-se-á inalterada ao longo da execução.

```

fact Quorums {
    all x,y : Quorum | some x.acceptors & y.acceptors
    all a: Acceptor | some q: Quorum | a in q.acceptors
}
fact Init {
    no maxBal
    no maxVal
    no Message
    no maxVBal
}
fact Next {
    always ( nop or
        some b: Ballot | phase1A[b] or
        some a: Acceptor | phase1B[a] or phase2B[a] or
        some v: Value | phase2A[b, v] )
}

```

Extrato da Especificação 5.2: Axiomas da especificação *Paxos*.

5.1.4 Ações

O *Paxos* é um algoritmo de consenso distribuído que visa a escolha de um valor anteriormente proposto. Tal como já referimos anteriormente, para que haja garantidamente um único valor escolhido, este deve ser proposto por um líder e aceite pelos membros de um quórum.

Ademais, as nossas especificações abstraem o conceito de maioria. Por conseguinte, para que um valor seja escolhido todos os membros que integram uma assinatura `Quorum` devem votar no mesmo boletim.

O *Paxos* inicia-se com o envio de uma mensagem do tipo `M1A`, por parte do líder, cujo boletim identifica a ronda de consenso. Esta mensagem tem como finalidades: a) dar a conhecer aos demais participantes que uma nova ronda de consenso teve início e, b) caso alguma ronda ainda esteja a decorrer, interromper a mesma.

Ora, os participantes que recebem a mensagem `M1A` com um boletim maior do que qualquer outro que, até então, tenham conhecido devem responder ao líder, com uma mensagem `M1B`, indicando a última ronda e respetivo valor em que votaram. No estado seguinte, o remetente da mensagem `M1B` vê o seu `maxBal` atualizado para o valor do boletim ao qual respondeu. Como o participante apenas pode processar as mensagens `M1A`

maiores do que o seu `maxBal`, este necessita de guardar somente o maior boletim que conheceu. Esta restrição representa o compromisso em não participar, no futuro, em rondas menores àquelas que conheceu.

Note-se que apesar de o líder não estar impedido de enviar várias mensagens `M1A` iguais, o participante que as receber só irá responder uma única vez, dado que o processamento da mensagem `M1A` - especificado no predicado `phase1B` - implica a atualização do `maxBal` do participante, logo eventuais mensagens `M1A` iguais e posteriores deixarão de verificar a condição `no a.maxBal or gt[m1a.bal, a.maxBal]`, prevista na Especificação 5.3.

Convém mencionar que em *Alloy*, se utilizarmos as funções e predicados do módulo `ordering` para fazer comparações, o conjunto vazio é sempre maior do que qualquer outro. Por conseguinte, sempre que queremos comparar as relações dos boletins cuja cardinalidade seja `1one`, temos de previamente verificar se o mesmo, efetivamente, existe. Por exemplo, conforme podemos observar na Especificação 5.3, declaramos explicitamente que o participante pode enviar uma mensagem `M1B` se não tiver `maxBal` ou se o boletim da mensagem `M1A` recebida for maior do que o seu `maxBal`.

```

pred phase1B[a: Acceptor] {
    some m1a: M1A {
        no a.maxBal or gt[m1a.bal, a.maxBal]
        some m1b: M1B' {
            m1b.bal' = m1a.bal
            m1b.acceptor' = a
            m1b.mbal' = a.maxVBal
            m1b.mval' = a.maxVal
            send[m1b]
        }
        maxBal' = maxBal ++ a->(m1a.bal)
        maxVal' = maxVal
        maxVBal' = maxVBal
    }
}
pred send[m : Message] {
    Message' = Message + m
}

```

Extrato da Especificação 5.3: Ação de resposta a uma mensagem `M1A`.

Além da Fase de Preparação, anteriormente descrita, podem-se executar as ações que integram a Fase de Aceitação. Esta segunda fase tem início com o envio de uma mensagem `M2A` por parte do líder. Para que tal seja exequível é imprescindível que o líder não tenha ainda enviado uma mensagem `M2A` nessa ronda e que o mesmo conheça todas as mensagens `M1B` de pelo menos um quórum. Com efeito, só depois de obter esta informação é que o líder está em condições de propor o valor adequado. Aliás, desde que respeitadas todas as restrições impostas na especificação, a correção do protocolo não fica comprometida se o líder que coordena a Fase de Aceitação for diferente daquele que liderou a Fase de Preparação. Isto pode acontecer porque a

especificação não distingue entre líder e participante, sendo ambos um `Acceptor`, nem identifica o remetente e o destinatário das mensagens.

O predicado `phase2A` é central na especificação do algoritmo, pois é o que determina o valor adequado (i.e., `safe`) a propor. Este é encontrado tendo em consideração as seguintes restrições:

- Se nenhum elemento do quórum indicou qualquer voto anterior, então qualquer valor pode ser proposto na ronda atual - `safeNoVote`;
- Se algum ou alguns elementos do quórum tiverem votado anteriormente, então o líder proporá o valor associado ao maior boletim anteriormente votado - `safeVotedValue`.

Estas regras, cumulativamente com a imposição de que todos os quóruns se intersectam, garantem que, uma vez escolhido um valor, apenas esse valor será votado em rondas posteriores. Desta forma, assegura-se que todas os participantes aprendem o mesmo valor, mesmo que o aprendam em rondas diferentes.

Por um lado, para determinarmos se inexiste votos anteriores, verificamos se todos os elementos do quórum enviaram uma mensagem `M1B` relativa à ronda em questão e se o campo da mensagem que indica o maior voto do participante, `mval`, está vazio, ver Especificação 5.4. Note-se que a especificação garante que se o `mval` da mensagem está vazio, então o seu `mval` também está.

```
pred safeNoVote[q: Quorum, b: Ballot] {
    all a: q.acceptors {
        some m1b: M1B {
            m1b.acceptor = a
            m1b.bal = b
            no m1b.mbal
        }
        all m1b: M1B | (m1b.acceptor = a and m1b.bal = b) implies no m1b.mbal
    }
}
```

Extrato da Especificação 5.4: Predicado `safeNoVote` da especificação *Paxos*.

Por outro lado, para encontrarmos o maior boletim anteriormente votado por algum membro do quórum, o líder terá de possuir todas as respostas desse quórum e terá de comparar todos os valores inscritos no campo `mval`, conforme se descreve na Especificação 5.5. Como o módulo `ordering` considera o conjunto vazio maior do que qualquer outro tivemos de considerar expressamente a hipótese em que alguns membros do quórum votaram e outros não.

```
pred safeVotedValue[q: Quorum, b: Ballot, v: Value] {
    all a: q.acceptors | some m1b: Msg1B {
        m1b.acceptor = a
        m1b.bal = b
    }
    some n : M1B {
        n.bal = b
    }
}
```

```

n.acceptor in q.acceptors
n.mval = v
all msg1b: M1B {
    (msg1b.bal = b and msg1b.acceptor in q.acceptors) implies {
        gte[n.mbal, msg1b.mbal] or
        no msg1b.mbal
    }
}
}

```

Extrato da Especificação 5.5: Predicado *safeVotedValue* da especificação Paxos.

Para que o acordo se concretize é necessário que o valor proposto seja aceite pelos participantes através do envio de uma mensagem M_{2B} . Como é óbvio, esta mensagem só poderá ser enviada depois de recebida uma mensagem M_{2A} . Note-se que o boletim da mensagem M_{2A} rececionada terá de ser maior ou igual ao maxBal do destinatário porque este não deve, conforme já vimos, responder a rondas obsoletas. No entanto, o boletim da mensagem M_{2A} não necessita de ser maior do que o maxBal do participante. De facto, estes podem ser iguais porque se o participante tiver anteriormente respondido a uma mensagem M_{1A} com o mesmo boletim, então já atualizou o seu maxBal . Neste caso, quando o participante recebe uma mensagem M_{2A} com o boletim igual ao seu maxBal , significa que a ronda continua ativa, pelo que deve aceitar o valor proposto.

Acresce que, no estado imediatamente seguinte ao envio da mensagem M_{2B} são atualizadas todas as relações do participante que a enviou, maxBal, maxVBal e maxVal, passando as duas primeiras a conter o boletim da ronda e a última o valor votado. Poder-se-ia questionar a necessidade de atualizar o maxBal do participante nesta fase, visto que o mesmo foi já atualizado na Fase de Preparação, com o envio da mensagem M_{1B}. Contudo, pode acontecer que um participante, que não participou na Fase de Preparação, aceite o valor proposto numa mensagem M_{2A}, sem ter recebido anteriormente uma mensagem M_{1A} com o mesmo boletim. Ora, neste caso, o participante tem ainda o seu maxBal desatualizado e, por isso, deve modificá-lo.

A nossa especificação permite que as mensagens `M1A` e `M2B` sejam duplicadas, pois isso em nada viola o algoritmo e simula a retransmissão de mensagens. O nosso modelo permite ainda *stuttering steps*, ver predicho `nop`. Ademais, a escolha da próxima transição do sistema é não determinística, o que permite simular atrasos na receção e envio das mensagens e até a sua perda. Finalmente, a ordem pela qual as mensagens são recebidas é aleatória, podendo ser processadas por ordem diferente do seu envio, conforme já foi explicado.

5.2 MULTI-PAXOS

Este protocolo deriva do *Paxos* e na sua essência tem o mesmo algoritmo. A grande diferença reside no facto de não se chegar a um consenso acerca de um único valor, mas de uma sequência ordenada de valores que deve ser executada de forma determinística. Para se alcançar esse objetivo, a especificação do protocolo *Multi-Paxos* necessita de introduzir a noção de *Slot*, peça chave na construção da sequência de comandos (i.e., valores)

que deve ser ordenadamente executada pelos participantes. Cada um dos `slots` necessita de ser preenchido com um único comando e a reunião de todos eles compõem o *log* da [MER](#).

Na prática, este protocolo é mais usado do que o *Paxos* e constatamos que existem poucos modelos formais ([Chand et al., 2016](#)) que caracterizem as diferentes fases que o compõem de uma forma simples e clara.

5.2.1 Módulos

Na especificação deste protocolo utilizamos dois módulos nativos do *Alloy*, ver [Especificação 5.6](#). O primeiro foi o `ordering` que serviu para criar um conjunto totalmente ordenado de `Ballot`, pelas razões aduzidas anteriormente, sendo também necessário para o conjunto `Slot`.

Acresce que, como temos duas assinaturas distintas associadas ao mesmo módulo podemos utilizar a palavra reservada `as` seguida de uma palavra à nossa escolha que servirá para desambiguar, ao longo da especificação, a assinatura utilizada com o referido módulo. Por exemplo, no predicado `preempt`, para comparar o boletim da mensagem e o boletim que identifica o líder da ronda colocamos o vocábulo `Bal` seguido da função ou predicado do módulo em questão, `Bal/gt[m.bal, p.pBal]`. Assim, torna-se claro que o predicado `gt` irá receber como parâmetros dois `Ballot` e não `Slot`.

Além deste módulo importamos o módulo `ternary` que nos permite navegar facilmente nos quádruplos (*Acceptor*, *Ballot*, *Slot*, *Value*) que constituem as relações `aVoted` e `voted`.

```
open util/ordering[Ballot] as Bal
open util/ordering[Slot] as Slt
open util/ternary
```

Extrato da Especificação 5.6: Módulos utilizados na especificação *Multi-Paxos*.

5.2.2 Assinaturas

Em primeiro lugar, tal como no *Paxos*, existem três tipos de intervenientes: *Acceptors*, *Proposers* ou *Leaders* e *Learners*. Todavia, a modelação do *Multi-Paxos* omite este último tipo de agentes porque queremos que a mesma seja o mais clara e concisa possível. Ademais, a especificação com apenas dois intervenientes distintos já é, por si só, demasiado complexa, implicando que, muitas vezes, o *Analyzer* seja extremamente lento ou não suporte a quantidade de átomos necessários para validar algumas propriedades.

Face ao exposto, decidimos criar somente as assinaturas de `Proposer` e de `Acceptor` para acentuarmos os diferentes papéis destes dois intervenientes e para adicionarmos a ação de `preempt`, a qual ajuda a que haja progresso ([van Renesse and Altinbuken, 2015](#)).

Acresce que, por razões de generalização das demais assinaturas do protocolo, designadamente das mensagens, as duas anteriores assinaturas referidas são um subtipo da assinatura abstrata `Role`. Esta abstração permite que haja um tipo de mensagem, também geral, onde se incluem as relações variáveis `bal` e `from`,

sendo que esta última é que terá como contradomínio a assinatura `Role`, evitando-se repetir, em cada um dos diferentes tipos de mensagens, qual o emissor da mesma, ver Especificação 5.7.

Em segundo lugar, o `Acceptor` contém agora duas relações dinâmicas: `aBal`, que corresponde ao `maxBal` do *Paxos*, e `aVoted`, a qual substitui as relações variáveis `maxVBal` e `maxVal` do *Paxos* e cujo contradomínio é composto por um triplo (`Ballot`, `Slot`, `Value`). O referido triplo armazena os votos do participante, com a particularidade de que apenas guarda um voto por `slot`, o que está associado ao maior boletim. Em consequência, a relação `aVoted` não terá `slots` repetidos.

Por outro lado, o `Proposer` tem a relação variável `pBal` que indica a ronda que este lidera. Note-se que podem existir líderes concorrente e dois líderes diferentes nunca utilizam os mesmos boletins.

As assinaturas seguintes, com exceção do `Slot`, o qual é um marcador sempre associado a um valor votado, são iguais à especificação do *Paxos*, com ligeiras alterações nas relações que as compõem.

No que se refere à caracterização das mensagens, estas são criadas dinamicamente e têm as seguintes propriedades distintivas em relação ao *Paxos*:

- Cada mensagem terá, além do boletim identificador da ronda, o identificador do seu remetente, `from`;
- As mensagens `M1B` possuem um conjunto de triplos (*Ballot*, *Slot*, *Value*) que representa os votos anteriores dos participantes;
- O líder na Fase de Aceitação não se limita a propor um valor. Com efeito, na sua mensagem `M2A` deve constar um conjunto de pares composto por (*Slot*, *Value*);
- As respostas `M2B` também passam a ser compostas por conjuntos de pares (*Slot*, *Value*).

Além destas, é ainda definido uma nova classe de mensagens, as `Preempt`. Estas servem para otimizar o algoritmo quando o participante recebe mensagens do líder com boletins inferiores ao seu `aBal`. Esta é uma forma de o participante abandonar as propostas ao mesmo tempo que informa o líder do sucedido.

```
abstract sig Role {}

sig Acceptor extends Role {
    var aBal : lone Ballot,
    var aVoted: Ballot -> Slot -> Value
}

sig Proposer extends Role {
    var pBal: one Ballot
}

sig Quorum {
    acceptors : some Acceptor
}

sig Slot {}

sig Value {}

sig Ballot {}

var abstract sig Message {
    var bal: one Ballot,
    var from: one Role
}
```

```

} {
    bal = bal'
    from = from'
}
var sig M1A extends Message{ }
var sig M1B extends Message{
    var voted: Ballot -> Slot -> Value
} {
    voted = voted'
}
var sig M2A extends Message{
    var propSV: Slot-> Value
} {
    propSV = propSV'
}
var sig M2B extends Message{
    var propSV: Slot-> Value
} {
    propSV = propSV'
}
var sig Preempt extends Message{
    var to: one Role
} {
    to = to'
}

```

Extrato da Especificação 5.7: Assinaturas da especificação *Multi-Paxos*.

5.2.3 Axiomas

Os axiomas que integram a especificação do Multi-Paxos dizem respeito ao estado inicial, às possíveis ações que podem ser executadas em cada instante de tempo e aos factos relativos aos quóruns.

Em primeiro lugar, no estado inicial inexiste quaisquer relações ou conjuntos variáveis, à exceção do boletim relativo a cada `Proposer`. Para garantir a unicidade dos boletins, o axioma `SetBallotProposer` obriga a que no estado inicial a relação `pBal` tenha valores diferentes para os diferentes líderes.

Em segundo lugar, no que se refere às possíveis transições de estado, pode ocorrer qualquer uma das fases que caracterizam o *Paxos*, bem como a ação `preempt`, a qual consiste no envio de uma mensagem com o mesmo nome e com boletim maior ao então proposto.

Por último, os factos que restringem o quórum são os mesmos da especificação do *Paxos*, ver Secção 5.1.3.

5.2.4 Ações

Na sua essência, este algoritmo consiste em executar repetidamente as duas fases do *Paxos*, com algumas pequenas nuances. Por conseguinte, iremos fazer o exame desta especificação concentrando-nos apenas nas suas particularidades.

Em primeiro lugar, a mensagem $M1_A$ enviada pelo líder terá como boletim o mesmo que consta na relação $pBal$ do líder. Ademais, para permitir que haja abandono das propostas e que o mesmo seja comunicado ao líder é ainda necessário que a mensagem $M1_A$ identifique o emissor da mesma, o que é feito através da relação $from$.

Em segundo lugar, o envio da mensagem $M1_B$ está sujeito às mesmas restrições que a sua mensagem homóloga do *Paxos*. De realçar que a resposta do participante passará a conter, ao invés de um único valor, um conjunto de triplos correspondente aos seus votos precedentes.

Contudo, há agora a possibilidade do participante, em vez de ignorar as mensagens $M1_A$ com boletim menor do que o seu $aBal$, abandonar expressamente uma determinada ronda, enviando, para esse efeito, uma mensagem $Preempt$ ao líder. O envio de uma mensagem $Preempt$ permite ao líder ajustar o seu $pBal$ sempre que receba uma mensagem deste tipo cujo boletim é maior do que o seu $pBal$.

A grande diferença entre o *Paxos* e o *Multi-Paxos* está na Fase de Aceitação. Na mensagem $M2_A$, será proposto um conjunto de pares (*Slot*, *Value*) determinado a partir das mensagens $M1_B$ de um quórum. Neste algoritmo, serão propostos todos os $slots$ votados anteriormente pelos membros do quórum, com a particularidade de que o valor associado ao mesmo corresponderá ao do boletim mais elevado votado em cada *slot*. Por exemplo, se numa ronda 6 temos um quórum de participantes que enviou os seguintes votos nas suas mensagens $M1_B$, (*Ballot*, *Slot*, *Value*):

- $Acceptor1 \rightsquigarrow \{(B5, S1, V1), (B5, S2, V2)\}$
- $Acceptor2 \rightsquigarrow \{(B1, S1, V4)\}$
- $Acceptor3 \rightsquigarrow \{(B2, S1, V4)\}$

Neste caso concreto, o líder terá de propor os pares $\{ (S1, V1), (S2, V2) \}$ acrescidos de algum par ou pares (*Slot*, *Value*) ainda “não usados”, caso existam. Para encontrar os $slots$ já votados, o líder tem de computar, depois de recebidas as mensagens $M1_B$ de todo os elementos de um quórum, o maior boletim associado a cada *slot* e correspondente valor. De seguida terá de escolher um conjunto de $slots$ que não conste do conjunto anterior. Para determinar os elementos deste último conjunto definimos o seguinte algoritmo, ver Especificação 5.8:

1. Ao conjunto de $slots$ existentes no escopo do modelo retiramos os que foram anteriormente votados pelos elementos do quórum.
2. Executamos a função $mins$ que retorna os n menores $slots$ “não usados”.
3. Garantimos que cada *slot* pertencente ao conjunto dos “não usados” tem um e só um valor associado, de modo a que o mesmo *slot* não possa ser repetido na proposta.

Note-se que, com base no algoritmo acima mencionado, invariavelmente, a escolha dos *slot* “não usados” é feita por ordem crescente, sendo selecionados n *slots* “não usados” em cada iteração do protocolo. Apesar de na prática não ser este o padrão utilizado foi este o critério por nós adotado para a propositura de novos *slots*, com vista a simplificar a especificação.

```
//returns the previous slot of x
fun nth[s : set Slot, n : Int] : lone Slot {
    { x : s | #(x.*prev & s) = n }
}

//returns the n smaller elements from a set
fun mins[s : set Slot, n : Int] : set Slot {
    {x : s | some i : Int | gte[i,0] and lte[i,n] and
        x = nth[s,i]}
}

pred phase2A [p : Proposer] {
(...)

let unused = mins[Slot - Ballot.(m1B.voted).Value, 1] |{
    m1B.from = q.acceptors
    some m: M2A' {
        m.from' = p
        m.bal' = p.pBal
        maxSV in m.propSV'
        all s : Slot, v : Value | s->v in
            (m.propSV' - maxSV) implies s
            in unused
        all s : unused | one s.(m.propSV')
        send[m]
    }
}
(...)
```

Extrato da Especificação 5.8: Algoritmo de seleção dos *slots* ainda não votados.

Depois de processada a mensagem $M2A$, os participantes podem responder com os mesmos pares, indicando desta forma que votaram e armazenaram os mesmos na sua relação variável $aVoted$.

A última operação possível no nosso modelo é a possibilidade de execução do predicado $preempt$. Esta ação permite controlar a escolha dos boletins que acompanham as propostas dos líderes, garantindo que todos os líderes propõem sempre boletins disjuntos e superiores aos, até então, propostos. De outra forma, nada impediria os líderes de escolherem boletins menores do que os já processados pelos participantes, o que seria inútil e apenas contribuiria para aumentar o processamento supérfluo. Ademais, correr-se-ia ainda o risco de existirem $pBal$ repetidos, o que violaria a correção do modelo.

Desta feita, para especificar esta operação fizemos as seguintes adaptações no nosso modelo, ver Especificação 5.9:

- Em primeiro lugar, definimos o predicado `preempt`, que é também uma possível ação do sistema, que determina como e quando os líderes devem alterar o seu `pBal`. Como essa escolha não pode ser aleatória, a mesma resulta do cálculo do maior boletim proposto pelos líderes e o seu, posterior, incremento em uma unidade. Assim, garantimos que os diferentes `pBal` são únicos e nunca são revisitados.
- Posteriormente, adicionamos uma alternativa nos predicados `phase1B` e `phase2`, permitindo ao participante informar o líder que já participou numa votação com um boletim superior. Esta comunicação é feita através da mensagem `Preempt`, na qual consta o remetente e o destinatário da mensagem, bem como a maior ronda em que o participante interveio.

```

pred preempt[p: Proposer] {
    let maximum = Bal/next[Bal/max[Proposer.pBal]]
    (...)
}

pred phase1B[a: Acceptor] {
    (...)

    some m1a: M1A {
        (no a.aBal or Bal/gt[m1a.bal, a.aBal]) implies {
            (...)

            else {
                some m: Preempt' {
                    m.bal' = a.aBal
                    m.from' = a
                    m.to' = m1a.from
                    send[m]
                }
            }
        }
    }
}

```

Extrato da Especificação 5.9: Envio da mensagem *Preempt*.

5.3 VERTICAL PAXOS I

Nesta primeira variante do algoritmo de reconfiguração vertical, a nova configuração proposta pelo mestre da configuração, doravante mestre, fica imediatamente ativa. Assim, permite-se que várias configurações se encontrem, simultaneamente, a processar. Ao coordenar o processo de reconfiguração, o líder informa o mestre sempre que a transferência de estado estiver completa. Depois de processar esta informação, o mestre aprende que a configuração completou devidamente o processo de reconfiguração e considera as configurações precedentes obsoletas.

Na prática, esta versão obriga a que o líder de cada configuração tenha de contactar várias configurações anteriores, visto que as mesmas são consideradas inativas apenas quando o protocolo de reconfiguração de uma ronda posterior se completa.

5.3.1 Módulos

Esta especificação faz uso do módulo nativo do *Alloy* que estabelece uma ordem total da assinatura `Ballot` de forma a que se consiga ordenar cada uma das rondas de reconfiguração, bem como do acordo distribuído inerente a esse processo, conforme já explicado.

5.3.2 Assinaturas

Tal como na especificação mais abstrata do *Paxos*, apresentada no Capítulo 4, também aqui temos o `Acceptor`, o `Value` e o `Ballot`, cujas assinaturas têm as mesmas relações e propósito referidos nesse capítulo, conforme se pode observar na Especificação 5.10.

Outra assinatura do modelo é o `Leader`, o qual está encarregue de gerir e controlar os algoritmos de acordo embebidos na reconfiguração vertical. Esta assinatura tem uma relação estática, designada `id` que, por razões de pragmatismo, identifica a única ronda que cada líder controla. Além desta, a mesma assinatura tem ainda mais quatro relações dinâmicas que se referem a:

- `safeVal` - valor (ou comando) que, a existir, deve ser transferido para a nova configuração;
- `previousBal` - número de uma configuração ainda ativa cujo quórum de leitura deve ser contactado;
- `lCompleteBal` - número da última configuração que foi considerada completa, ou seja, é um marcador que indica que todas as rondas anteriores a esta estão obsoletas e não precisam de ser contactadas pelo líder;
- `allPreviousBal` - conjunto de todas as rondas, cujas configurações estão ainda ativas e anteriores à nova ronda proposta. Não esquecer que a última configuração completa também está ativa.

Ainda no que se refere ao líder, temos uma assinatura variável designada `RcvdNewBallot` que é um sub-conjunto da assinatura `Leader` e reúne todos os líderes que tomaram conhecimento de que a configuração que cada um lidera foi considerada ativa. Por outras palavras, este conjunto irá reunir todos os líderes que processaram a mensagem `MsgNewBallot` com o boletim igual ao seu identificador. Em algumas etapas da especificação é necessário verificar se o líder integra esse conjunto para determinar as possíveis transições. A título de exemplo, se não fosse feita esta verificação poderia ser enviada uma `MsgComplete` sem que a correspondente `MsgNewBallot` tivesse sido anteriormente processada.

Outra assinatura da nossa especificação é o `Master`, vulgo mestre. Na definição deste tipo estipulamos a sua cardinalidade como sendo `one`, de modo a forçar a que haja sempre, exatamente, um mestre, obrigando ainda que qualquer referência a esta assinatura aponte para o mesmo átomo (Brunel et al., 2021). As relações

variáveis que integram esta assinatura referem-se à última ronda que se encontra concluída, $mCompleteBal$, e ao número do boletim da nova configuração, $nextBal$. Convém ainda notar que na definição do escopo dos comandos `run` ou `check`, o número de `Ballot` tem de ser sempre superior ao número de `Leader` pois a assinatura `Master`, na sua relação $nextBal$, tem sempre o boletim relativo à ronda seguinte.

Ademais, temos a assinatura `Quorum`. A novidade está em o tipo `Quorum` ser uma assinatura abstrata, cujos átomos se desdobram nos seus subtipos: `WQuorum` e `RQuorum`, os quais representam os quóruns de escrita e de leitura, respetivamente. Na especificação do protocolo de reconfiguração, como vamos ter de lidar com várias configurações, que possuem diversos e diferentes quóruns, temos de conseguir distinguir que membros intervêm em determinada ronda. Assim, precisamos que cada `Quorum` identifique os seus membros, `acceptors`, e a ronda em que estes participam, `ballot`.

Por último, temos os diferentes tipos de mensagens que são trocadas no nosso modelo. As mensagens do tipo `M1A`, `M1B`, `M2A` e `M2B` são as mesmas que integram a especificação do *Paxos*, tendo as mesmas funções e relações anteriormente referidas na Especificação 5.1. A mensagem `MsgClient`, tal como a designação indica, corresponde às solicitações de clientes. De resto, as mensagens `MsgNewBallot` e `MsgComplete` têm como objetivo iniciar e concluir o algoritmo de reconfiguração, respetivamente.

```

sig Acceptor {
    var maxBal : lone Ballot,
    var votes: Ballot -> lone Value
}

sig Leader {
    var safeVal: lone Value,
    var previousBal: lone Ballot,
    var lCompleteBal: lone Ballot,
    var allPreviousBal: set Ballot,
    id: one Ballot
}

var sig RcvdNewBal in Leader {}

one sig Master {
    var mCompleteBal: lone Ballot,
    var nextBal: one Ballot
}

sig Value {}

sig Ballot {}

abstract sig Quorum {
    ballot: one Ballot,
    acceptors : some Acceptor
}

sig WQuorum extends Quorum {}
sig RQuorum extends Quorum {}

var abstract sig BasicMessage {}

var abstract sig ServerMessage extends BasicMessage {
    var bal: one Ballot
} {
    bal = bal'
}

```

```

}

var sig MsgClient extends BasicMessage {
    var value: one Value
} {
    value = value'
}
var sig MsgNewBallot extends ServerMessage {
    var completeBal: lone Ballot
} {
    completeBal = completeBal'
}
var sig Msg1A extends ServerMessage {
    var prevBal: lone Ballot
} {
    prevBal = prevBal'
}
var sig Msg1B extends ServerMessage {
    var acceptor: one Acceptor,
    var mbal: lone Ballot,
    var mval: lone Value
} {
    acceptor = acceptor'
    mbal = mbal'
    mval = mval'
}
var sig Msg2A extends ServerMessage {
    var value: one Value
} {
    value = value'
}
var sig Msg2B extends ServerMessage {
    var acceptor: one Acceptor,
    var value: one Value
} {
    value = value'
    acceptor = acceptor'
}
var sig MsgComplete extends ServerMessage{}
```

Extrato da Especificação 5.10: Assinaturas da especificação *Vertical Paxos I*.

5.3.3 Axiomas

Além do facto que determina o estado inicial `Init` e do que estabelece as possíveis ações que alteram o estado do sistema, `Next`, temos o invariante que estabelece as condições relativas à formação dos quóruns, ver Especificação 5.11.

Os quóruns de leitura intersetam-se com os de escrita pertencentes à mesma ronda. Por questões de simplicidade e eficiência do modelo, estipulamos que existe somente um membro em cada quórum de leitura e um único quórum de escrita em cada ronda. Estas condições obrigam a que o quórum de escrita seja composto, pelo menos, por todos os elementos dos diversos quóruns de leitura da respetiva ronda. Como o líder tem de aceder ao quórum de escrita da ronda que coordena e aos quóruns de leitura anteriores ativos, compensa tornar os quóruns de leitura mais pequenos (Lamport et al., 2009b). Note-se que estipulamos também, por razões de simplicidade, que todos os participantes pertencem a um quórum.

Além disso, para que o valor do campo `safeVal` não fique “poluído”, fixamos que cada líder lidere uma única ronda. Com esta decisão garantimos que dois líderes distintos têm identificadores diferentes e que o seu `safeVal` começa sempre vazio. Estabelecemos também que os boletins associados aos identificadores dos líderes são sempre os mais baixos de modo a evitar que o *Alloy* avalie configurações que não permitem o progresso, ver Especificação 5.11.

Resta mencionar que no estado inicial todas relações variáveis do modelo são vazias, com exceção da relação `nextBal` que guarda o número da ronda da próxima configuração e, por isso, no estado inicial, armazena o primeiro boletim.

```

fact Quorums {
    all b: Ballot {
        all wq: WQuorum, rq: RQuorum | (wq.ballot = b and rq.ballot = b)
            implies some (wq.acceptors & rq.acceptors)
        one wq: WQuorum | wq.ballot = b
        some rq: RQuorum | rq.ballot = b
    }
    all rq: RQuorum | one rq.acceptors
    all a: Acceptor | some q: Quorum | a in q.acceptors
}
fact SetBallotLeader {
    all disjoint l1, l2: Leader | no l1.id & l2.id
    all l : Leader | no b : Ballot | lt[b, l.id] and no id.b
}
fact Init {
    no maxBal
    no votes
    no safeVal
    no previousBal
    no lCompleteBal
    no mCompleteBal
    Master.nextBal = first
    no BasicMessage
}
```

```

    no allPreviousBal
    no RcvdNewBal
}

fact Next {
    always (nop or
        some m: Master | newReconfig[m] or
        completeReconfig[m] or
        some a: Acceptor | phase1B[a] or
        phase2B[a] or
        some l: Leader | leaderRcvdNewReconfig[l]
        or phase1A[l] or
        allValuesSafeOrOneValueSafe[l] or
        allValuesSafe[l] or
        stateTransferCompleted[l] or
        oneValueSafe[l] or
        some v: Value | clientRequest[v])
}

```

Extrato da Especificação 5.11: Axiomas da especificação *Vertical Paxos I*.

5.3.4 Ações

A Figura 23 apresenta, sob a forma de um diagrama de atividades, as possíveis sequências de ações que podem ocorrer. Os nomes inseridos nos retângulos representam as ações e as cores das operações indicam o interveniente que as executa: amarelo corresponde ao mestre, o verde ao líder, o azul aos participantes e o roxo ao cliente. Ademais, as setas apresentadas têm associadas as mensagens que são enviadas em cada etapa. As barras horizontais representam ações concorrentes, os losangos ações alternativas e os círculos o estado inicial e final do algoritmo de reconfiguração.

O algoritmo de reconfiguração inicia-se sempre por iniciativa do mestre, sendo que este define o líder, o número da ronda, a configuração, bem como os vários quóruns que a integram. Porém, estas ações foram por nós abstraidas da especificação porque o nosso foco é somente o algoritmo de reconfiguração.

Depois de determinados os elementos essenciais da nova configuração, impõem-se comunicá-los ao líder, através da `MsgNewBallot`. Esta mensagem contém: o boletim, representado na relação `bal`, cujo valor é igual ao `nextBal` do mestre, e a última configuração completa, informação constante do seu `completeBal` e igual ao `mCompleteBal` atual do mestre. No estado imediatamente seguinte ao envio desta mensagem, o `nextBal` do mestre tem, impreterivelmente, de ser incrementado em uma unidade.

Depois de enviada a referida mensagem e tendo a mesma sido processada pelo líder da nova configuração, este passa a integrar o conjunto `RcvdNewBallot`. Ainda nesta fase, o líder terá de atualizar as suas relações `allPreviousBal` e `lCompleteBal`, de acordo com os elementos da mensagem `MsgNewBallot`.

Sempre que se inicia o procedimento de reconfiguração os passos anteriormente descritos são incontornáveis. Todavia, chegados a este ponto, pode acontecer uma de três alternativas:

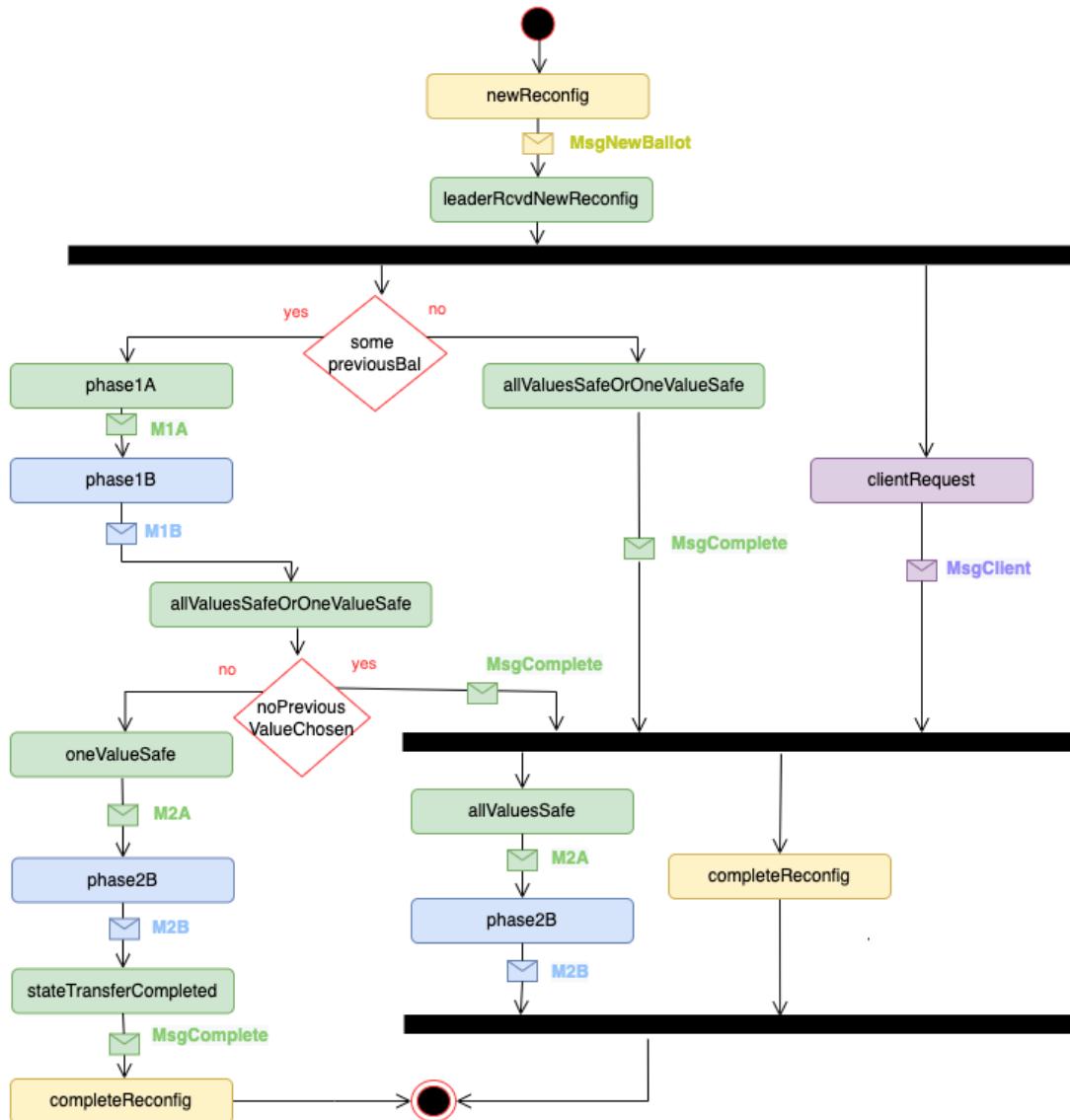


Figura 23: Diagrama de atividades da especificação *Vertical Paxos I*.

- A) existe uma ou mais configurações ativas, i.e., o campo `previousBal` da mensagem anterior está preenchido, e foi anteriormente votado um valor;
- B) não existe nenhuma configuração anterior ativa;
- C) existe configurações anteriores ativas, mas ainda não há qualquer valor escolhido.

ALTERNATIVA A): Neste caso, a premissa é que há configurações anteriores ativas e também existe pelo menos uma configuração considerada completa, sendo que só nos interessa a configuração completa com o maior boletim.

Depois de recebida a `MsgNewBallot`, o líder poderá enviar uma mensagem `M1A` aos quóruns de leitura das configurações anteriores. Estas configurações têm de ter associado um boletim compreendido no intervalo entre o boletim da configuração completa mais recente e o boletim da nova configuração. Para que tal aconteça, o líder dispõe da relação variável `allPreviousBal`, a qual armazena as configurações que devem receber e responder à mensagem `M1A`.

As mensagens `M1A` são enviadas enquanto não se verificar uma de duas situações: a) o valor que deve ser votado na ronda atual é encontrado, `safeVal`; ou b) o `previousBal` é menor do que a última ronda completa. No presente caso, vamos assumir que é encontrado um valor anteriormente votado. Para tornar esta operação clara, imaginemos o seguinte cenário (Exemplo 1): inicia-se a reconfiguração com o boletim 6 e a ronda completa mais recente tem o boletim 3. Logo, as rondas ativas são a 3, a 4, a 5 e a 6. Então, o líder da ronda 6 tem de enviar mensagens `M1A` para os membros dos quóruns de leitura das rondas 3 a 6 para saber qual o estado que deve ser transferido para a nova configuração.

Depois de recebida a mensagem `M1A` anterior, os participantes, cujo `maxBal` seja menor do que o boletim dessa mensagem `M1A`, respondem ao líder indicando o valor que votaram na ronda equivalente ao `prevBal` da mensagem. Posteriormente, cabe ao líder verificar se algum quórum de leitura das rondas ativas votou e qual o valor votado.

Se observarmos o predicado `someOneVoted`, ver Especificação 5.12, basta que haja um único voto de um quórum de leitura de uma configuração ativa, para que o líder aprenda o estado que deve ser transferido. Isto não compromete a correção da especificação porque os nossos quóruns de leitura têm um único elemento e o quórum de escrita integra todos os quóruns de leitura.

```

pred someOneVoted[l: Leader] {
  some b: l.allPreviousBal | some rq: RQuorum {
    rq.ballot = b
    some m1b: M1B {
      m1b.acceptor = rq.acceptors
      m1b.bal = l.id
      m1b.mbal = b
      some m1b.mval

      safeVal' = safeVal ++ (l->m1b.mval)
    }
  }
}

```

```

    }
( . . . )
}

```

Extrato da Especificação 5.12: Predicado *someOneVoted* da especificação *Vertical Paxos I*.

Assim, no Exemplo 1, se o líder receber uma resposta de um quórum de leitura da ronda 4, indicando um valor votado, este valor pode ser proposto na ronda 6. No caso de se transferir o estado do quórum da ronda 4, o que acontece é que tudo quanto tenha sido processado na ronda 5 será ignorado. Note-se que uma configuração ativa, mas não completa, não é o mesmo que uma configuração completa. Enquanto o estado de uma configuração completa é irrevogável (Lamport et al., 2010), o que foi processado numa configuração ativa e não completa pode ser descartado.

Depois de conhecido o valor que deve ser proposto, o líder pode enviar aos membros do seu quórum de escrita uma mensagem $M2A$ com esse valor. Este valor será aceite desde que satisfeitas as restrições anteriormente referidas.

Para que o estado se considere devidamente transferido, o líder necessita de receber a indicação de que todos os membros do quórum de escrita votaram no valor proposto. De facto, só assim há a garantia de que a transferência de estado está completa, podendo, então, enviar uma mensagem, *MsgComplete*, ao mestre informando-o desse facto.

Finalmente, tendo o mestre recebido a confirmação de que a transferência de estado foi devidamente efetuada, o mesmo considera a configuração completa. Esta operação é realizada através da atualização da sua variável *mCompleteBal*, que estabelece a linha de fronteira entre as configurações obsoletas e as ativas. De forma a garantir que as rondas obsoletas não se conseguem completar, o boletim da mensagem *MsgComplete* terá de ser superior ao valor inscrito em *mCompleteBal*, ver Especificação 5.13.

```

pred completeReconfig[m: Master] {
  some msgComplete : MsgComplete {
    no m.mCompleteBal or gt[msgComplete.bal, m.mCompleteBal]
    mCompleteBal' = mCompleteBal ++ (m->msgComplete.bal)
  }
  ( . . . )
}

```

Extrato da Especificação 5.13: Predicado *completeReconfig* da especificação *Vertical Paxos I*.

ALTERNATIVA B) Agora atentemos na situação em que não há qualquer estado a transferir. Neste caso, o líder, após ter executado a ação *leaderRcvdNewReconfig*, constata que a sua relação *previousBal* continua vazia. Neste caso, não é necessário processar a Fase de Preparação, podendo o líder enviar uma mensagem *MsgComplete*, solicitando ao mestre que considere a configuração completa. Ao receber uma mensagem *MsgComplete*, o mestre atualiza a sua primitiva *mCompleteBal*, ver Especificação 5.13.

Ademais, dado que não existe nenhum valor anteriormente votado, o líder pode propor um valor solicitado por um cliente, despoletando a Fase de Aceitação previamente descrita, ver Especificação 5.19.

Nesta alternativa, o predicado `completeReconfig` é concorrente com os predicados `allValuesSafe` e `phase2B`, ver Figura 23. Na verdade, depois de enviada a mensagem `MsgComplete` é irrelevante a ordem pela qual o predicado `completeReconfig` é processado. Isto apenas é possível porque a especificação obriga a que se realize a Fase de Preparação sempre que há configurações anteriores ativas. Assim, se for iniciada uma nova reconfiguração, os participantes terão de enviar uma mensagem `M1B` e incrementar o seu `maxBal`, o que obriga a que se transfira o estado da configuração anterior.

```
pred allValuesSafe[l: Leader] {
    no l.safeVal
    some msgComplete: MsgComplete {
        msgComplete.bal = l.id
        no m2a: M2A | m2a.bal = l.id
        some msgClient: MsgClient {
            some m2a: Msg2A' {
                m2a.bal' = l.id
                m2a.value' = msgClient.value
                send[m2a]
            }
        }
    }
    (...)
}
```

Extrato da Especificação 5.14: Predicado `allValuesSafe` da especificação *Vertical Paxos I*.

ALTERNATIVA C) A última alternativa refere-se à situação em que há configurações anteriores ativas, mas ainda nenhum membro de qualquer quórum de escrita votou.

Imaginemos a situação em que o sistema está a efetuar o seu primeiro processo de reconfiguração, ou seja, não existe nenhuma configuração anterior ativa. Neste caso, conforme vimos anteriormente (alternativa B)), depois do líder processar a mensagem `MsgNewBallot` envia uma mensagem `MsgComplete`.

Agora, imaginemos que no estado imediatamente seguinte, o mestre processa a referida mensagem `MsgComplete`, atualizando a sua relação `mComplete`, ver Especificação 5.13. De seguida, o mestre decide iniciar um novo processo de reconfiguração, sem que a Fase de Aceitação da configuração anterior tenha sido executada, ou seja, ainda ninguém votou em qualquer valor.

Neste caso concreto, a segunda ronda de reconfiguração terá de incluir, obrigatoriamente, a Fase de Preparação, em que o campo `mbal` das mensagens `M1B` enviadas pelos participantes da ronda anterior será vazio. Com efeito, para que o líder possa considerar que nenhum valor foi anteriormente escolhido, este terá de ter as respostas de um quórum de leitura de **todas** as rondas anteriores ativas, ver Especificação 5.15. E só depois de cumprida esta fase é que o líder pode enviar uma mensagem `MsgComplete`.

Assim, sempre que o líder aprende que existe alguma ronda anterior ativa, este terá de realizar a Fase de Preparação.

```
pred noPreviousValueChosen[l: Leader] {
    all b: l.allPreviousBal {
        some rq: RQuorum {
            rq.ballot = b
            some m1b: M1B {
                m1b.acceptor = rq.acceptors
                m1b.bal = l.id
                m1b.mbal = b
                no m1b.mval
            }
        }
    }
}
```

Extrato da Especificação 5.15: Predicado *noPreviousValueChosen* da especificação *Vertical Paxos I*.

OTIMIZAÇÕES Em suma, atendendo à complexidade do protocolo introduzimos algumas otimizações na especificação:

1. Quando, não existe histórico de configurações, não é processada a Fase de Preparação;
2. Eliminamos a possibilidade de envio de mensagens duplicadas;
3. Cardinalidade e formação dos quóruns, conforme já referido anteriormente.

As otimizações elencadas foram também aplicadas na especificação da segunda variante deste protocolo.

5.4 VERTICAL PAXOS II

Ao contrário da variante anterior, no *Vertical Paxos II* nunca temos, em simultâneo, mais do que uma configuração ativa. A nova configuração permanece suspensa enquanto não for expressamente declarada ativa pelo mestre, em substituição da anterior.

5.4.1 Módulos

Também aqui importamos o módulo `ordering` para poder ordenar e comparar os boletins, exatamente conforme explicado nas especificações anteriores.

5.4.2 Assinaturas

As assinaturas desta segunda versão do protocolo de reconfiguração são iguais à primeira, à exceção do conjunto `RcvdNewBal` que deixa de existir. E algumas destas assinaturas apenas diferem nas relações que as integram, pelo que somente estas serão de seguida destacadas, ver Especificação 5.16.

Em primeiro lugar, temos o `Leader` que tem a relação estática `id`, que o associa a uma determinada configuração, bem como duas relações variáveis:

- `safeVal` - armazena o valor, caso exista, que deve ser votado naquela ronda;
- `previousBal` - refere-se ao número da ronda que se encontra ativa.

Outra assinatura da nossa especificação é o `Master` que corresponde ao mestre da configuração. A cardinalidade deste conjunto é `one` pelas razões anteriormente expostas. Ademais, as duas relações que integram esta assinatura são variáveis e referem-se ao boletim da configuração ativa mais recente, `curBallot`, e à ronda imediatamente seguinte, `nextBallot`.

As restantes assinaturas referem-se ao tipo de mensagens que estão envolvidas na comunicação entre os diversos atores que integram o modelo. No que se refere às especificidades das mensagens desta variante, a mensagem que assinala o início de um algoritmo de reconfiguração, `MsgNewBallot`, tem a relação dinâmica `prevBal`, que armazena o boletim da última ronda ativa aquando do seu envio. A relação `prevBal` da mensagem `MsgComplete` tem o mesmo propósito.

Por último, esta especificação tem um novo tipo de mensagem, `MsgActivated`, enviada pelo mestre. A sua finalidade é informar o líder de que a reconfiguração foi devidamente concluída e de que a nova configuração está ativa.

```
open util/ordering[Ballot]
sig Leader {
    var safeVal: lone Value,
    var previousBal: lone Ballot,
    id: one Ballot
}
one sig Master {
    var curBallot: lone Ballot,
    var nextBallot: one Ballot
}
var sig MsgNewBallot extends ServerMessage {
    var prevBal: lone Ballot
} {
    prevBal = prevBal'
}
var sig MsgComplete extends ServerMessage{
    var prevBal: lone Ballot
} {
    prevBal = prevBal'
}
```

```
var sig MsgActivated extends ServerMessage {}
```

Extrato da Especificação 5.16: Assinaturas da especificação *Vertical Paxos II*.

5.4.3 Axiomas

Os axiomas que restringem este modelo são iguais ao da variante anterior, com as necessárias adaptações às relações variáveis existentes e às transições de estado possíveis.

5.4.4 Ações

A Figura 24 apresenta as possíveis sequências de ações, aplicando-se o mesmo método e palete de cores anteriormente explicados, ver Secção 5.3.4.

Tal como no *Vertical Paxos I*, o mestre inicia o protocolo de reconfiguração com o envio de uma mensagem `MsgNewBallot`. O boletim desta mensagem é igual ao `nextBallot` do mestre e o `prevBal` da mesma é igual ao `curBallot` do mestre. No estado seguinte ao envio desta mensagem, o mestre incrementa o seu `nextBallot`.

Depois de enviada a referida mensagem e tendo sido a mesma rececionada pelo líder da configuração, pode acontecer uma de três alternativas:

- A) existe uma configuração ativa, i.e., o campo `prevBal` da mensagem anterior está preenchido, e foi anteriormente votado um valor;
- B) não existe nenhuma configuração ativa;
- C) existe uma configuração anterior ativa, mas ainda não há qualquer valor escolhido.

ALTERNATIVA A): As restrições da primeira hipótese desta variante são mais leves comparadas com as da mesma hipótese da variante anterior. De facto, como temos somente uma única configuração ativa não temos de verificar se esta está ou não completa porque a configuração ativa está necessariamente completa.

No caso em apreço, depois de recebida a mensagem `MsgNewBallot`, o líder dará início à Fase de Preparação, ou seja, envia um mensagem `M1A`. Ademais, impõe-se que o líder atualize o seu `previousBal` para o valor da relação `prevBal` da referida mensagem, de modo a que as verificações posteriores sejam corretamente efetuadas.

As respostas ao *Prepare Request* enviado fazem-se nos moldes e condições anteriormente explicados na variante anterior e no *Paxos*.

Agora cabe ao líder verificar se algum membro de algum quórum de leitura da ronda ativa votou e qual o valor que foi votado para atualizar o seu campo `safeVal`. Assim, basta que haja um único voto de um quórum de leitura da configuração ainda ativa.

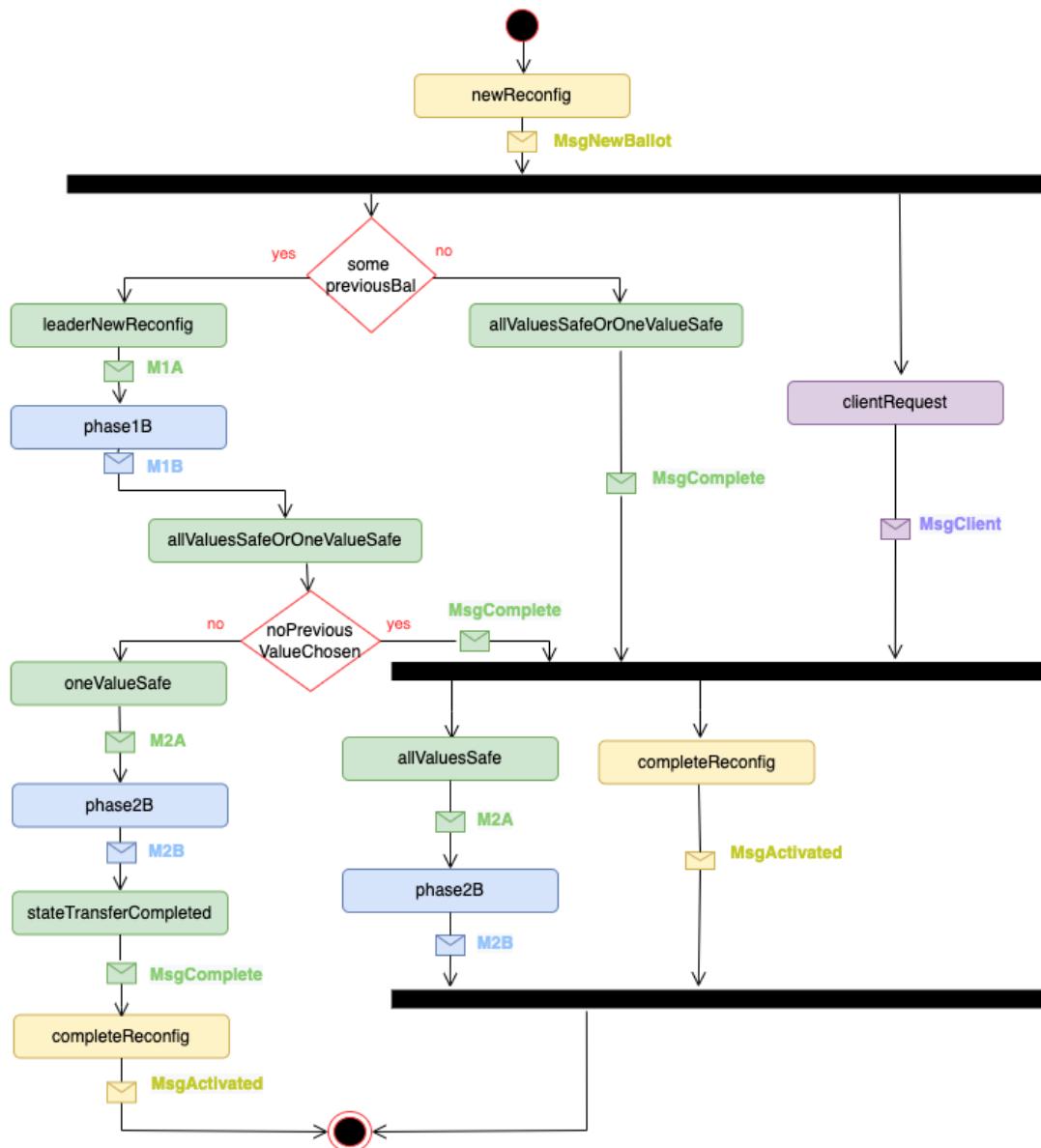


Figura 24: Diagrama de atividades da especificação *Vertical Paxos II*.

Depois de conhecido o valor que deve ser transferido para os membros do quórum de escrita da nova configuração, o líder pode enviar uma mensagem do tipo `M2A`, sendo a aceitação desta proposta feita nos termos anteriormente referidos.

Caso o líder receba a indicação de que todos os membros do quórum de escrita votaram no valor proposto envia uma mensagem `MsgComplete` ao mestre, solicitando a ativação da nova configuração. Finalmente, tendo o mestre recebido a confirmação de que a transferência de estado foi devidamente realizada, então pode ativar a nova configuração através do envio da mensagem `MsgActivated`. Todavia, para que a referida mensagem seja enviada é necessário que o valor da relação `prevBal` da mensagem `MsgComplete` seja igual ao `curBallot` do mestre, de modo a obstar que configurações obsoletas possam ser consideradas ativas quando devem ser definitivamente abandonadas. Ademais, no estado imediatamente seguinte ao envio da referida mensagem, a relação `curBallot` do mestre deve ser atualizada, ver Especificação 5.17.

```
pred completeReconfig[m: Master] {
    some msgComplete: MsgComplete {
        no m.curBallot or msgComplete.prevBal = m.curBallot
        no msgAct:MsgActivated|msgAct.bal = msgComplete.bal
        some msgAct : MsgActivated' {
            msgAct.bal' = msgComplete.bal
            send[msgAct]
        }
        curBallot' = curBallot ++ (m->msgComplete.bal)
    }
    maxBal' = maxBal
    votes' = votes
    safeVal' = safeVal
    previousBal' = previousBal
    nextBallot' = nextBallot
}
```

Extrato da Especificação 5.17: Predicado `completeReconfig` da especificação *Vertical Paxos II*.

ALTERNATIVA B: Neste caso, ainda não se conseguiu ativar nenhuma configuração, i.e., não existe `curBallot`, e, por isso, não há qualquer estado a transferir. Assim, o líder pode imediatamente solicitar que a nova configuração seja ativada, sem ter de realizar a Fase de Preparação. Para que o líder possa enviar uma mensagem `MsgComplete` têm de estar satisfeitas as restrições do predicado `allValuesSafeOrOneValueSafe`. Este predicado obriga a que o líder tenha processado um pedido de reconfiguração a si dirigido, não tenha enviado ainda uma mensagem `MsgComplete`, nem exista um valor a ser transferido. A inexistência do valor é aferida pelo predicado `noOneVoted`, o qual se considera desde logo satisfeito se não existir a relação `previousBal` do líder, conforme se pode observar na Especificação 5.18.

```
pred noOneVoted[l: Leader] {
    some q: RQuorum {
        no l.previousBal or
```

```
( . . . )
}
}
```

Extrato da Especificação 5.18: Predicado `noOneVoted` da especificação *Vertical Paxos II*.

Enviada e rececionada a mensagem `MsgComplete`, o mestre pode ativar a configuração através do envio da mensagem `MsgActivated`. Após receber a mensagem de ativação da configuração, o líder, como não tem nenhum comando anterior para processar, pode propor na mensagem `M2A` qualquer comando solicitado, anteriormente, por um cliente, conforme o predicado `allValuesSafe` da Especificação 5.19

```
pred allValuesSafe[l: Leader] {
    some mActv: MsgActivated {
        mActv.bal = l.id
        no l.safeVal
        no m2a: M2A | m2a.bal = l.id
        some msgClient: MsgClient {
            some m2a: M2A' {
                m2a.bal' = mActv.bal
                m2a.value' = msgClient.value
                send[m2a]
            }
        }
    }
    ( . . . )
}
```

Extrato da Especificação 5.19: Predicado `allValuesSafe` da especificação *Vertical Paxos II*.

Tal como na variante anterior e pelas mesmas razões referidas, também aqui os predicados `allValuesSafe`, `phase2B` e `completeReconfig` são concorrentes.

ALTERNATIVA C): A última hipótese é muito semelhante à descrita na alternativa C) da primeira variante. Neste caso, o algoritmo de reconfiguração terá de executar, invariavelmente, a Fase de Preparação e na resposta dos participantes o `mval` das mensagens do tipo `M1B` estará vazio. Por conseguinte, depois de processado o predicado `allValuesSafeOrOneValueSafe` poderá ser imediatamente enviada uma mensagem do tipo `MsgComplete` e ativada a nova configuração, sem necessidade de se realizar previamente a Fase de Aceitação.

LIMITAÇÃO Esta variante foi proposta para restringir o número de configurações ativas em simultâneo, de modo a reduzir a quantidade de quóruns de leitura a consultar no processo de reconfiguração.

No entanto, esta variante apresenta uma limitação. Imagine-se o seguinte cenário:

1. Estamos no início do processamento do sistema e é enviada pelo mestre uma mensagem `MsgNewBallot` com o boletim 0 (primeiro boletim possível);
2. No estado seguinte, o mestre envia uma outra `MsgNewBallot`, agora com o boletim 1. Isto implica que ambas as mensagens têm o seu campo `prevBal` vazio, pois como ainda não há nenhuma configuração ativa, o `curBallot` do mestre continua vazio.
3. Entretanto, a configuração com o boletim 0 é ativada. Note-se que, quando é enviada a `MsgActivated` referente ao boletim 0, a relação `curBallot` do mestre passa a ter o valor constante do boletim da `MsgComplete`, que é, no caso, 0 (ver predicado `completeReconfig` na Especificação 5.17).
4. Por conseguinte, no futuro, a configuração relativa ao boletim 1 nunca vai ser ativada porque a condição `msgComplete.prevBal = m.curBallot` do predicado `completeReconfig` nunca será satisfeita. Nesta situação, o `prevBal` da `MsgComplete` com o boletim 1 é vazio, pois quando a reconfiguração referente a este boletim foi iniciada não existia ainda nenhuma configuração anterior ativa.

O problema está em ter-se iniciado um processo de reconfiguração que visava suspender uma configuração anterior, sendo que a primeira configuração (a mais antiga) é que acabou por ser ativada, impedindo a segunda (a mais recente) de fazer progresso. Isto revela uma limitação da nossa especificação que deve ser revisitada no futuro.

Uma possível solução para esta questão passaria por se enviar uma mensagem `MsgNewBallot` quando o `prevBal` da `MsgComplete` for menor do que o `curBallot` do mestre e este `curBallot` for menor do que o boletim da `MsgComplete`.

6

VALIDAÇÃO E VERIFICAÇÃO DOS MODELOS

Ao longo do nosso trabalho foram inúmeras as vezes em que utilizamos o comando `run`, bem como as ferramentas de simulação do *Analyzer* para validar o comportamento dos vários modelos. A exploração interativa das instâncias revelou-se muito útil, permitindo-nos depurar e adquirir mais segurança nas mesmas. A validação dos modelos passou por examinar inúmeros cenários com o intuito de aperfeiçoar cada uma das especificações.

Apesar de, no desenvolvimento da nossa especificação, termos gerado e explorado inúmeras instâncias, apenas faremos referência aos cenários mais complexos para não aborrecer o leitor com situações triviais.

Depois da validação, o próximo passo foi o da análise da correção da especificação através da verificação das propriedades de *safety* dos protocolos. A principal preocupação deste trabalho foi a verificação deste tipo de propriedades em detrimento das de *liveness*. No entanto, estas podem ser objeto de trabalho futuro.

6.1 ESTRATÉGIAS GERAIS DE VALIDAÇÃO

Ao longo do nosso trabalho e à medida que fomos desenvolvendo as especificações sentimos necessidade de validar cada uma das etapas que concretizamos. Para esse efeito tivemos o auxílio de diversas técnicas e opções que o *Analyzer* oferece. Além do comando `run`, exaustivamente usado para perscrutar os traços de execução, legitimamos o comportamento dos elementos variáveis do sistema com outras técnicas de validação que fomos aplicando em conjunto e/ou separadamente.

Em primeiro lugar, o menu do *Analyzer* foi de grande serventia, principalmente no que se refere à opção de criar `New Fork` a partir de determinado passo. Com esta opção, pudemos examinar as instâncias, observando se as possíveis transições eram válidas. Ademais, a depuração da nossa especificação foi bastante facilitada com o uso do `Evaluator`, nomeadamente para verificar cada um dos `check` do nosso modelo. Para que tal fosse possível, bastou transformar as asserções a examinar em predicados e executar um comando `run`. Depois de gerada uma instância é possível através do `Evaluator` avaliar a asserção transformada em predicado.

Em segundo lugar, quando a especificação se tornou demasiado complexa e os átomos e as relações que povoavam uma instância eram de tal forma numerosos tivemos de criar um idioma para a representação de eventos (ou ações) de maneira a seguir os acontecimentos relevantes em cada passo da execução. Esta estratégia será pormenorizada na Secção 6.2.

À medida que uma especificação aumenta em tamanho e complexidade, a sua validação através da exploração aleatória de instâncias torna-se menos útil porque a obtenção de instâncias variadas é mais improvável. Chegados a esse estádio, sentimos a necessidade de complementar e conjugar as opções de exploração mais comuns com técnicas mais refinadas. Uma das soluções foi testar configurações específicas, definindo cenários de teste em diferentes módulos, conforme o exemplo descrito na Especificação 6.1. Note-se que ao atribuir a cardinalidade de `one` a cada assinatura não nos podemos esquecer de colocar todas as assinaturas do ficheiro importado como `abstract` para que as mesmas não sejam também instanciadas.

```
open Paxos
one sig A1, A2, A3 extends Acceptor{}
one sig Q1, Q2, Q3 extends Quorum {}
one sig B1, B2 extends Ballot {}
one sig V1 extends Value {}
run scenario1{
    Q1 = A1 + A2 + A3
    Q2 = A1 + A3
    Q3 = A1 + A2
    eventually some chosenValues
} for 9 Message, 10..10 steps
```

Extrato da Especificação 6.1: Exemplo de ficheiro de teste com uma configuração específica.

Outras duas estratégias relevantes para definir cenários específicos quando estamos perante contextos mutáveis são o uso de fórmulas temporais e do operador temporal sequencial ponto e vírgula para restringir as transições de estado. Por exemplo, o comando `run` na Especificação 6.2 define a exata sequência de transições de estado até se obter um acordo.

```
run example {
    some q: Quorum, disjoint a1, a2: q.acceptors, b1: Ballot, v: Value {
        phase1A[b1];
        phase1B[a1];
        phase1B[a2];
        phase2A[b1, v];
        phase2B[a1];
        phase2B[a2]
    }
} for exactly 1 Value, 1 Quorum, exactly 2 Acceptor,
exactly 2 Ballot, 6 Message, 7..7 steps expect 1
```

Extrato da Especificação 6.2: Cenário de teste com o operador ponto e vírgula.

Acresce que, o *Alloy* permite que se testem cenários pela negativa, contando com a palavra-chave `expect` para ajudar a identificar os mesmos. Por um lado, a indicação `expect 1` significa que se espera obter um cenário que obedeça às restrições do modelo, ver definição do escopo do comando `run` na Especificação 6.2.

Por outro lado, a expressão `expect 0` é usada quando confiamos que nenhum cenário com aquelas especificidades poderá ser gerado.

6.2 REPRESENTAÇÃO DE EVENTOS

O *Analyzer* não identifica o evento (i.e., ação) que corresponde a determinada transição de estado. Na verdade, não há noção de evento em *Alloy*, dado que este é modelado através de predicados, tal como as demais expressões lógicas, pelo que o *Analyzer* não tem como os distinguir. Contudo, é possível criar relações auxiliares personalizáveis que nos ajudem a compreender o comportamento do modelo e os seus eventos.

Em geral, quando a especificação é bastante complexa, o facto dos eventos não estarem especificados torna a interpretação do traço bastante difícil e propensa a erros. Assim, quisemos criar um idioma, conforme sugestão em (Brunel et al., 2021; Jackson, 2012), para retratar os eventos da especificação do protocolo *Paxos*. As especificações dos demais protocolos têm também um idioma de representação de eventos que segue o mesmo padrão. No entanto, as especificações das variantes do *Paxos* são, já de si, tão complexas que a introdução deste idioma torna-as de tal forma pesadas que o *Analyzer*, muitas vezes, não as suporta, tornando-se recorrente o erro “*Translation capacity exceeded*”.

Em primeiro lugar, começamos por introduzir uma enumeração com os possíveis eventos que podem ocorrer. Para tornar mais percutível atribuímos a cada evento o nome do correspondente predicado, ver Especificação 6.3. No menu de configuração do *Theme*, a assinatura `Event` deve ter o seu campo *Hide unconnected nodes* em *On*, de modo a que não sejam visíveis todos os possíveis eventos em todos os estados do sistema, mas somente aqueles que, efetivamente, ocorreram em cada estado.

```
enum Event {Nop, Phase1A, Phase1B, Phase2A, Phase2B}
```

Extrato da Especificação 6.3: Enumeração dos possíveis eventos da especificação *Paxos*.

De seguida, introduzimos as diversas relações auxiliares que caracterizam cada evento, ver Especificação 6.4. Em primeiro lugar, começamos pela relação `Nop`, a qual se verifica quando o estado se mantém inalterado. Esta relação auxiliar produz, em cada estado, um subconjunto de `Event`, designado `Nop`, o qual terá um elemento se o predicado `nop` for verdadeiro e será vazio, caso contrário. Note-se que as relações auxiliares que se referem a elementos mutáveis são avaliadas no estado atual aquando da sua execução, ou seja, no estado em que são chamadas.

```
fun nop_happens : set Event {
    { e: Nop | nop }
}
```

Extrato da Especificação 6.4: Função *nop_happens*.

Ademais, precisamos de alterar algumas configurações pré-existentes no menu do *Theme*. Em primeiro lugar, é necessário que os campos *Hide unconnected nodes* e *Show as labels* do evento `$nop_happens` sejam colocados em *Off*. Esta operação permitirá visualizar os eventos `Nop`.

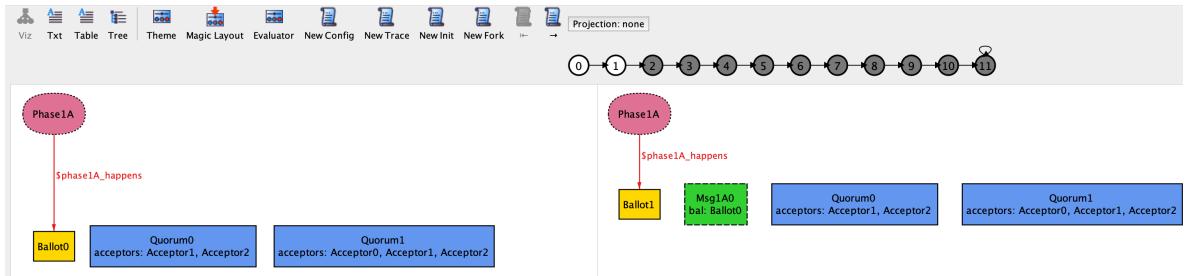


Figura 25: Representação de um evento no *Analyzer*.

Além deste, temos o evento `Phase1A` que recebe um `Ballot` como parâmetro. Logo, a relação auxiliar `phase1A_happens` deve ser capaz de identificar o boletim envolvido no predicado `phase1A`, retornando um par cujo primeiro elemento é o evento `Phase1A` e o segundo é o boletim que foi utilizado nesse mesmo evento, ver Especificação 6.5.

```
fun phase1A_happens : Event -> Ballot {
    { e: Phase1A, b: Ballot | phase1A[b] }
}
```

Extrato da Especificação 6.5: Função `phase1A_happens`.

Com estes dois eventos caracterizados e algumas modificações relativamente à forma e cores das relações, conseguimos obter a representação dos eventos conforme apresentado na Figura 25.

A representação dos dois eventos que faltam, `Phase2A` e `Phase2B`, segue a mesma estrutura que a anterior, conforme se pode observar na Especificação 6.6.

```
fun phase2A_happens : Event -> Ballot -> Value {
    { e: Phase2A, b: Ballot, v: Value | phase2A[b, v] }
}
fun phase2B_happens : Event -> Acceptor {
    { e: Phase2B, a: Acceptor | phase2B[a] }
}
```

Extrato da Especificação 6.6: Funções `phase2A_happens` e `phase2B_happens`.

Ademais, aprimoramos o nosso *Theme* introduzindo os parâmetros do evento como atributos do mesmo. Para isso, criamos outra relação auxiliar que apenas devolve o conjunto de todos os eventos que ocorrem em cada transição, ver Especificação 6.7.

```
fun events : set Event {
    nop_happens + phase1A_happens.Ballot +
    phase1B_happens.Acceptor + phase2B_happens.Acceptor +
    phase2A_happens.Value.Ballot
}
```

Extrato da Especificação 6.7: Função `events`.

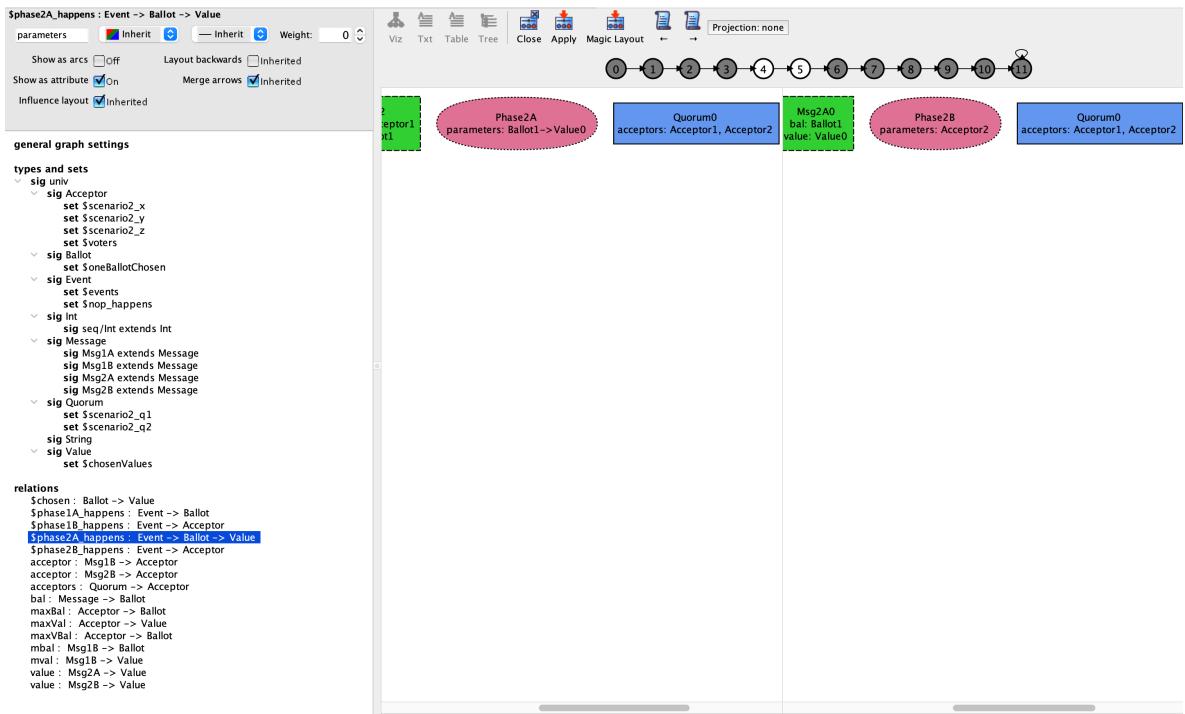


Figura 26: Representação dos parâmetros de cada evento como atributos.

Para que consigamos ter os parâmetros de cada evento como atributo do mesmo temos ainda que selecionar como *Off* os campos *Uncheck Hide unconnected* e *Show as labels* do conjunto `events` então criado, ver Especificação 6.7. Ademais, nas configurações gerais de visualização do gráfico, na parte referente a `relations`, todas as relações auxiliares relativas aos eventos devem ter o campo *Show as attribute* definido como *On*. Finalmente, renomeamos as relações auxiliares, inserindo nas respectivas caixas de texto a palavra `parameters`, ver Figura 26.

Posteriormente, quisemos garantir que ocorre sempre algum evento em cada transição, (`fact trace`). Além disso, verificamos se em cada transição de estado ocorre sempre um único evento, (`check singleEvent`), ver a Especificação 6.8.

```

fact trace {
    always some events
}
check singleEvent {
    always one events
}

```

Extrato da Especificação 6.8: Invariante `trace` e verificação da existência de um único evento em cada passo.

No entanto, esta verificação gerou um contra-exemplo, conforme se pode observar na Figura 27. Como na nossa especificação as mensagens `M1A` e `M2B` podem ser duplicadas e o envio de uma mensagem duplicada ou a ocorrência de um evento `Nop` tem exatamente o mesmo efeito, manter o estado inalterado, para o *Analyzer* a

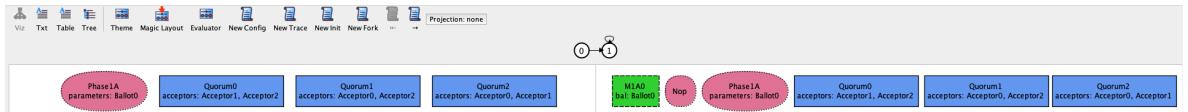


Figura 27: Contra-exemplo resultante da execução do *check singleEvent*.

ocorrência destes eventos é indistinguível. Por isso, os eventos `Nop` e `Phase1A` podem ocorrer simultaneamente porque são indiferenciáveis, tal como os eventos `Nop` e `phase2B`.

Assim, para desambiguar os efeitos dos predicados `phase1A` e `phase2B`, tivemos de proibir o envio de mensagens `M1A` e `M2B` duplicadas, inserindo-se as duas guardas constantes da Especificação 6.9 nos referidos predicados.

```
pred phase1A[b: Ballot] {
    no m1a: Msg1A | m1a.bal = b
    (...)
}

pred phase2B[a:Acceptor] {
    (...)
    no m2b: Msg2B | m2b.bal = m2a.bal and m2b.acceptor = a
    some m2b: Msg2B' {
        (...)
    }
}
```

Extrato da Especificação 6.9: Guarda introduzida nos predicados `phase1A` e `phase2B`.

6.3 VALIDAÇÃO DOS MODELOS

A título de exemplo, pois muitos outros cenários foram testados, apresentamos algumas das validações mais relevantes das nossas especificações.

6.3.1 Paxos

Em primeiro lugar, testamos se o nosso modelo consegue escolher um único valor. Assim, no `scenario1` instruímos o *Analyzer* a encontrar uma instância que satisfaça a fórmula “*inevitavelmente, algum valor é escolhido*”. Como o *Analyzer* nos dá a menor instância possível, para avaliarmos a possibilidade de acordo quanto a um único valor, temos de utilizar o quantificador `some` em detrimento do `1one`, caso contrário a instância gerada não contemplaria a escolha de um valor. O escopo definido para esta instância teve na sua base o princípio *small scope hypothesis* (Jackson, 2019).

Os cenários que se seguem dizem respeito a instâncias concretas que quisemos validar, algumas por omissão, descrevendo expressamente o traço de execução pretendido. O escopo definido para cada `run` teve em

consideração o número mínimo de elementos de cada assinatura *top-level* e de passos necessários para obter uma instância que satisfaça as restrições impostas.

A segunda situação, `scenario2`, descreve uma sequência de passos em que, de acordo com a especificação, é possível chegar a um acordo. Neste caso, quisemos averiguar se estando todas as condições reunidas para a obtenção de um acordo, inevitavelmente, um único valor era escolhido.

O `scenario3` pretende encontrar uma instância em que numa ronda anterior tenha sido escolhido um valor e na ronda seguinte seja escolhido o mesmo valor por outro quórum distinto.

De seguida avaliamos o `scenario4` que representa a possibilidade de um participante, *a1*, pertencente ao quórum *q1*, votar num determinado valor, *v1*, na ronda *b1*. Todavia, é iniciada uma nova ronda, *b2*, e um quórum *q2*, do qual o participante *a1* não faz parte, escolhe um valor diferente de *v1*, por exemplo, *v2*. Este cenário é plausível, dado que o participante que havia votado anteriormente não pertence ao quórum *q2*.

O exemplo previsto no `scenario5` é um dos casos em que é expectável que não exista nenhuma instância que satisfaça as condições impostas. Nesta hipótese, o líder inicia duas rondas sucessivas e os participantes recebem primeiro a mensagem `M1A` com o boletim maior e atualizam os seus `maxBal`. Posteriormente, recebem a mensagem `M1A` com o boletim menor do que o seu `maxBal`, pelo que devem ignorar esta última mensagem. Todavia, este cenário presume a resposta e votação num valor associado a esse boletim inferior ao `maxBal`, o que, para nosso gáudio, revelou-se impossível, conforme era desejado. Com este cenário quisemos averiguar se seria possível chegar a um consenso numa ronda obsoleta, contrariando o comportamento esperado.

O cenário `scenario6` destinou-se a demonstrar que um participante **pode** responder a uma mensagem `M2A`, sem ter recebido, anteriormente, a correspondente mensagem `M1A`. De resto, isto é sempre possível desde que o participante não tenha respondido ainda a essa mensagem e o seu `maxBal` seja menor ou igual ao boletim inscrito na referida mensagem.

No `scenario7` quisemos testar se seria possível escolher um valor diferente daquele que foi votado, na maior ronda anterior, por um participante do quórum. Como é óbvio, esta situação, a ser possível, punha em causa o âmago do *Paxos*. Tal como era expectável, esta fórmula mostrou-se insatisfazível.

Outro teste negativo que realizamos, `scenario8`, foi o envio de duas mensagens `M2A` repetidas, ou seja, com o mesmo boletim. Este predicado também se revelou inconsistente conforme era pretendido.

Por último, no `scenario9` quisemos analisar se tendo havido um voto anterior por um dos membros do quórum, o valor então votado será o valor proposto e escolhido nas rondas subsequentes. Este cenário também gerou uma instância que se comporta de acordo com o modelo.

6.3.2 Multi-Paxos

Os cenários avaliados neste modelo foram idênticos aos anteriormente testados na especificação do *Paxos*, com as necessárias adaptações. Infelizmente, esta especificação tem muitas assinaturas e a estrutura das mesmas é complexa, o que impede que o seu escopo varie muito. Sempre que tentamos aumentar o escopo era gerado o seguinte erro: “*Translation capacity exceeded*”. O motor de busca de modelos do *Alloy*, o *Kodkod*, para representar uma relação *r* de aridade *k* aloca uma matriz de tamanho n^k , onde *n* é o número de átomos

do universo. Por motivos de desempenho é usado um único *array* sequencial indexado por um inteiro da linguagem *Java* e, portanto, o tamanho da matriz é limitado a $2^{31} - 1$, valor em *Java* definido para a constante *Integer.MAX_VALUE* (Milicevic et al., 2011).

Apesar desta limitação do *Analyzer*, avaliamos alguns cenários que nos permitem ter alguma segurança na nossa especificação.

Em primeiro lugar, *scenario1*, quisemos assegurar que o modelo chegava a um consenso quanto a uma sequência de valores. Assim, estabelecemos que, inevitavelmente, um valor tem de ser escolhido para cada *slot*. Como o escopo tem de ser muito reduzido, fixamos que todos os conjuntos têm 2 elementos, com exceção do quórum, que só tem 1, e das mensagens, que têm 12. Além disso, o *Proposer* somente pode propor um *slot* “não usado” em cada mensagem *M2A*.

No segundo cenário, *scenario2*, testamos se vários *slots* eram propostos e escolhidos numa única ronda. Pelas limitações de escopo tivemos de testar esta situação com apenas dois *slots*. Para avaliarmos este cenário não podemos esquecer de passar dois *slots* como parâmetro à função *mins*.

De seguida, *scenario3*, avaliamos a possibilidade de um participante que não pertence ao quórum poder votar num valor, ou seja, enviar a mensagem *M2B*, sem anteriormente ter recebido a mensagem *M1A* respetiva. Tal como na especificação do *Paxos*, também na do *Multi-Paxos* é possível um participante, desde que respeitadas as devidas restrições, processar uma mensagem *M2A* sem ter processado a correspondente mensagem *M1A*.

O exemplo previsto no *scenario4* refere-se ao caso em que é enviada uma mensagem *Preempt* porque o participante recebeu uma mensagem *M1A* com boletim menor do que o seu *maxBal*. Neste caso quisemos aferir se o participante envia a referida mensagem, sendo esta posteriormente recebida pelo líder.

No *scenario5* quisemos mostrar que se um quórum escolhe determinado conjunto de pares (*Slot, Value*), nas rondas subsequentes os mesmos também terão de ser escolhidos.

Os três cenários seguintes referem-se a cenários inviáveis. No *scenario6* temos um quórum com dois membros, sendo que ambos nunca participaram em nenhuma ronda, pelo que o seu *aBal* é vazio. De seguida, estes participantes processam as mesmas mensagens *M1A* e *M2A* e votam nos pares propostos, no entanto, o conteúdo das suas relações *aVoted* é diferente. Conforme esperado, não há nenhuma instância que satisfaça este cenário. O *scenario7* confirma a impossibilidade de duas mensagens *M2A* terem o mesmo boletim e valores diferentes e o *scenario8* testa o envio de uma mensagem *M2A* repetida.

6.3.3 Vertical Paxos

As validações realizadas para as especificações das duas variantes deste protocolo foram muito similares, pelo que nos abstemos de as analisar separadamente, fazendo, contudo, referência às especificidades de cada uma das versões quando tal for adequado.

Assim, começamos por validar os três cenários descritos nas Secções 5.3.4 e 5.4.4. Em primeiro lugar, *scenario1* (alternativa A), avaliamos o comportamento do sistema quando é necessário transferir o estado para a nova configuração antes desta estar completa ou ativada, conforme a versão do protocolo. Com a

exploração da instância gerada quisemos perceber se, quando existe estado anterior a ser transferido, as Fases de Preparação, Aceitação e Ativação da nova configuração são devida e ordenadamente executadas, de modo a que os valores anteriormente votados sejam oportunamente conhecidos.

Em segundo lugar, `scenario2` (alternativa B), apresentamos um traço de execução compatível com a inexistência de qualquer configuração ativa aquando do início do algoritmo. Neste caso, assim que o líder tem conhecimento desta situação despoleta a Fase de Ativação antes de qualquer outra.

Na terceira situação, `scenario3` (alternativa C), quisemos descrever o cenário em que existe uma configuração ativa, mas ainda não há qualquer valor escolhido. Assim, decorrida a Fase de Preparação do protocolo, o líder aprende que existe, pelo menos, um quórum de leitura que ainda não votou, pelo que pode despoletar imediatamente a Fase de Ativação.

Estes três cenários são os mais importantes porque descrevem os três caminhos mais comuns do modelo, tal como analisamos anteriormente. No entanto, queremos também fazer referência a outros casos que nos parecem relevantes para descrever as suas particularidades, nomeadamente as características dos seus quórums e as diferentes etapas do algoritmo.

Conforme vimos anteriormente, este protocolo distingue dois tipos de quórums: leitura e escrita. A fim de garantir a correção da especificação basta que todos os quórums de leitura de determinada ronda se intersectem com o quórum de escrita dessa mesma ronda. Ao contrário do que acontece no *Paxos*, não há a obrigação de se escolher o valor anteriormente votado por um membro do quórum na ronda anterior com o maior boletim. Com efeito, um cenário impossível no *Paxos* (ver o `scenario7` do *Paxos*) é possível no *Vertical Paxos*. Imaginemos o seguinte, `scenario4`: na primeira ronda temos um quórum de escrita constituído por três participantes, a_1 , a_2 e a_3 que formam, cada um deles, diferentes quórums de leitura referentes à mesma ronda. Na referida ronda foi votado o valor v_1 , por apenas um dos participantes do mencionado quórum de escrita, a_2 . Entretanto, o mestre decidiu reconfigurar novamente o sistema. Nesta segunda ronda, há somente um quórum de leitura e de escrita constituído por um único participante, a_2 . No decurso do algoritmo, o líder contacta um quórum de leitura anterior, por exemplo, o a_1 , sendo que este não votou em nenhum valor. Assim, o líder da segunda ronda pode propor qualquer valor, dado que há, pelo menos, um quórum de leitura da ronda anterior que ainda não votou. Com efeito, independentemente de um participante, que pertence aos quórums de leitura e escrita atuais, ter votado anteriormente, nada impede que seja escolhido um valor diferente de v_1 .

O `scenario5` explora a possibilidade de serem enviadas duas mensagens `MsgNewBallot`, sequenciais e sucessivas, tendo sido completa ou ativada a configuração com o maior boletim. Nada impede que o líder que coordena a configuração com o boletim menor envie ao mestre uma mensagem `MsgComplete`. Porém, esta configuração nunca será considerada completa, na versão do *Vertical Paxos I*, ou ativada, na variante do *Vertical Paxos II*.

Finalmente, os últimos cenários referem-se a instâncias impossíveis. Estes testes negativos examinam a impossibilidade de se completar ou ativar uma reconfiguração sem o conhecimento e intervenção do líder (`scenario6`); a obrigatoriedade de se processar a Fase de Preparação quando se conhece uma ronda anterior que foi considerada ativa (`scenario7`); a impossibilidade do líder enviar uma mensagem `M2A` sem previamente ter enviado uma mensagem `MsgComplete`, (`scenario8`) e, finalmente, a obrigatoriedade de, no caso de um

valor ter sido escolhido numa ronda anterior, ser transferido o estado antes de ser processado o predicado `completeReconfig, (scenario9)`.

6.4 VERIFICAÇÃO DOS MODELOS

A próxima etapa foi a verificação de algumas propriedades dos protocolos.

6.4.1 Paxos

No modelo do *Paxos* a verificação passou por demonstrar as seguintes propriedades:

1. no máximo, um único valor é escolhido pelos elementos do quórum - `check ChosenValue;`
2. se um processo votou num determinado boletim b e correspondente valor v , implica que nenhum valor, exceto v , foi ou será escolhido num boletim inferior a b - `check votesSafe;`
3. cada boletim só pode ter associado um valor - `check oneVote;`

Contudo, ao verificar as referidas propriedades, com diferentes escopos, concluímos que a especificação com mensagens dinâmicas era muito lenta. Por conseguinte, decidimos adaptar o nosso modelo, desenvolvendo uma versão mais eficiente da especificação. Apesar de ser mais complexa e menos inteligível, daí que tenhamos decidido não a introduzir no Capítulo 5, esta versão não modela explicitamente as mensagens, o que acaba por se traduzir numa versão mais leve. Esta versão (*no messages*) tem as assinaturas apresentadas na Especificação 6.10.

```

sig Value {}
sig Ballot {}
sig Quorum {
    acceptors : some Acceptor
}
sig Acceptor {
    var maxBal : lone Ballot,
    var maxVote : lone Vote,
    var sent : Type -> Ballot -> Payload
}
one sig None {}
abstract sig Type {}
one sig M1A,M1B,M2A,M2B extends Type {}
sig Vote {
    value : Value,
    ballot : Ballot
}
sig Payload = Value + Vote + None {}

```

Extrato da Especificação 6.10: Assinaturas da especificação *Paxos no messages*.

Conforme podemos observar, o `Acceptor` tem agora uma relação dinâmica `sent` que armazena os triplos relativos às mensagens enviadas por este. Cada um dos referidos triplos contém o tipo de mensagem, o seu boletim e qualquer informação adicional de que esta necessite, de acordo com o estabelecido na especificação. Desta forma, ficciona-se o envio de uma mensagem, diminuindo-se o número de variáveis que o `solver` tem de tratar. Com efeito, apenas é necessário representar os tipos das mensagens com assinaturas *singleton*, ou seja, conjuntos que contém um único elemento e, portanto, são muito leves.

A assinatura `Payload` representa uma união de assinaturas, podendo apresentar-se, alternativamente, como um valor, um voto ou um conjunto vazio, este último simbolizado pela assinatura `None`. A assinatura `Payload` passa a ser um conjunto que reúne diversas assinaturas com a finalidade de uniformizar o número de campos da relação `sent`.

Ademais, temos ainda a assinatura abstrata `Type` que pode ser instanciada através dos vários subtipos de mensagens do *Paxos*. Na verdade, em *Alloy* ao declarar uma assinatura abstrata cujas suas extensões têm aridade `one` estamos a declarar uma enumeração (Jackson, 2012). Assim, em alternativa, os diversos tipos de mensagens podiam ter sido declarados como `enum`.

Finalmente, para se conseguir que a relação `sent` seja uniforme, nomeadamente que cada mensagem tenha apenas um campo de `Payload`, foi ainda criada uma assinatura que representa um voto, a qual é composta por duas relações identificadoras do valor e boletim votados, `value` e `ballot`, respetivamente. Esta assinatura é estática, bem como as suas relações. Dada a existência desta nova assinatura, foram unificadas as relações `maxVBal` e `maxVal` na relação variável `maxVote`, que armazena o último voto de cada participante.

Com esta nova versão, além dos axiomas relativos ao estado inicial, à formação dos quóruns e dos possíveis eventos, torna-se necessário estabelecer um axioma relativo ao universo dos votos. Como queremos percorrer a estrutura de votos de cada participante e verificar asserções relativas aos mesmos, tivemos de estabelecer um axioma "gerador", que obriga a que todas as combinações de votos possíveis existam. Caso contrário, dado que os `Vote` de cada instância são imutáveis, teríamos de encontrar primeiro a configuração com todos os votos necessários, usando a funcionalidade `New Config`, para poder validar a instância gerada, o que tornaria essa tarefa impossível. Assim, aplicamos a *bounded-universal rule* (Jackson, 2012) aos votos, gerando todas as combinações possíveis de valores e boletins que integram as duas relações da assinatura `Vote`, ver Especificação 6.11. Note-se que esta opção faz com que tenha de existir um átomo para cada combinação de `Vote` possível, o que obriga a algum cuidado na definição do respetivo escopo. Mesmo assim, os resultados de performance desta versão mostraram-se mais satisfatórios do que usando mensagens explícitas (ver análise de desempenho detalhada no Capítulo 7).

```
fact AllPossibleVotesExist {
    all v : Value, b : Ballot | some x : Vote | x.value = v and x.ballot = b
}
```

Extrato da Especificação 6.11: Axioma que impõe a existência de todas as combinações de votos.

A grande desvantagem desta versão é que é confusa para quem não domina a linguagem *Alloy* e talvez dificulte a compreensão do protocolo e do modelo em questão. Para darmos um exemplo da sua complexidade

podemos analisar o predicado `safeVotedValue`, ver Especificação 6.12. O referido predicado verifica se existe algum membro do quórum que tenha votado numa ronda anterior.

```
pred safeVotedValue[q: Quorum, b: Ballot, v: Value] {
    some m1b_bal:((q.acceptors).sent[M1B][b] & value.v).ballot{
        all m1b_ballot:((q.acceptors).sent[M1B][b]).ballot |
            gte[m1b_bal,m1b_ballot]
    }
}
```

Extrato da Especificação 6.12: Predicado `safeVotedValue` na versão *no messages*.

Além das duas variantes descritas do *Paxos*, desenvolvemos ainda uma terceira alternativa, que denominamos de *static messages*, com a particularidade das mensagens serem estáticas, ou seja, as mesmas existem *ab initio*, sendo enviadas ao longo da execução do protocolo. Na verdade, esta versão da especificação do *Paxos*, por ser a mais simples e inteligível, foi o nosso ponto de partida.

No entanto, a dificuldade em validar e iterar sobre as instâncias fez com que pensássemos em novas soluções. O principal problema desta versão é que, como as mensagens das configurações são fixas, obriga a que primeiro se encontre uma configuração, usando a opção `New Config`, que apresente todas as mensagens necessárias para explorar a instância gerada, o que obviamente torna a tarefa impossível. Ademais, não se justifica gerar todas as combinações possíveis de mensagens, dada a complexidade envolvida. Estas dificuldades motivaram a especificação do protocolo com mensagem dinâmicas, *dynamic messages*. Contudo, mantivemos a versão *static messages* para efeitos de verificação e análise de desempenho.

Se compararmos esta alternativa, *static messages*, com a versão *dynamic messages*, verificamos que as diferenças são sutis. A principal diferença é que as mensagens e as suas relações deixam de ser variáveis e passam a ser estáticas. Contudo, para simular o envio e receção das mensagens declaramos uma assinatura `MsgBox` que tem, necessariamente, de ser variável, ver Especificação 6.13.

```
sig Acceptor {
    var maxBal : lone Ballot,
    var maxVal: lone Value,
    var maxVBal: lone Ballot
}
sig Quorum {
    acceptors : some Acceptor
}
sig Value {}
sig Ballot {}
abstract sig Message {
    bal: one Ballot
}
var sig MsgBox in Message{}
sig M1A extends Message{}
sig M1B extends Message{
    acceptor: one Acceptor,
```

```

    mbal: lone Ballot,
    mval: lone Value
}
sig M2A extends Message{
    value: one Value
}
sig M2B extends Message {
    acceptor: one Acceptor,
    value: one Value
}

```

Extrato da Especificação 6.13: Assinaturas da especificação do *Paxos static messages*.

As assinaturas, os axiomas e as ações destas versões são muito semelhantes, conforme podemos constatar através do predicado `phase1B` da Especificação 6.14, não merecendo comentários ou explicações adicionais.

Ademais, gostaríamos de referir que modelamos as diferentes versões anteriormente descritas - *static messages*, *dynamic messages* e *no messages* - para os protocolos *Paxos* e *Vertical Paxos* que estudamos. Esta decisão deveu-se à lentidão e, às vezes, inviabilidade do *Analyzer* em verificar as propriedades dos diferentes protocolos nas versões com mensagens explícitas.

Por último, o protocolo *Multi-Paxos* apenas foi formalizado com duas versões - mensagens estáticas e dinâmicas - pois tornou-se de tal forma complexo e pesado que decidimos abandonar a última versão. Efetivamente, dada a quantidade de relações ternárias e quaternárias, consideramos que este protocolo não iria beneficiar, em termos de eficiência, da modelação sem mensagens explícitas.

```

pred phase1B[a: Acceptor] {
    some m1a: M1A & MsgBox {
        no a.maxBal or gt[m1a.bal, a.maxBal]
        some m1b: M1B {
            m1b.bal = m1a.bal
            m1b.acceptor = a
            m1b.mbal = a.maxVBal
            m1b.mval = a.maxVal
            send[m1b]
        }
        maxBal' = maxBal ++ a->(m1a.bal)
        maxVal' = maxVal
        maxVBal' = maxVBal
    }
}

```

Extrato da Especificação 6.14: Predicado *phase1B* na versão *static messages*.

6.4.2 Multi-Paxos

Na verificação do nosso modelo preocupamo-nos em testar três categorias de asserções referentes: a) ao comportamento genérico do sistema; b) aos participantes e c) às mensagens.

No que se refere às propriedades da primeira categoria, pretende-se verificar que nunca há dois valores diferentes escolhidos para o mesmo slot e boletim correspondente. Para tanto é necessário verificar duas proposições:

1. Se um participante armazena na sua relação `aVoted` o triplo (b,s,v) , então nenhum valor a não ser v foi ou será escolhido em qualquer boletim inferior a b com o slot s .
2. Se dois participantes diferentes votam nos triplos $(b,s,v1)$ e $(b,s,v2)$, respetivamente, então $v1$ e $v2$ são o mesmo valor.

No que se refere aos participantes é necessário que se verifiquem sempre quatro condições cumulativas. Em primeiro lugar, se a relação `aBal` não tiver nenhum boletim no seu contradomínio implica que o participante nunca votou. A segunda condição estabelece que o `aBal` de cada participante tem de ser maior ou igual a todos os boletins que foram votados pelo mesmo. A terceira condição obriga a que se o participante `a` votou no valor v e slot s , ambos associados ao boletim b , então terá de existir na relação `aVoted` desse mesmo participante algum voto no slot s com um boletim associado maior ou igual a b . Por último, o participante não pode ter votado num boletim maior do que o mais elevado que consta na sua relação `aVoted`.

As propriedades referentes às mensagens que queremos verificar distinguem-se entre as que pertencem aos conjuntos de mensagens M_{1B} , M_{2A} e M_{2B} .

Para qualquer mensagem M_{1B} enviada terão de se verificar três requisitos cumulativos:

- o `aBal` do participante que enviou a mensagem tem de ser maior ou igual ao campo `bal` da mensagem em causa;
- os triplos enviados no campo `voted` foram efetiva e anteriormente votados pelo remetente da mensagem;
- o participante que envia a mensagem não vota nos boletins que medeiam o maior boletim votado em determinado slot e o boletim anterior da mensagem M_{1B} em análise. Por outras palavras, imaginemos que estamos na ronda 5 e que, de todos os participantes do quórum, só o $a1$ tem um único voto anterior, $(B2, S1, V1)$. O boletim $B2$ foi o maior boletim votado para o slot $S1$ e com o valor $V1$. Então, a mensagem M_{1B} deste participante conterá o voto $(B2, S1, V1)$. Se no final da ronda 5 se chegar a um acordo, o participante $a1$ terá de ter na sua relação `aVoted` o voto no $(B5, S1, V1)$ porque será o maior boletim votado no slot $S1$. Ora, o que se quer salvaguardar é que, posteriormente, o $a1$ não vote nos boletins entre 2 e 4.

Na Fase de Aceitação também temos restrições relativas às mensagens trocadas. Em primeiro lugar, vamos analisar as condições impostas às mensagens M_{2A} :

- A mensagem M_{2A} não tem slots repetidos;

- Existe no máximo uma mensagem M_{2A} para cada boletim.

Finalmente, as mensagens M_{2B} obrigam a que:

- O voto dos participante em quaisquer tuplos ($Slot, Valor$) pressupõe a sua proposta;
- O boletim da mensagem M_{2B} não pode ser maior do que o valor inscrito na relação $aBal$ do remetente da mensagem.

6.4.3 Vertical Paxos

No que se refere ao protocolo de reconfiguração vertical decidimos verificar as seguintes propriedades, para ambas as variantes:

1. um qualquer quórum de escrita apenas escolhe no máximo um valor;
2. existe apenas um voto por boletim, ou seja, o valor votado no boletim b é sempre o mesmo.
3. pode ser votado qualquer valor desde que nenhum quórum de escrita anterior tenha escolhido valor diferente.

Em primeiro lugar, relembramos que o termo votação refere-se a qualquer mensagem M_{2B} enviada por um participante, ao passo que a noção de escolha implica a votação de todos os membros de um quórum de escrita no mesmo boletim e valor.

A principal propriedade a verificar é a de que apenas um único valor é escolhido (`check ChosenValue`). Na reconfiguração vertical a abstração de quórum foi dividida em quórum de leitura e de escrita, sendo que somente este último escolhe um valor.

Na verdade, comparando esta verificação com a correspondente da especificação *Paxos*, observamos que a única alteração é no predicado `chosenAt`, ver Especificação 6.15, porque temos de referir expressamente que os participantes que votam integram um quórum de escrita.

```
pred chosenAt[b: Ballot, v: Value] {
    some wq: WQuorum | all a: wq.acceptors[b] | votedFor[a, b, v]
}
```

Extrato da Especificação 6.15: Predicado *chosenAt* nas duas variantes do *Vertical Paxos*.

Acresce que, a verificação da segunda propriedade também se faz nos mesmos moldes que no *Paxos*.

De resto, a novidade está em determinar e verificar quando é legítimo votar num certo valor, tendo em consideração o histórico de votações. No que se refere a esta matéria, nas especificações do *Vertical Paxos*, há bastante mais flexibilidade do que na do *Paxos*. Com efeito, no *Vertical Paxos*, enquanto um quórum de escrita não escolher um valor, nada impede que nas rondas subsequentes possa ser proposto e escolhido qualquer valor. Daí que a asserção `assert VotesSafe` afirme que sempre que um participante vota este tem de votar

no mesmo valor anteriormente escolhido por um quórum de escrita ou qualquer outro, caso nenhum valor tenha sido ainda escolhido, ver a Especificação 6.16.

```

pred safeQuorum[b: Ballot, v: Value] {
    some wq: WQuorum | (all a: wq.acceptors[b] | votedFor[a, b, v]) or
        (some a: wq.acceptors[b] | didNotVoteAt[a, b])
}
pred safeAt[b: Ballot, v: Value] {
    all c: prevs[b] | safeQuorum[c, v]
}
assert VotesSafe {
    all a : Acceptor, b : Ballot, v : Value | always (votedFor[a, b, v]
        implies safeAt[b, v])
}
check VotesSafe

```

Extrato da Especificação 6.16: Predicados *safeQuorum* e *safeAt* nas duas variantes do *Vertical Paxos*.

7

AVALIAÇÃO DE DESEMPENHO

Neste capítulo avaliamos o grau de escalabilidade e de desempenho do *Analyzer* no tratamento das especificações apresentadas neste trabalho. Para tal, usamos a técnica de verificação automática limitada e ilimitada na execução do comando `check ChosenValue`, variando o seu escopo, os *solvers* e as estratégias de decomposição.

Na verificação automática limitada usamos os *solvers* *Glucose41*, *Glucose*, *Lingeling*, *PLingeling*, *MiniSat* e, em casos particulares, o *ElectrodX* (*i.e.*, *nuXmv*). Examinamos a performance de cada um com base nas três diferentes estratégias de decomposição de modelos, com exceção dos *solvers* *Lingeling* e *PLingeling* que apenas correram em modo *batch* porque o *Analyzer* não suporta as demais estratégias para estes dois *solvers*. Com efeito, conforme podemos observar na Figura 28, os referidos *solvers* reportam explicitamente um erro no modo híbrido e apesar da estratégia paralela não retornar um erro, nunca chega a processar (*0 vars. 0 primary vars*).

Acresce que, não obstante o *PLingeling*, em modo *batch*, correr localmente com a interface gráfica do *Analyzer*, quando executado nas máquinas utilizadas na avaliação de desempenho detetamos um *bug* que nos impedi de recolher quaisquer dados, conforme se demonstra através da Figura 29.

Por outro lado, a verificação automática ilimitada foi efetuada com o *ElectrodX*, com as três diferentes estratégias de decomposição. Em particular, esta técnica foi utilizada para comparar os tempos de execução da nossa especificação do protocolo *Paxos* com uma especificação do mesmo protocolo em *TLA+* (Kuppe et al.), ferramenta de eleição para especificar e verificar sistemas distribuídos. Convém salientar que a propriedade de *safety* verificada em *TLA+*, equivalente ao *ChosenValue*, tem a designação de *chosen*.

À exceção dos testes que envolveram a verificação automática ilimitada e a ferramenta *TLA+*, foi utilizada uma máquina virtual da *Cloud* da *Oracle*, com *shape* do tipo *VM.Standard.E4.Flex*, 24 *OCPUs*, 388 *GBytes* de Memória, com *Oracle Linux* 8.6-2022.08.29-0. Por incompatibilidade da biblioteca *glibc* com o *solver* *ElectrodX*, foi utilizada uma máquina com o sistema operativo *Oracle-Linux-9.0-2022.08.17-0* para testar a verificação automática ilimitada e o *TLA+*.

Todos os testes foram executados 3 vezes, com as versões 6 do *Alloy* e 1.7.1 do *TLA+*, ambas configuradas com 16 *threads* ou *workers* ($\frac{2}{3}$ do número de *OCPUs*), utilizando a *Java Virtual Machine* 8, com a *flag* *-Xmx16384m*, a qual estabelece que o máximo de memória alocada à *Java Virtual Machine* é de 16 GB. Os tempos apresentados representam a média de execução para cada comando de análise, tendo sido fixado um *Timeout* de execução de 10 minutos para a técnica de verificação automática limitada e de 240 minutos para

```

Option Verbosity changed to low
Option Solver changed to Lingeling
Executing "Check ChosenValue for 0 int 20..20 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor,
Solver=lingeling(jni) Steps=20..20 Bitwidth=0 MaxSeq=0 Symmetry=20 Mode=hybrid
Generating CNF...
A fatal error has occurred:
Unknown exception occurred: java.lang.NullPointerException

Option Decompose strategy changed to parallel
Executing "Check ChosenValue for 0 int 20..20 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor,
Solver=lingeling(jni) Steps=20..20 Bitwidth=0 MaxSeq=0 Symmetry=20 Mode=parallel
0 vars. 0 primary vars. 566ms.
No counterexample found. Assertion may be valid. 0ms.

Option Solver changed to PLingeling
Option Decompose strategy changed to hybrid
Executing "Check ChosenValue for 0 int 20..20 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor,
Solver=plingeling(jni) Steps=20..20 Bitwidth=0 MaxSeq=0 Symmetry=20 Mode=hybrid
Generating CNF...
A fatal error has occurred:
Unknown exception occurred: java.lang.NullPointerException

Option Decompose strategy changed to parallel
Executing "Check ChosenValue for 0 int 20..20 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor,
Solver=plingeling(jni) Steps=20..20 Bitwidth=0 MaxSeq=0 Symmetry=20 Mode=parallel
0 vars. 0 primary vars. 608ms.
No counterexample found. Assertion may be valid. 1ms.

```

Figura 28: Output gerado com os solvers *Lingeling* e *PLingeling* nos modos híbrido e paralelo.

a ilimitada. Os referidos tempos foram estabelecidos considerando a complexidade e grandeza dos escopos envolvidos na verificação.

7.1 PAXOS BENCHMARKS

No que diz respeito à especificação do *Paxos*, foram consideradas as três versões anteriormente mencionadas: *Paxos static messages* - com mensagens estáticas; *Paxos dynamic messages* - com mensagens dinâmicas e *Paxos no messages* - sem mensagens explícitas.

As tabelas que se seguem descrevem a performance do *Analyzer* para as diferentes versões do modelo, variando o escopo do comando em análise, os solvers e a estratégia de decomposição adotada. Os ficheiros com os diferentes escopos de cada modelo estão disponíveis no repositório mencionado no Capítulo 5.

Para cada versão da especificação existem duas tabelas que se referem a diferentes números de participantes, 3 e 4, respetivamente. Na primeira coluna de cada tabela temos o número de passos de execução considerados na análise limitada. O número de passos foi determinado tendo em consideração o número de participantes e o número de mensagens. Em particular, o critério escolhido na determinação do número mínimo

```

Check ChosenValue for 17..17 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote
Running 2 out of 80
Exception: java.util.concurrent.ExecutionException: Fatal error:
Unknown exception occurred: kodkod.engine.AbortedException: kodkod.engine.satlab.SATAbortedException: java.io.IOException: Cannot run program "plingeling": error=2, No such file or directory
Duration Time: Timeout
Command: ChosenValue Some counter-example ----- Timeout
[exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote]
Check ChosenValue for 11..11 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote
Running 3 out of 80
Exception: java.util.concurrent.ExecutionException: Fatal error:
Unknown exception occurred: kodkod.engine.AbortedException: kodkod.engine.satlab.SATAbortedException: java.io.IOException: Cannot run program "plingeling": error=2, No such file or directory
Duration Time: Timeout
Command: ChosenValue Some counter-example ----- Timeout
[exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote]
Check ChosenValue for 21..21 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote
Running 4 out of 80
Exception: java.util.concurrent.ExecutionException: Fatal error:
Unknown exception occurred: kodkod.engine.AbortedException: kodkod.engine.satlab.SATAbortedException: java.io.IOException: Cannot run program "plingeling": error=2, No such file or directory
Duration Time: Timeout
Command: ChosenValue Some counter-example ----- Timeout
[exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote]
Check ChosenValue for 9..9 steps, exactly 2 Value, 3 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 3 Leader, exactly 4 Vote
Running 5 out of 80
Exception: java.util.concurrent.ExecutionException: Fatal error:
Unknown exception occurred: kodkod.engine.AbortedException: kodkod.engine.satlab.SATAbortedException: java.io.IOException: Cannot run program "plingeling": error=2, No such file or directory
Duration Time: Timeout

```

Figura 29: Log com o erro referente ao solver *PLingeling* no modo *batch*.

de passos a analisar teve por referência o menor número de transições de estado necessárias para se chegar a um consenso, sendo o limite máximo fixado no seu dobro. Neste intervalo, vamos sucessivamente incrementando os passos em duas unidades. No caso do *Paxos static messages* e do *dynamic messages*, o número de mensagens trocadas em cada escopo corresponde a menos uma unidade do que os respetivos passos de execução.

A segunda coluna indica o escopo das assinaturas *Value*, *Ballot* e *Quorum*. Por exemplo, o escopo *V2B2Q2* representa exatamente 2 valores, *V2*, exatamente dois boletins, *B2*, e dois quóruns, *Q2*. Note-se que somente o número de quóruns possíveis não é precedido da palavra *exactly*, o que implica que o *Analyzer* terá de verificar todas as configurações possíveis com um determinado número de participantes, de valores e de boletins até ao número de quóruns especificado. Cada uma das referidas assinaturas varia entre dois e três elementos em cada passo de execução.

As restantes colunas descrevem o tempo de execução de cada escopo, com cada um dos *solvers* e estratégias de decomposição indicadas.

No caso do *Paxos dynamic messages* e do *Paxos no messages* existe uma coluna extra que indica o número de configurações existentes para cada escopo definido. Esta informação é fornecida pelo *Analyzer* quando executamos em modo paralelo. Note-se que as configurações do estado inicial dependem exclusivamente do número de variáveis estáticas existentes no modelo, sendo este número igual nestas duas versões do *Paxos*. Daí que o número de configurações a analisar em cada escopo destas duas versões seja equivalente.

Além disso, podemos verificar que as variáveis estáticas que efetivamente influenciam a quantidade de configurações correspondem às assinaturas *Acceptor* e *Quorum*. As razões porque isto acontece prendem-se com diversos fatores:

- a propriedade de *symmetry breaking* do *Analyzer* elimina os casos simétricos, sendo irrelevante a quantidade de valores ou de boletins;
- os quóruns não são precedidos da palavra-chave do *Alloy* *exactly*, o que faz com que o número de configurações aumente há medida que o número de quóruns aumenta;
- o número de participantes é relevante para a formação dos quóruns, tanto mais que definimos um axioma que obriga a que todos os participantes pertençam a um quórum.

```
Option Decompose strategy changed to parallel
Executing "Check ChosenValue for 9..9 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 B
Solver=glucose(jni) Steps=9..9 Bitwidth=4 MaxSeq=4 SkolemDepth=1 Symmetry=20 Mode=parallel
Generating CNF...
An error has occurred!
```

Figura 30: Erro de execução do modo paralelo na versão *static messages*.

```
Command: ChosenValue Some counter-example ----- Timeout
Check ChosenValue for 9..9 steps, exactly 3 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, 8 Message
Running 9 out of 32
Exception in thread "pool-623-thread-2" java.lang.OutOfMemoryError: GC overhead limit exceeded

Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "pool-623-thread-4"
Exception in thread "pool-623-thread-12" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-5" Exception in thread "pool-623-thread-11" java.lang.OutOfMemoryError: GC overhead limit exceeded
java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-15" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-9" [java.lang.OutOfMemoryError: Java heap space]: failed reallocation of scalar replaced objects
Exception in thread "pool-623-thread-10" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-1" [java.lang.OutOfMemoryError: GC overhead limit exceeded]
Exception in thread "pool-623-thread-17" java.lang.OutOfMemoryError: Java heap space failed reallocation of scalar replaced objects
Exception in thread "pool-623-thread-20" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-13" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-8" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "pool-623-thread-14" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception: java.util.concurrent.TimeoutException
Duration Time: Timeout
Command: ChosenValue Some counter-example ----- Timeout
Check ChosenValue for 11..11 steps, exactly 3 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, 10 Message
Skipping 10 out of 32
Duration Time: Timeout
```

Figura 31: Erro de execução do modo paralelo na versão *static messages* após execuções sucessivas.

Assim, apuramos que no caso de 3 participantes, o número de configurações possíveis, para o *Paxos no messages* e para o *dynamic messages*, fixa-se em 5 e 16, consoante do escopo façam parte 2 ou 3 Quorum, respetivamente. No caso de termos 4 participantes, estes valores ascendem a 8 e 44 configurações possíveis.

No que se refere ao *Paxos static messages* foi-nos impossível detetar a quantidade de configurações existentes em cada escopo porque o *Analyzer* apresentou problemas de processamento dada a elevada carga de variáveis estáticas, não conseguindo terminar nenhuma execução no modo paralelo, mesmo com os escopos mais pequenos e com apenas um passo de execução (`1..1 steps`). Na verdade, nesta versão o *solving* bloqueia, tal é a dimensão do conteúdo a analisar em cada configuração, acabando por reportar um erro (ver Figura 30).

Ademais, por diversas vezes, aquando da execução do *Paxos static messages* no modo paralelo, o *Analyzer* gerou inúmeros erros do tipo “`java.lang.OutOfMemoryError: Java heap space`”, ver Figura 31. Aparentemente, estes resultaram de um *leak* de memória que comprometeu todos os resultados do modo paralelo, embora tenhamos configurado a nossa *Java Virtual Machine* para poder utilizar até 16 GB de memória (`-Xmx16384m`). Daí que, as tabelas referentes a esta versão da especificação do protocolo *Paxos* não apresentem resultados relativos a esta estratégia.

No que diz respeito aos tempos de execução, começamos por analisar as Tabelas 1 a 4 relativas às mensagens estáticas e dinâmicas. Os *solvers* *Glucose41* e *Glucose* são os mais eficientes, com a estratégia *batch* a destacar-se ligeiramente das demais. Nota-se ainda que, à medida que os passos vão aumentando, o *MiniSat* vai perdendo competitividade, sendo inclusive o primeiro *solver* a passar a barreira do *Timeout*.

```

Alloy Analyzer [what is new] [spec] 6.1.0 built 2021-11-03T15:25:43.736Z

Option Solver changed to Glucose41
Executing "Check ChosenValue for 9..9 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, 8 Message exp"
Solver=glucose 4.1(jni) Steps=9..9 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=batch
9..9 steps. 41474 vars. 420 primary vars. 80956 clauses. 2638ms.
No counterexample found. Assertion may be valid, as expected. 2513ms.

Executing "Check ChosenValue for 9..9 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, 8 Message"
Solver=glucose 4.1(jni) Steps=9..9 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=batch
9..9 steps. 120498 vars. 1662 primary vars. 231532 clauses. 5346ms.
No counterexample found. Assertion may be valid. 277538ms.

Executing "Check ChosenValue for 0 int 9..9 steps, exactly 2 Value, 2 Quorum, exactly 3 Acceptor, exactly 2 Ballot, exactly 2
Solver=glucose 4.1(jni) Steps=9..9 Bitwidth=0 MaxSeq=0 Symmetry=20 Mode=batch
9..9 steps. 17588 vars. 1222 primary vars. 113978 clauses. 429ms.
No counterexample found. Assertion may be valid. 10ms.

```

Figura 32: Número de variáveis a tratar pelo *Analyzer* nas diferentes versões do *Paxos*.

Quanto à *perfomance* destas duas versões, podemos verificar que os resultados do *Paxos static messages* são muito mais satisfatórios do que os do *Paxos dynamic messages*, sendo a primeira versão ~ 50 vezes mais rápida do que a segunda no caso do *Glucose41*, modo *batch*, com o escopo *V2B2Q2*, 3 participantes e 9 passos. A justificação está relacionada com a quantidade de variáveis que o *Analyzer* tem de gerir. Enquanto, a versão *static messages*, para 3 participantes, 9 passos de execução e escopo *V2B2Q2*, tem de processar 41474 variáveis e 420 variáveis primárias, a versão *dynamic messages* tem de processar 120498 variáveis e 1662 variáveis primárias, ou seja, esta última versão tem de analisar 3 vezes mais variáveis e 4 vezes mais variáveis primárias, ver Figura 32. Aliás, esta discrepância entre os tempos de execução é bem patente se compararmos as Tabelas 2 e 4, onde esta última tabela não apresenta nenhuma entrada diferente de *Timeout*.

Embora a versão *Paxos static messages* seja mais eficiente do que a versão *Paxos dynamic messages*, não escala bem. Na Tabela 1, o aumento de dois passos implica uma degradação substancial nos tempos de execução, havendo uma variação de 1 ou 2 dígitos de ordem de grandeza. Ademais, as entradas da Tabela 2, com 13 passos, são totalmente preenchidas com *Timeout*.

No caso da versão *Paxos no messages*, de acordo com as Tabelas 5 e 6, os solvers *Glucose41* e *Glucose* revelaram-se mais uma vez os mais eficientes. No que se refere à estratégia de decomposição mais apropriada, concluímos que no caso de 3 participantes, ver Tabela 5, são todas muito equivalentes, não existindo nenhuma que se destaque. Todavia, no caso de 4 participantes, Tabela 6, a estratégia *batch* obteve tempos de execução mais baixos. Novamente, o *MiniSat* é o menos escalável, sendo o único a não conseguir, em 10 minutos, apresentar uma solução para alguns escopos com 3 participantes, ver Tabela 5.

A variável com mais impacto no tempo de execução é o participante, pois comparando as Tabelas 5 e 6, constatamos que para o mesmo escopo e número de passos o tempo de execução tem um aumento expressivo. Por exemplo, com a estratégia *batch*, do *Glucose41*, no passo 15, escopo *V3B3Q3*, constatamos que para 3 participantes a verificação é efetuada em 124.2s, ver Tabela 5. Contudo, para 4 participantes demora 293.06s, ver Tabela 6. A explicação para este impacto prende-se com o aumento do número de configurações possíveis.

A segunda variável que mais influencia o desempenho do *Analyzer* é o boletim. Uma justificação plausível para esta situação é o facto de em quase todas as possíveis transições de estado da especificação verificar-se alguma propriedade relacionada com os boletins. Embora o *Quorum* seja o único cujo número de átomos varia na análise de cada comando, o universo de quóruns possíveis a testar é relativamente pequeno e a especificação não o tem em conta tantas vezes quanto os boletins. Daí que não tenha um impacto tão significativo. Um exemplo, entre muitos, que ilustra bem esta realidade é o caso do *Glucose41*, modo *batch*, com 21 passos e 4 participantes, ver Tabela 6. Enquanto o escopo *V2B2Q2*, executa em 9.04s, o escopo *V2B3Q2* demora 40.48s a terminar. Todavia, o escopo *V2B2Q3* leva somente 18.21s.

Acresce que esta versão escala bastante melhor, sendo apenas de reportar a sensibilidade do maior escopo, *V3B3Q3*, a pequenas variações do número de passos. Na Tabela 5, do passo 9 até ao passo 15, e na Tabela 6, do passo 11 até ao 15, a execução do escopo *V3B3Q3* duplica a cada dois passos de intervalo.

Após compararmos as diversas versões do *Paxos*, concluímos que as duas primeiras são completamente inefficientes. Basta olharmos para as tabelas com 4 participantes das diferentes versões, Tabelas 2 e 4, para percebermos que se torna uma tarefa quase impossível extrair resultados. Com efeito, apenas para escopos muito pequenos conseguimos verificar a correção do modelo em tempo útil.

Estes resultados derivam da discrepância do número de variáveis que o *Analyzer* tem de processar nas diversas versões da especificação do protocolo. A Figura 32 ilustra bem esse facto. Note-se que o *Paxos no messages* tem 17588 variáveis e 1222 variáveis primárias, para 9 passos, 4 participantes e *V2B2Q2*.

Por último, não podemos deixar de mencionar que julgamos que as técnicas de decomposição paralela e híbrida, não obstante a sua evolução e estudo (Brunel et al., 2018; Macedo et al., 2017), ainda não se encontram estáveis pois apresentam resultados, por vezes, incongruentes. Efetivamente, à medida que os passos de execução ou a complexidade da especificação aumentam estas estratégias tendem a mascarar erros. Por exemplo, com 4 participantes, 25 passos e escopo *V3B3Q2*, se corrermos o *Paxos no messages* no modo *batch* o *Analyzer* retorna o erro “*Translation capacity exceeded*”. No entanto, as estratégias de decomposição híbrida e paralela, para o mesmo cenário, retornam em tempo recorde que “*No counterexample found. Assertion may be valid.*”, sendo que nada foi, efetivamente, processado (*0 vars. 0 primary vars.*), ver Figura 33.

Passos	Escopo	Glucose41			Glucose			<i>Lingeling</i> Batch	Batch	<i>MiniSat</i> Híbrido	Paralelo
		Batch	Híbrido	Paralelo	Batch	Híbrido	Paralelo				
9	V2B2Q2	3.67	3.7	-	4.26	4.51	-	6.17	4.39	4.52	-
	V2B2Q3	3.23	3.48	-	4.48	4.61	-	5.57	6.22	6.61	-
	V2B3Q2	4.41	5.4	-	4.89	4.87	-	8.22	6.68	6.88	-
	V2B3Q3	6.56	6.88	-	6.68	6.87	-	8.87	12.07	11.15	-
	V3B2Q2	4.42	4.61	-	3.79	3.86	-	7.12	6.32	7.19	-
	V3B2Q3	5.08	6.04	-	6.68	6.83	-	8.61	10.21	10.18	-
11	V3B3Q2	5.44	5.57	-	6.22	6.38	-	11.38	8.25	9.91	-
	V3B3Q3	9.43	9.61	-	9.76	10.51	-	13.12	15.37	18.05	-
	V2B2Q2	38.93	40.46	-	54.68	57.5	-	53.98	97.89	108.6	-
	V2B2Q3	63.43	67.33	-	80.61	82.17	-	65.83	165.32	186.68	-
	V2B3Q2	79.72	86.55	-	80.22	84.38	-	87.02	242.18	290.07	-
	V2B3Q3	107.78	111.59	-	119.83	124.86	-	131.08	288.29	327.16	-
15	V3B2Q2	55.9	58.99	-	75.79	82.47	-	96.11	174.69	190.11	-
	V3B2Q3	88.55	95.34	-	85.93	113.89	-	153.48	236.44	282.73	-
	V3B3Q2	124.21	136.1	-	111.11	134.91	-	222.57	328.95	416.42	-
	V3B3Q3	173.26	191.87	-	228.43	289.94	-	281.67	Timeout	Timeout	-

Tabela 1: Resumo da análise performance (em segundos) da especificação *Paxos static messages*, com 3 participantes.

```

Alloy Analyzer [what is new] [spec] 6.1.0 built 2021-11-03T15:25:43.736Z

Option Decompose strategy changed to batch
Executing "Check ChosenValue for 25..25 steps, exactly 3 Value, 2 Quorum, exactly 4 Acceptor, exactly 3 Ballot, exactly 9 V
Solver=glucose 4.1(jni) Steps=25..25 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=batch
Generating CNF...
A type error has occurred:
Translation capacity exceeded.
In this scope, universe contains 74 atoms
and relations of arity 5 cannot be represented.
Visit http://alloy.mit.edu/ for advice on refactoring.

Option Decompose strategy changed to hybrid
Executing "Check ChosenValue for 25..25 steps, exactly 3 Value, 2 Quorum, exactly 4 Acceptor, exactly 3 Ballot, exactly 9 V
Solver=glucose 4.1(jni) Steps=25..25 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=hybrid
0 vars. 0 primary vars. 703ms.
No counterexample found. Assertion may be valid. 0ms.

Option Decompose strategy changed to parallel
Executing "Check ChosenValue for 25..25 steps, exactly 3 Value, 2 Quorum, exactly 4 Acceptor, exactly 3 Ballot, exactly 9 V
Solver=glucose 4.1(jni) Steps=25..25 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=parallel
0 vars. 0 primary vars. 279ms.
No counterexample found. Assertion may be valid. 12ms.

```

Figura 33: Exemplo de resultados inconsistentes nas estratégias híbrida e paralela.

Passos	Escopo	Batch	Glucose41 Híbrido	Paralelo	Batch	Glucose Híbrido	Paralelo	Lingeling Batch	Batch	MiniSat Híbrido	Paralelo
11	V2B2Q2	47.47	51.19	-	70.08	76.6	-	87.32	124.78	191.24	-
	V2B2Q3	68.35	78.59	-	85.72	93.5	-	103.74	194.49	289.23	-
	V2B3Q2	88.43	144.21	-	109.4	177.74	-	162.39	258.78	370.1	-
	V2B3Q3	138.02	182.47	-	188.79	257.54	-	167.16	442.48	Timeout	-
	V3B2Q2	87.04	124.93	-	99.51	144.95	-	142.81	316.88	495.48	-
	V3B2Q3	118.9	181.06	-	173.56	263.36	-	189.16	363.41	Timeout	-
	V3B3Q2	206.16	302.99	-	205.25	298.01	-	298.17	428.93	Timeout	-
13	V3B3Q3	261.56	378.61	-	371.89	Timeout	-	Timeout	Timeout	Timeout	-
	V2B2Q2	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
	V2B2Q3	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
	V2B3Q2	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
	V2B3Q3	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
	V3B2Q2	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
	V3B2Q3	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
15	V3B3Q2	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-
	V3B3Q3	Timeout	Timeout	-	Timeout	Timeout	-	Timeout	Timeout	Timeout	-

Tabela 2: Resumo da análise performance (em segundos) da especificação *Paxos static messages*, com 4 participantes.

Passos	Escopo	Batch	Glucose41 Híbrido	Paralelo	Batch	Glucose Híbrido	Paralelo	Lingeling Batch	Batch	MiniSat Híbrido	Paralelo	Número de Configurações
9	V2B2Q2	191.14	223.03	238.63	266.37	295.44	246.93	90.72	Timeout	Timeout	Timeout	5
	V2B2Q3	290.29	361.01	283.28	357.42	396.94	305.68	165.74	Timeout	Timeout	Timeout	16
	V2B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	219.75	Timeout	Timeout	Timeout	5
	V2B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	254.5	Timeout	Timeout	Timeout	16
	V3B2Q2	202.94	235.38	228.16	284.92	223.91	176.67	112.42	Timeout	Timeout	Timeout	5
	V3B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	372.75	196.55	Timeout	Timeout	16
	V3B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	259.57	Timeout	Timeout	Timeout	5
11	V3B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	378.48	Timeout	Timeout	Timeout	16
	V2B2Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	394.35	Timeout	Timeout	Timeout	5
	V2B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	16
	V2B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	5
	V2B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	16
	V3B2Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	5
	V3B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	16
15	V3B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	5
	V3B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	16

Tabela 3: Resumo da análise performance (em segundos) da especificação *Paxos dynamic messages*, com 3 participantes.

Passos	Escopo	Glucose41				Glucose				Lingeling		MiniSat			Número de Configurações
		Batch	Híbrido	Paralelo	Batch	Híbrido	Paralelo	Batch	Híbrido	Batch	Híbrido	Batch	Híbrido	Paralelo	
11	V2B2Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
	V2B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V2B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
	V2B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V3B2Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
	V3B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V3B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
13	V2B2Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
	V2B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V2B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
	V2B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V3B2Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	8
	V3B2Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V3B3Q2	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44

Tabela 4: Resumo da análise performance (em segundos) da especificação *Paxos dynamic messages*, com 4 participantes.

Passos	Escopo	Glucose41				Glucose				Lingeling		MiniSat			Número de Configurações
		Batch	Híbrido	Paralelo	Batch	Híbrido	Paralelo	Batch	Híbrido	Batch	Híbrido	Batch	Híbrido	Paralelo	
9	V2B2Q2	1.17	0.88	0.82	1.21	0.73	0.68	5.38	1.14	0.82	1.15	1.14	0.82	1.15	5
	V2B2Q3	1.56	1.24	1.16	1.0	1.06	0.93	5.23	1.18	1.17	0.89	1.18	1.17	0.89	16
	V2B3Q2	3.22	3.04	3.1	2.62	2.65	2.7	11.11	3.21	2.96	2.92	3.21	2.96	2.92	5
	V2B3Q3	5.9	4.58	3.41	4.34	4.11	2.91	11.85	4.45	4.06	3.02	4.45	4.06	3.02	16
	V3B2Q2	1.73	1.72	1.71	1.36	1.33	1.33	9.63	1.77	1.57	1.55	1.77	1.57	1.55	5
	V3B2Q3	2.98	3.01	2.0	2.45	2.36	1.7	9.87	2.83	2.34	2.21	2.83	2.34	2.21	16
	V3B3Q2	6.97	6.7	6.6	5.91	5.33	5.32	11.71	6.42	6.41	6.31	6.42	6.41	6.31	5
11	V2B3Q3	11.5	10.39	7.22	7.87	8.33	5.96	21.55	12.06	8.95	7.84	12.06	8.95	7.84	16
	V2B2Q2	1.53	1.44	1.5	1.41	1.26	1.25	5.29	1.77	1.28	1.29	1.77	1.28	1.29	5
	V2B2Q3	2.92	2.32	2.57	2.31	1.77	1.86	5.65	2.88	2.76	2.76	2.88	2.76	2.76	16
	V2B3Q2	6.45	6.35	6.71	5.27	5.17	5.21	13.51	9.72	6.15	6.05	9.72	6.15	6.05	5
	V2B3Q3	11.09	7.97	7.89	10.5	7.95	8.19	31.21	18.03	9.52	9.52	18.03	9.52	9.52	16
	V3B2Q2	2.98	2.97	3.09	2.72	2.36	2.38	10.69	3.81	3.18	3.02	3.81	3.18	3.02	5
	V3B2Q3	6.0	4.26	3.85	5.48	4.09	4.08	24.16	8.62	8.23	7.93	24.16	8.23	7.93	16
13	V3B3Q2	15.78	13.88	13.83	14.45	13.47	13.31	25.68	20.82	19.39	19.23	20.82	19.39	19.23	5
	V3B3Q3	23.42	18.55	18.58	30.0	16.13	15.77	48.72	53.85	42.01	41.87	48.72	53.85	42.01	16
	V2B2Q2	2.49	2.52	3.0	1.91	1.85	1.9	6.27	2.24	2.19	3.06	2.24	2.19	3.06	5
	V2B2Q3	3.9	3.89	4.47	3.64	2.71	3.04	7.76	5.73	3.47	3.57	5.73	3.47	3.57	16
	V2B3Q2	8.84	9.13	9.96	9.3	7.27	7.72	32.74	16.02	15.7	15.72	16.02	15.7	15.72	5
	V2B3Q3	19.93	15.27	15.01	20.63	15.63	16.23	65.47	36.76	33.66	36.14	36.76	33.66	36.14	16
	V3B2Q2	6.06	5.54	5.51	4.0	4.03	4.28	25.51	6.0	5.92	6.92	6.0	5.92	6.92	5
15	V3B2Q3	10.95	8.45	8.57	11.69	8.23	8.27	38.87	20.4	17.47	17.18	20.4	17.47	17.18	16
	V3B3Q2	26.33	26.4	28.75	30.57	29.32	29.76	56.17	57.03	42.66	42.69	56.17	57.03	42.66	5
	V3B3Q3	58.36	33.38	33.11	65.38	43.89	42.96	133.34	184.76	143.94	143.64	184.76	143.94	143.64	16
	V2B2Q2	3.04	3.11	3.49	3.29	2.81	2.81	8.65	5.69	3.78	3.73	8.65	5.69	3.78	5
	V2B2Q3	5.97	5.38	6.35	6.4	5.47	5.78	11.66	10.66	8.53	8.13	10.66	8.53	8.13	16
	V2B3Q2	12.54	11.85	12.71	12.66	12.66	14.02	42.7	21.78	22.86	27.7	21.78	22.86	27.7	5
	V2B3Q3	38.98	26.44	26.25	45.79	30.66	31.79	110.33	156.04	117.34	117.0	156.04	117.34	117.0	16
17	V3B2Q2	7.04	7.12	7.46	7.36	7.35	7.53	29.25	13.28	13.99	15.1	29.25	13.28	13.99	5
	V3B2Q3	15.57	11.85	11.35	14.91	13.15	13.13	63.22	40.2	33.34	32.73	63.22	40.2	33.34	16
	V3B3Q2	33.82	34.13	34.88	33.97	34.27	40.2	81.47	110.11	117.03	135.26	81.47	110.11	117.03	5
	V3B3Q3	124.2	68.4	68.21	134.04	87.49	84.87	241.99	Timeout	Timeout	505.63	241.99	87.49	84.87	16
	V2B2Q2	4.32	4.39	5.11	4.01	3.86	4.11	11.26	10.72	4.71	4.55	10.72	4.71	4.55	5
	V2B2Q3	8.36	7.42	9.0	10.93	7.03	7.98	14.59	20.02	16.79	17.03	20.02	16.79	17.03	16
	V2B3Q2	16.24	16.89	21.49	15.49	15.78	25.72	94.47	122.85	40.51	44.26	94.47	40.51	44.26	5
19	V2B3Q3	47.87	50.34	51.03	59.48	52.08	52.27	134.75	380.39	338.33	372.31	380.39	338.33	372.31	16
	V3B2Q2	9.29	8.89	9.42	10.7	10.45	10.48	67.36	27.58	26.64	39.64	67.36	27.58	26.64	5
	V3B2Q3	17.56	17.66	17.54	23.59	19.48	19.5	71.86	91.03	75.03	73.0	71.86	91.03	75.03	16
	V3B3Q2	43.91	46.09	48.82	43.09	46.33	62.38	141.54	194.99	204.62	285.73	141.54	194.99	204.62	5
	V3B3Q3	146.69	153.03	154.0	158.79	154.59	154.66	401.82	Timeout	Timeout	Timeout	401.82	Timeout	Timeout	16
20	V2B2Q2	4.72	4.77	5.97	4.99	5.05	5.16	11.25	8.13	8.29	15.01	8.13	8.29	15.01	5
	V2B2Q3	9.9	9.86	12.42	11.96	9.46	10.94	16.01	25.57	26.44	35.46	25.57	26.44	35.46	16
	V2B3Q2	22.8	21.9	24.65	25.66	25.88	27.57	105.29	243.67	136.97	135.48	105.29	243.67	136.97	135.48
	V2B3Q3	64.76	53.9	53.27	102.25	72.55	73.32	186.39	Timeout	Timeout	Timeout	102.25	72.55	73.32	16
21	V3B2Q2	13.35	13.38	14.05	16.75	15.05	15.4	69.23	24.92	26.37	95.44	24.92	26.37	95.44	5
	V3B2Q3	29.49	24.29	23.45	28.69	28.23	30.84	97.2	195.41	140.89	141.91	97.2	195.41	140.89	141.91
	V3B3Q2	63.45	67.12	90.78	67.02	68.63	102.09	182.26	Timeout	Timeout	Timeout	182.26	Timeout	Timeout	5

V3B3Q3	215.78	189.68	190.32	229.1	231.7	221.09	473.45	Timeout	Timeout	Timeout	16
--------	--------	--------	--------	-------	-------	--------	--------	---------	---------	---------	----

Tabela 5: Resumo da análise performance (em segundos) da especificação *Paxos no messages*, com 3 participantes.

Passos	Escopo	Batch	Glucose41		Glucose		Lingeling Batch	Batch	MiniSat		Número de Configurações	
			Híbrido	Paralelo	Híbrido	Paralelo			Híbrido	Paralelo		
11	V2B2Q2	2.78	2.56	2.54	2.13	1.94	1.94	10.1	2.83	2.51	2.45	8
	V2B2Q3	4.1	4.14	10.5	3.84	3.93	9.01	15.8	5.53	5.74	11.78	44
	V2B3Q2	11.28	11.25	10.97	10.17	9.92	9.18	28.84	11.03	11.25	13.88	8
	V2B3Q3	22.08	23.45	42.56	22.0	23.79	36.27	49.63	30.4	32.72	55.19	44
	V3B2Q2	5.68	5.78	5.53	4.19	4.61	5.15	27.6	5.64	5.46	5.4	8
	V3B2Q3	10.3	11.67	23.77	9.26	9.84	21.32	54.1	16.77	19.06	30.54	44
	V3B3Q2	25.55	22.29	21.85	27.4	23.62	22.15	49.21	38.1	34.68	33.37	8
13	V3B3Q3	60.07	66.38	93.57	73.65	80.93	85.65	133.14	138.53	144.51	166.47	44
	V2B2Q2	3.11	3.21	3.98	2.59	2.63	3.15	14.56	3.1	3.26	4.11	8
	V2B2Q3	7.08	7.38	17.5	6.65	6.81	15.71	17.39	12.71	12.5	24.26	44
	V2B3Q2	15.17	15.54	17.16	12.53	13.11	18.71	42.75	31.64	26.04	25.05	8
	V2B3Q3	43.3	46.54	79.3	44.58	47.85	80.51	92.43	168.04	173.49	152.87	44
	V3B2Q2	7.92	7.59	7.25	7.1	7.9	10.33	37.19	12.63	10.79	9.99	8
	V3B2Q3	16.73	18.23	37.45	16.37	18.25	38.71	94.41	49.65	54.15	78.98	44
15	V3B3Q2	39.25	40.15	50.2	43.09	45.82	50.8	112.65	111.87	110.21	102.32	8
	V3B3Q3	143.55	159.49	181.21	153.46	162.37	198.33	308.44	Timeout	Timeout	Timeout	44
	V2B2Q2	4.7	4.76	5.22	3.78	3.81	4.45	14.85	7.76	6.89	6.9	8
	V2B2Q3	9.64	10.07	25.19	9.65	10.01	21.44	26.95	37.31	42.49	48.66	44
	V2B3Q2	19.4	20.7	22.5	23.18	21.77	21.27	64.68	57.5	59.98	57.33	8
	V2B3Q3	81.58	88.08	128.4	76.87	81.55	133.9	177.69	Timeout	Timeout	Timeout	44
	V3B2Q2	10.0	11.12	11.42	10.01	11.32	10.67	42.91	22.35	19.65	19.1	8
17	V3B2Q3	21.34	21.54	57.16	22.83	26.18	56.37	120.28	127.11	140.32	178.39	44
	V3B3Q2	52.11	54.79	60.44	54.66	57.46	85.11	162.33	420.06	248.95	227.0	8
	V3B3Q3	239.06	255.94	352.12	328.11	343.4	435.65	429.14	Timeout	Timeout	Timeout	44
	V2B2Q2	5.66	5.8	6.34	5.39	5.52	6.13	19.29	8.88	9.06	9.71	8
	V2B2Q3	13.17	13.63	33.06	13.34	14.0	32.66	29.56	59.09	65.72	96.76	44
	V2B3Q2	27.53	29.16	29.1	26.38	28.73	30.37	97.15	77.47	82.66	123.46	8
	V2B3Q3	99.07	106.1	175.06	111.64	129.72	202.4	242.85	Timeout	Timeout	Timeout	44
19	V3B2Q2	12.99	14.86	14.38	12.65	14.97	15.28	97.79	43.7	34.53	34.71	8
	V3B2Q3	31.73	33.34	78.32	34.18	35.96	81.89	184.4	Timeout	322.6	319.96	44
	V3B3Q2	77.94	81.33	79.81	79.42	83.31	82.16	217.45	Timeout	Timeout	Timeout	8
	V3B3Q3	330.38	374.81	506.79	408.68	419.04	Timeout	521.2	Timeout	Timeout	Timeout	44
	V2B2Q2	7.01	7.26	9.53	6.54	6.69	8.19	24.39	9.49	10.63	10.6	8
	V2B2Q3	14.32	15.21	45.07	15.78	16.65	43.2	35.22	109.18	116.39	114.5	44
	V2B3Q2	31.07	34.13	41.13	29.36	32.87	42.01	137.87	151.18	153.95	156.35	8
21	V2B3Q3	131.34	140.53	229.43	131.46	157.17	258.75	307.06	Timeout	Timeout	Timeout	44
	V3B2Q2	17.49	19.27	18.8	15.24	17.55	20.44	120.13	41.31	46.44	59.75	8
	V3B2Q3	48.88	54.49	102.03	43.82	48.26	103.38	220.2	Timeout	406.84	Timeout	44
	V3B3Q2	94.71	107.07	106.79	114.99	124.83	127.43	270.8	Timeout	Timeout	Timeout	8
	V3B3Q3	372.45	415.94	Timeout	487.6	511.46	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V2B2Q2	9.04	9.49	12.8	8.12	8.48	10.54	35.86	13.24	14.36	19.45	8
	V2B2Q3	18.21	19.01	56.16	20.04	21.66	52.16	41.36	88.64	96.23	150.67	44
23	V2B3Q2	40.48	44.01	47.89	41.89	47.16	54.23	146.96	134.46	147.53	185.72	8
	V2B3Q3	151.83	178.28	291.43	163.32	172.35	340.49	339.16	Timeout	Timeout	Timeout	44
	V3B2Q2	19.72	22.08	24.08	20.66	22.95	27.39	122.63	45.96	49.51	124.68	8
	V3B2Q3	49.63	54.18	135.9	51.19	59.8	137.74	287.25	Timeout	Timeout	Timeout	44
	V3B3Q2	125.13	134.48	143.04	142.95	153.09	153.49	306.25	Timeout	Timeout	Timeout	8
	V3B3Q3	451.79	520.89	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44
	V2B2Q2	10.46	10.79	12.5	9.4	10.12	12.82	31.39	15.37	16.89	55.25	8
25	V2B2Q3	24.61	26.39	64.14	31.35	34.44	65.88	56.07	130.65	143.25	291.85	44
	V2B3Q2	49.1	56.77	68.98	51.99	59.41	68.35	201.47	151.57	198.94	505.21	8
	V2B3Q3	162.49	178.55	371.38	184.04	218.64	423.85	387.35	Timeout	Timeout	Timeout	44
	V3B2Q2	24.77	25.68	36.91	26.16	28.93	32.92	211.01	75.77	85.94	109.25	8
	V3B2Q3	65.36	68.08	166.89	67.63	79.78	166.21	276.4	Timeout	Timeout	Timeout	44
	V3B3Q2	136.41	149.8	179.97	172.97	186.22	223.4	373.19	Timeout	Timeout	Timeout	8
	V3B3Q3	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	Timeout	44

Tabela 6: Resumo da análise performance (em segundos) da especificação *Paxos no messages*, com 4 participantes.

7.1.1 Análise Comparativa com o TLA+

Dado que a versão do *Paxos no messages* se mostrou a mais eficiente aquando da verificação automática limitada elegemo-la para efetuar a verificação automática ilimitada e ser termo de comparação com o *TLA+*.

Em primeiro lugar, executamos a verificação automática ilimitada do comando `check ChosenValue` na versão *Paxos no messages* com os seguintes escopos para as diferentes estratégias de decomposição:

- `exactly 3 Value, 2 Quorum, exactly 3 Acceptor, exactly 3 Ballot, exactly 9 Vote, 1.. steps`
- `exactly 3 Value, 2 Quorum, exactly 3 Acceptor, exactly 4 Ballot, exactly 12 Vote, 1.. steps`
- `exactly 4 Value, 2 Quorum, exactly 3 Acceptor, exactly 3 Ballot, exactly 12 Vote, 1.. steps`
- `exactly 3 Value, 3 Quorum, exactly 4 Acceptor, exactly 3 Ballot, exactly 9 Vote, 1.. steps`

Os resultados obtidos são apresentados na Tabela 7. Na coluna escopo, a sigla *AiViBiQi* representa o escopo de cada cenário, referindo-se a letra *i* ao número de elementos de cada conjunto.

No que se refere ao *TLA+*, verificamos a propriedade *chosen* da especificação formal do *Paxos*, disponível em (Kuppe et al.). Como o número e configuração dos quóruns é variável tivemos de gerar todas as combinações possíveis dos mesmos para cada um dos escopos testados, tendo sido apenas consideradas as configurações não simétricas (assumimos que o utilizador do *TLA+* é um utilizador experiente e inteligente, abstendo-se de testar os casos equivalentes). Assim, para o primeiro, segundo e terceiro escopos foram geradas cinco possíveis combinações de quóruns e para o último escopo quarenta e quatro.

Os valores obtidos são elucidativos quanto à superioridade do *TLA+*, conforme se pode observar na Tabela 7. Enquanto que a melhor estratégia do *ElectrodX* demora cerca de 3 minutos a verificar se a fórmula *ChosenValue* é satisfazível para o primeiro escopo, o *TLA+* necessita de uns céleres segundos. Para o segundo escopo, o *ElectrodX* demora aproximadamente 9 minutos e o *TLA+* cerca de 2 minutos. Acresce que, para o terceiro escopo, o *TLA+* termina em segundos e o *ElectrodX* demora cerca de 7 minutos. Finalmente, no último escopo, o *ElectrodX* demora cerca de 70 minutos e o *TLA+* responde em aproximadamente 3 minutos. Estas diferenças no tempo de execução são talvez mais percutentes na Figura 34.

Enquanto o *TLA+* utiliza uma representação explícita dos estados, verificando cada um dos traços possíveis, o *Analyzer* utiliza uma representação simbólica, traduzindo as restrições do modelo em variáveis lógicas. Uma justificação plausível para o desempenho do *TLA+* pode estar relacionada com a possível paralelização dos traços e com o pouco não determinismo associado a este protocolo e respetiva especificação, fazendo com que o *TLA+* não saia prejudicado, mesmo tendo de percorrer inúmeros traços de execução. Para comprovar esta teoria, testamos ainda o escopo *A3V3B4Q2*, mas com apenas um *worker* (i.e., *thread*) para os dois *model checkers*. Neste caso, a performance do *TLA+* degrada-se significativamente, demorando cerca de 20 minutos.

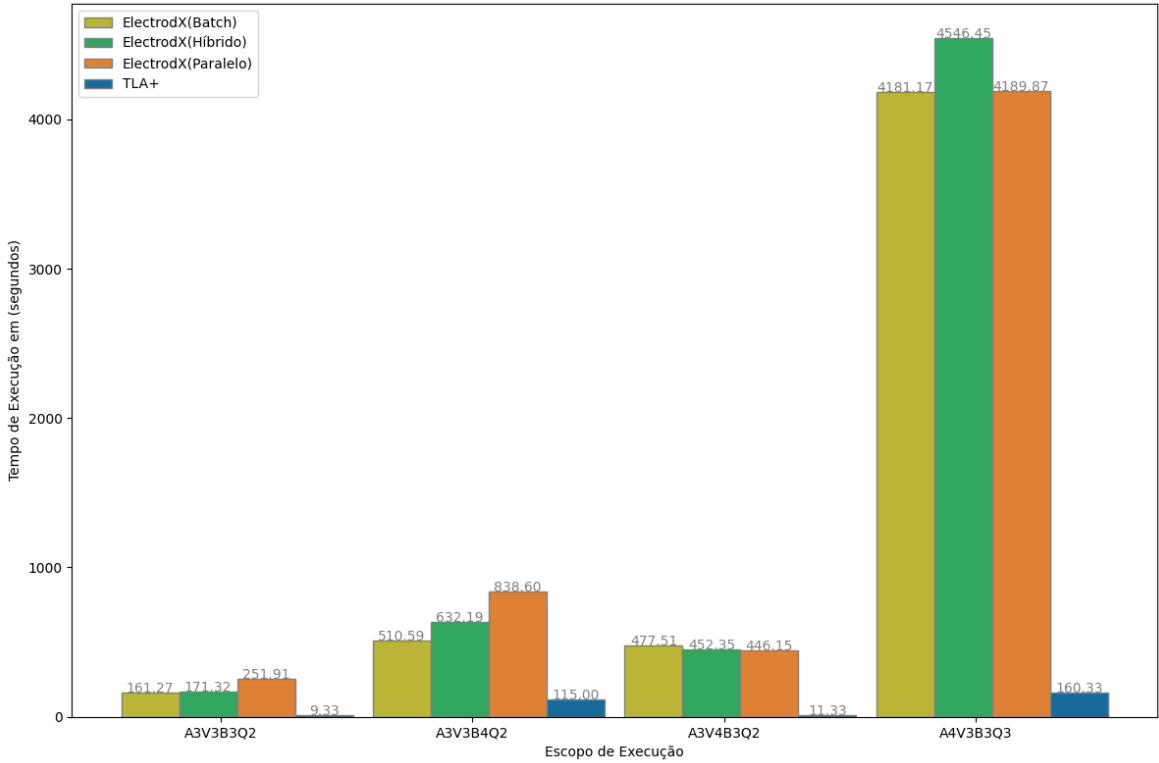


Figura 34: Gráfico com os tempos de execução da verificação ilimitada do *Alloy* e [TLA+](#).

Na verdade, fica muito aquém do *ElectrodX*, que mantém os mesmos resultados para as estratégias de decomposição *batch* e híbrida. Contudo, como era de esperar, a sua estratégia paralela é muito mais lenta ($\approx 40m$), dado que não tem como parallelizar a execução das *partial solutions*. O [TLA+](#) tem pior desempenho porque este cenário gera uma explosão no número de estados a verificar (ultrapassa os 4 milhões de estados distintos) e não há hipótese de paralelismo na execução. Todavia, como o *ElectrodX* beneficia da técnica de *symmetry breaking*, explicada na Secção 4.2.1, continua a ter somente 5 configurações a verificar.

Passos	Escopo	<i>ElectrodX</i>			TLA+
		<i>Batch</i>	Híbrido	Paralelo	
1..	A3V3B3Q2	161.27 \approx (3m)	171.32 \approx (3m)	251.92 \approx (4m)	9.33
1..	A3V3B4Q2	510.59 \approx (9m)	632.19 \approx (11m)	838.60 \approx (14m)	115 \approx (2m)
1..	A3V4B3Q2	477.51 \approx (8m)	452.35 \approx (8m)	446.15 \approx (7m)	11.33
1..	A4V3B3Q3	4181.17 \approx (70m)	4546.45 \approx (76m)	4189.87 \approx (70m)	160.33 \approx (3m)

Tabela 7: Resumo da análise performance (em segundos) do *Alloy* (*Paxos no messages*), usando a técnica de verificação ilimitada, e do [TLA+](#).

7.1.2 *ElectrodX vs. Glucose41*

Para compreender todas as potencialidades do *ElectrodX*, decidimos também avaliar a sua *performance* na sua vertente de *bounded model checker*. Para tirarmos ilações acerca da sua eficiência comparamos os seus tempos de execução com o *Glucose41*, conforme se pode observar na Tabela 8. Este *solver* foi selecionado por

ter apresentado os melhores resultados nas diversas versões do *Paxos*, bem como no *Vertical Paxos*, conforme verificaremos na Secção 7.3. Relembre-se que todos estes testes foram executados numa máquina com o sistema operativo *Oracle-Linux-9.0-2022.08.17-0* e o *Timetout* estendido para 240 minutos.

Se observarmos a Tabela 8, verificamos que as entradas da coluna *Passos* são apresentadas sob a forma $1 \dots n$ *steps*. Isto acontece porque o solver *ElectrodX* obriga a que o número de passos comece em 1, caso contrário gera o erro “*Invalid solver parameters. Electrod bounded model checking must start at length 1*”. Assim, tivemos de correr também o solver *Glucose41* com o mesmo intervalo de passos para fazermos uma comparação justa e criteriosa.

Analisando a Tabela 8, verificamos que nos primeiros intervalos de passos, o *ElectrodX*, no geral, tem melhores resultados. Todavia, à medida que o intervalo de passos aumenta, essa tendência é contrariada. Em concreto, a partir de $1 \dots 17$ passos, a execução dos escopos com maior número de configurações é muito mais rápida com o *Glucose41*. No que se refere à melhor estratégia de decomposição, neste caso, a paralela e a híbrida evidenciam-se. Convém salientar que as execuções foram efetuadas com 16 threads.

Ao decompor uma especificação em diferentes configurações e posteriormente paralelizar a sua exploração, a estratégia paralela acaba por se revelar mais vantajosa. Ao gerar as configurações não simétricas, ou *partial solutions*, existentes no modelo para depois as integrar com a sua parte dinâmica, esta estratégia permite que à medida que o intervalo de passos de execução aumente se tire maior partido do paralelismo. O mesmo acontecendo com o modo híbrido.

De resto, comparando os tempos de execução do *Glucose41* da Tabela 5 com os da Tabela 8, concluímos que a primeira apresenta melhor desempenho. Aliás, como seria de esperar, dado que na segunda tabela a gama de passos a analisar é muito maior. No entanto, com um escopo $1 \dots n$ *steps*, caso a propriedade não fosse válida, seria retornado o menor contra-exemplo possível (em número de transições), o que é vantajoso em termos de compreensão.

<i>Passos</i>	<i>Escopo</i>	<i>ElectrodX</i>			<i>Glucose41</i>			<i>Número de Configurações</i>
		<i>Batch</i>	<i>Híbrido</i>	<i>Paralelo</i>	<i>Batch</i>	<i>Híbrido</i>	<i>Paralelo</i>	
1..9	V2B2Q2	2.19	1.32	1.13	3.27	2.68	2.49	5
	V2B2Q3	1.94	2.08	1.28	3.77	3.94	2.96	16
	V2B3Q2	10.18	5.33	5.34	10.35	9.68	9.68	5
	V2B3Q3	11.09	11.02	6.26	13.94	14.5	11.34	16
	V3B2Q2	4.86	2.56	2.56	5.34	5.24	5.2	5
	V3B2Q3	5.75	5.34	3.01	7.78	7.89	6.02	16
1..11	V3B3Q2	34.42	15.9	15.72	23.35	20.81	22.08	5
	V3B3Q3	44.24	28.46	17.03	29.43	31.96	24.77	16
	V2B2Q2	2.01	1.33	1.25	4.96	4.59	4.58	5
	V2B2Q3	2.63	2.48	1.77	8.13	7.81	6.09	16
	V2B3Q2	11.57	6.09	6.04	20.53	18.14	18.31	5
	V2B3Q3	15.99	11.47	8.66	30.75	28.34	22.58	16
1..13	V3B2Q2	5.61	2.96	2.95	10.55	9.77	9.77	5
	V3B2Q3	8.48	6.92	4.75	17.13	16.38	12.84	16
	V3B3Q2	37.69	17.74	17.97	45.12	41.65	40.03	5
	V3B3Q3	87.72	30.69	26.98	66.09	62.66	50.19	16
	V2B2Q2	2.65	1.63	1.63	8.86	8.49	8.38	5
	V2B2Q3	4.79	3.03	2.85	14.13	12.69	11.84	16
1..15	V2B3Q2	14.77	7.67	7.58	35.85	34.21	33.49	5
	V2B3Q3	35.55	16.54	16.45	64.77	47.49	45.68	16
	V3B2Q2	7.62	4.46	4.23	19.78	19.02	18.78	5
	V3B2Q3	15.99	12.09	12.0	33.56	28.39	25.97	16
	V3B3Q2	52.22	26.35	26.58	85.43	85.03	85.03	5
	V3B3Q3	273.14	73.66	74.31	147.46	106.67	101.65	16
1..17	V2B2Q2	3.78	2.18	2.14	14.11	13.33	13.3	5
	V2B2Q3	10.1	5.6	5.58	23.83	20.93	19.55	16
	V2B3Q2	20.33	14.22	10.5	55.96	54.69	54.28	5

1..15	V2B3Q3	108.32	71.62	72.11	115.25	88.51	85.22	16
	V3B2Q2	11.18	5.57	5.5	32.11	30.41	30.17	5
	V3B2Q3	48.36	44.56	43.99	60.09	47.88	46.2	16
	V3B3Q2	80.57	37.14	36.69	150.24	145.43	142.03	5
	V3B3Q3	803.19	550.27	548.62	335.77	211.97	217.79	16
<hr/>								
	V2B2Q2	5.62	4.14	2.95	20.86	19.9	19.76	5
	V2B2Q3	23.41	11.8	11.65	37.13	33.56	31.48	16
	V2B3Q2	32.33	19.86	19.77	83.57	83.39	83.1	5
1..17	V2B3Q3	397.31	269.02	269.88	202.89	171.28	167.68	16
	V3B2Q2	17.2	8.4	8.34	46.92	43.95	44.29	5
	V3B2Q3	129.63	126.36	128.55	93.63	80.01	78.98	16
	V3B3Q2	143.66	85.49	85.37	219.46	226.12	239.29	5
	V3B3Q3	2408.81	2037.41	1993.44	584.49	479.08	474.41	16
<hr/>								
	V2B2Q2	8.66	4.28	4.25	28.65	27.92	27.5	5
	V2B2Q3	54.52	33.84	20.9	53.7	50.47	48.03	16
	V2B3Q2	54.82	44.8	35.42	117.95	118.63	118.49	5
1..19	V2B3Q3	1509.19	873.47	866.16	318.23	271.28	268.48	16
	V3B2Q2	29.15	17.18	17.06	68.26	64.55	63.99	5
	V3B2Q3	397.38	357.23	339.14	138.42	120.06	118.24	16
	V3B3Q2	258.1	238.62	233.05	317.27	334.93	332.72	5
	V3B3Q3	3958.24	4041.74	5447.63	901.3	775.72	799.61	16

Tabela 8: Resumo da análise performance (em segundos) da especificação *Paxos no messages*, com 3 participantes e com o *ElectrodX* na sua vertente de verificação limitada.

7.2 MULTI-PAXOS BENCHMARKS

O *Multi-Paxos* revelou-se um protocolo tão complexo de modelar, com várias relações ternárias e quaternárias, que tornou-se impossível recolher resultados, visto que na grande maioria das situações o *Analyzer* reportava o erro “*Translation capacity exceeded*” e nos restantes casos não conseguiu manter-se abaixo dos 10 minutos. Por esse motivo, eximimo-nos de apresentar quaisquer resultados.

7.3 VERTICAL PAXOS BENCHMARKS

Tal como no *Paxos*, para cada variante do *Vertical Paxos* foram consideradas três versões: *static messages* - com mensagens estáticas; *dynamic messages* - com mensagens dinâmicas e *no messages* - sem mensagens explícitas. Todavia, atendendo aos resultados anteriores do *Paxos* e de testes preliminares, visto que as duas primeiras versões das especificações do *Vertical Paxos* são mais lentas, iremos somente discutir os dados relativos à versão *no messages* das suas variantes.

Em primeiro lugar, queremos salientar que também no *Vertical Paxos* não nos foi possível correr a estratégia paralela. Tal como no *Paxos static messages*, o *Analyzer* apresentou problemas de processamento com a elevada carga de variáveis estáticas.

Em segundo lugar, visto que o grau de complexidade desta especificação aumentou substancialmente face à especificação do *Paxos*, diversas entradas das tabelas que se seguem contêm a palavra *Error*, a qual representa a impossibilidade do *Analyzer* traduzir a quantidade de variáveis envolvidas no escopo da configuração. Este erro é reportado pelo *Analyzer* como “*Translation capacity exceeded*”.

Note-se que, como o modelo é bastante mais complexo do que o *Paxos*, envolvendo mais intervenientes, mais fases e mais transições, tivemos de alterar o escopo em análise. Assim, fixamos o número de líderes em

2 e, porque a nossa especificação estabelece que, para haver progresso, é necessário pelo menos mais um boletim do que o número de líderes, variamos a quantidade de boletins entre 3 e 4. Acresce que, uma vez que tem de existir, no mínimo, um quórum de escrita e de leitura por ronda, o número de quórums varia entre 6 e 8, quando temos 3 boletins, e entre 8 e 10, quando temos 4 boletins. Finalmente, no que se refere ao intervalo dos passos de execução, mantivemos a mesma estratégia adotada para o *Paxos*, todavia o incremento foi fixado em 5 unidades.

No que se refere aos *solvers* e respetivas estratégias, mais uma vez o *Glucose41* e o *Glucose* destacam-se pela positiva e o *MiniSat* é o primeiro a apresentar *Timeout*. Relativamente à estratégia, há um ligeiro ganho com a escolha da estratégia *batch* em detrimento da estratégia *híbrida*.

Num contexto com o mesmo número de participantes, o valor é a variável com maior impacto nos tempos de execução de ambas as variantes, mas o aumento do número de quórums e passos são os grande responsáveis pelas entradas *Error*.

Convém salientar que, conforme acima mencionado, por vezes, a estratégia *híbrida* mascara erros. Daí que, nas entradas das Tabelas 9 a 12, relativas a esta estratégia, o erro “*Translation capacity exceeded*” foi apresentado como *Timetout*. No entanto, quando a estratégia *batch* reporta *Error* a estratégia *híbrida*, para o mesmo contexto, deveria retornar o mesmo erro.

Conforme podemos observar nas Tabelas 9 e 10, a partir do passo 25 e 21, no caso de 3 e 4 participantes, respetivamente, as entradas das referidas tabelas são povoadas com *Timeout* e *Error*. O mesmo sucede na segunda variante, a partir dos passos 24 e 20, consoante sejam 3 ou 4 participantes, ver Tabelas 11 e 12.

No que se refere à comparação de desempenho das duas variantes, estas são equivalentes, não havendo variações significativas a notar. No entanto, acreditamos que a segunda variante, caso o *Analyzer* permitisse aumentar o número de quórums e de passos, se destacaria pela positiva. Na verdade, ao passo que na segunda variante o líder apenas contacta uma configuração ativa, na primeira pode ter de contactar inúmeras.

A razão subjacente a esta similitude de *performance* das duas variantes relaciona-se com o número de variáveis a processar e com o número de transições necessárias. Enquanto, o *Vertical Paxos II* tem mais variáveis a traduzir, o *Vertical Paxos I* necessita de mais um passo para completar o processo de reconfiguração, ver Figura 35. Por exemplo, no caso concreto de 3 participantes, com o escopo *V2B3Q6*, a variante do *Vertical Paxos I* traduz-se em 189284 variáveis e 13676 variáveis primárias, necessitando de 10 passos. Por seu turno, para o mesmo contexto, o *Vertical Paxos II* gera 192210 variáveis e 14139 variáveis primárias, mas necessita de menos 1 passo, ver Figura 35.

<i>Passos</i>	<i>Escopo</i>	<i>Glucose41</i>			<i>Glucose</i>			<i>Lingeling</i>	<i>MiniSat</i>	
		<i>Batch</i>	<i>Híbrido</i>	<i>Paralelo</i>	<i>Batch</i>	<i>Híbrido</i>	<i>Paralelo</i>		<i>Batch</i>	<i>Híbrido</i>
10	V2B3Q6	6.21	6.31	-	5.49	5.82	-	8.04	6.0	6.29
	V2B3Q8	5.11	5.45	-	5.44	5.89	-	7.23	4.99	5.3
	V2B4Q8	13.85	15.17	-	12.07	13.72	-	16.24	12.04	13.72
	V2B4Q10	12.06	13.62	-	14.94	16.39	-	16.53	11.83	12.67
	V3B3Q6	9.87	10.64	-	10.95	12.3	-	10.96	9.58	10.16
	V3B3Q8	9.72	10.68	-	11.07	12.13	-	15.59	10.16	11.37
	V3B4Q8	21.8	23.98	-	24.78	29.38	-	29.52	22.82	25.62
15	V3B4Q10	28.27	31.25	-	27.73	33.66	-	29.65	21.8	25.11
	V2B3Q6	38.64	47.02	-	20.16	24.59	-	31.82	69.34	85.27
	V2B3Q8	30.84	38.38	-	30.28	39.34	-	31.84	90.88	113.34
	V2B4Q8	69.99	96.02	-	64.03	83.89	-	57.34	144.34	177.44
	V2B4Q10	66.97	86.24	-	73.69	97.86	-	71.18	154.11	193.85
	V3B3Q6	68.65	93.6	-	62.08	79.31	-	67.39	197.08	257.45
	V3B3Q8	69.71	101.59	-	65.02	89.52	-	67.68	236.59	270.05

	V3B4Q8	157.57	224.5	-	148.55	187.43	-	131.35	342.48	421.2	-
20	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	63.08	81.33	-	64.73	86.04	-	72.97	Timeout	Timeout	-
	V2B3Q8	82.02	113.62	-	72.6	92.22	-	71.83	453.56	Timeout	-
	V2B4Q8	180.12	232.73	-	133.23	202.65	-	153.13	Timeout	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	173.25	262.34	-	170.49	255.85	-	136.04	Timeout	Timeout	-
	V3B3Q8	193.99	290.94	-	178.78	273.52	-	130.75	Timeout	Timeout	-
25	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	104.1	146.57	-	113.72	164.34	-	118.99	Timeout	Timeout	-
	V2B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
30	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-

Tabela 9: Resumo da análise performance (em segundos) da especificação *Vertical Paxos I no messages*, com 3 participantes.

Passos	Escopo	Batch	Glucose41		Glucose		Lingeling Batch	Batch	MiniSat		
			Híbrido	Paralelo	Híbrido	Paralelo			Híbrido	Paralelo	
11	V2B3Q6	11.34	12.1	-	8.81	9.73	-	11.41	10.23	10.97	-
	V2B3Q8	12.35	13.57	-	9.76	10.63	-	15.1	10.53	11.44	-
	V2B4Q8	27.94	32.19	-	23.98	27.55	-	29.63	24.75	27.89	-
	V2B4Q10	29.19	32.35	-	23.21	26.98	-	28.41	22.55	24.65	-
	V3B3Q6	24.96	27.89	-	19.75	23.76	-	25.16	20.91	24.24	-
	V3B3Q8	26.44	32.67	-	26.44	30.22	-	25.94	23.58	25.19	-
	V3B4Q8	49.0	62.85	-	43.94	54.59	-	47.06	68.94	75.36	-
16	V3B4Q10	59.39	72.84	-	44.33	52.3	-	50.06	63.07	76.69	-
	V2B3Q6	50.54	69.39	-	51.17	69.62	-	56.17	198.78	275.09	-
	V2B3Q8	52.66	74.76	-	47.11	62.3	-	60.12	190.54	276.8	-
	V2B4Q8	105.67	149.68	-	118.01	154.01	-	126.53	454.37	Timeout	-
	V2B4Q10	144.46	207.24	-	110.81	156.77	-	126.56	307.7	451.15	-
	V3B3Q6	116.9	161.08	-	102.07	159.51	-	115.32	Timeout	Timeout	-
	V3B3Q8	129.33	175.87	-	123.15	186.99	-	134.7	Timeout	Timeout	-
21	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	87.85	119.33	-	116.88	159.43	-	108.89	Timeout	Timeout	-
	V2B3Q8	105.4	144.27	-	95.43	136.56	-	97.22	Timeout	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	215.07	325.83	-	243.61	363.74	-	202.64	Timeout	Timeout	-
26	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
31	V3B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-

	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
--	---------	-------	---------	---	-------	---------	---	-------	-------	---------	---

Tabela 10: Resumo da análise performance (em segundos) da especificação *Vertical Paxos I no messages*, com 4 participantes.

Passos	Escopo	Glucose41		Glucose		Lingeling		MiniSat			
		Batch	Híbrido	Paralelo	Batch	Híbrido	Paralelo	Batch	Híbrido		
9	V2B3Q6	4.87	5.18	-	5.51	5.97	-	6.74	4.75	5.0	-
	V2B3Q8	4.14	4.65	-	4.63	5.54	-	6.27	4.26	4.92	-
	V2B4Q8	9.3	10.09	-	11.71	12.86	-	13.02	9.61	11.14	-
	V2B4Q10	9.5	10.81	-	10.24	11.47	-	13.93	11.27	12.61	-
	V3B3Q6	6.95	7.61	-	8.2	8.95	-	9.29	7.24	8.1	-
	V3B3Q8	7.39	7.88	-	8.25	8.9	-	12.76	6.64	7.2	-
14	V3B4Q8	15.5	16.89	-	21.47	24.6	-	21.44	17.5	20.19	-
	V3B4Q10	16.03	17.86	-	21.02	23.65	-	23.48	16.08	17.91	-
	V2B3Q6	35.41	45.46	-	37.02	45.07	-	34.86	39.41	54.2	-
	V2B3Q8	40.36	65.35	-	35.01	50.03	-	37.78	43.81	63.05	-
	V2B4Q8	67.04	92.94	-	74.08	109.06	-	75.45	93.49	136.47	-
	V2B4Q10	70.24	94.43	-	81.91	111.39	-	78.88	100.71	139.17	-
19	V3B3Q6	72.7	105.74	-	69.47	88.66	-	77.7	136.34	185.13	-
	V3B3Q8	83.15	115.31	-	70.53	102.51	-	93.68	128.06	182.06	-
	V3B4Q8	128.0	198.05	-	148.12	202.21	-	150.19	259.08	376.39	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	84.0	117.38	-	94.27	108.33	-	102.19	251.67	329.37	-
	V2B3Q8	93.49	126.95	-	110.18	145.21	-	88.75	255.4	392.83	-
24	V2B4Q8	193.74	274.09	-	186.69	257.34	-	196.71	Timeout	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	179.61	251.42	-	196.62	258.84	-	203.67	Timeout	Timeout	-
	V3B3Q8	216.15	310.27	-	328.53	449.81	-	194.26	Timeout	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
24	V2B3Q6	144.15	192.91	-	179.1	238.98	-	174.89	Timeout	Timeout	-
	V2B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
20	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	62.67	89.84	-	69.47	96.7	-	68.94	119.33	179.62	-
	V2B3Q8	77.2	105.23	-	70.59	98.71	-	70.12	145.14	195.61	-
	V2B4Q8	146.53	196.21	-	154.42	211.36	-	142.42	240.46	327.82	-
	V2B4Q10	135.28	191.16	-	163.43	251.0	-	150.02	302.74	458.48	-
20	V3B3Q6	132.18	182.39	-	204.36	263.92	-	151.02	405.48	Timeout	-
	V3B3Q8	194.08	307.25	-	168.35	235.86	-	159.17	389.68	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	125.96	167.91	-	118.08	164.6	-	136.96	Timeout	Timeout	-
	V2B3Q8	147.76	201.93	-	149.96	209.12	-	160.78	312.6	462.31	-
20	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	300.38	439.68	-	333.27	527.25	-	323.81	Timeout	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
20	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-

Tabela 11: Resumo da análise performance (em segundos) da especificação *Vertical Paxos II no messages*, com 3 participantes.

Passos	Escopo	Glucose41		Glucose		Lingeling		MiniSat			
		Batch	Híbrido	Paralelo	Batch	Híbrido	Paralelo	Batch	Híbrido		
10	V2B3Q6	7.75	8.55	-	8.56	9.31	-	13.36	9.09	9.95	-
	V2B3Q8	9.05	10.06	-	8.96	9.67	-	12.09	8.47	9.14	-
	V2B4Q8	18.18	21.5	-	25.2	28.91	-	34.97	17.66	20.53	-
	V2B4Q10	19.99	22.37	-	25.39	29.43	-	36.1	19.88	22.29	-
	V3B3Q6	15.85	18.76	-	22.86	27.49	-	28.86	15.01	17.6	-
	V3B3Q8	14.56	17.46	-	21.39	25.14	-	34.83	19.63	21.01	-
15	V3B4Q8	46.21	54.37	-	53.59	64.07	-	65.61	40.2	47.94	-
	V3B4Q10	43.15	52.03	-	56.94	72.12	-	66.04	37.84	43.85	-
	V2B3Q6	62.67	89.84	-	69.47	96.7	-	68.94	119.33	179.62	-
	V2B3Q8	77.2	105.23	-	70.59	98.71	-	70.12	145.14	195.61	-
	V2B4Q8	146.53	196.21	-	154.42	211.36	-	142.42	240.46	327.82	-
	V2B4Q10	135.28	191.16	-	163.43	251.0	-	150.02	302.74	458.48	-
20	V3B3Q6	132.18	182.39	-	204.36	263.92	-	151.02	405.48	Timeout	-
	V3B3Q8	194.08	307.25	-	168.35	235.86	-	159.17	389.68	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q6	125.96	167.91	-	118.08	164.6	-	136.96	Timeout	Timeout	-
	V2B3Q8	147.76	201.93	-	149.96	209.12	-	160.78	312.6	462.31	-
20	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	300.38	439.68	-	333.27	527.25	-	323.81	Timeout	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
20	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-

	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
25	V2B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
30	V2B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V2B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q6	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B3Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q8	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-
	V3B4Q10	Error	Timeout	-	Error	Timeout	-	Error	Error	Timeout	-

Tabela 12: Resumo da análise performance (em segundos) da especificação *Vertical Paxos II no messages*, com 4 participantes.

```

Alloy Analyzer [what is new] [spec] 6.1.0 built 2021-11-03T15:25:43.736Z

Option Solver changed to Glucose
Executing "Check ChosenValue for 10..10 steps, exactly 6 Quorum, exactly 2 Leader, exactly 3 Acceptor, exactly 3 Ballot, exactly
Solver=glucose(jni) Steps=10..10 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=batch
10..10 steps. 189284 vars. 13676 primary vars. 1564294 clauses. 6863ms.
No counterexample found. Assertion may be valid, as expected. 5408ms.

Executing "Check ChosenValue for 9..9 steps, exactly 6 Quorum, exactly 2 Leader, exactly 3 Acceptor, exactly 3 Ballot, exactly
Solver=glucose(jni) Steps=9..9 Bitwidth=4 MaxSeq=4 Symmetry=20 Mode=batch
9..9 steps. 192210 vars. 14139 primary vars. 1788285 clauses. 4133ms.
No counterexample found. Assertion may be valid, as expected. 3102ms.

```

Figura 35: Exemplo do número de variáveis envolvidas no *Vertical Paxos I e II*.

8

CONCLUSÃO

Neste trabalho especificamos em *Alloy* os protocolos de acordo distribuído *Paxos*, *Multi-Paxos* e as duas variantes do protocolo de reconfiguração *Vertical Paxos*. Esta é uma das primeiras, senão mesmo a primeira, formalização dos referidos protocolos nesta linguagem de especificação e pretende ser um contributo para um melhor entendimento das diferentes fases de cada um dos referidos protocolos.

O *Alloy* revelou-se uma linguagem muito flexível e rica. Ao incorporar noções como a de hierarquia, herança e tipos abstratos caracteriza-se por ser maleável e capaz de se ajustar facilmente às especificidades de um sistema real que se pretenda modelar. As referidas noções foram especialmente importantes na definição dos quóruns e das mensagens, permitindo agrupar os correspondentes elementos comuns numa “classe” e partilhá-los por herança, o que evita redundância e, consequentemente, maior propensão a erros.

A riqueza em elementos estruturais e a expressividade a nível comportamental do *Alloy* faz com que seja simples traduzir diferentes tipos de relações (estáticas, variáveis, abstratas), a sua aridez e *signature facts*, bem como descrever o comportamento do modelo, restringindo as transições de estado ou condicionando uma sequência de eventos através dos diversos operadores temporais.

Na validação das especificações dos referidos protocolos foram utilizados diversos mecanismos e estratégias oferecidas pela linguagem e pela sua ferramenta de visualização. Uma das estratégias utilizadas foi a criação de funções com o propósito de definir relações auxiliares para efeitos de visualização. Estas funções adicionais, depois de criadas, são captadas pela funcionalidade *Theme*, destacando-se facilmente alguma informação relevante para a compreensão do comportamento do modelo. Convém salientar que estas relações auxiliares foram o veículo para desenvolvermos um idioma de eventos que simplificou a interpretação dos traços de execução.

Ademais, além dos operadores de lógica temporal comumente conhecidos, esta linguagem dispõe de uma gama mais alargada, tais como *historically* ou *once*, que facilitam a modelação. Estas fórmulas temporais foram muito importantes na validação, visto que quisemos avaliar relações mutáveis, designadamente o envio e receção de mensagens ou o voto de um participante. Na construção de cenários específicos foi crucial o operador temporal sequencial ; (ponto e vírgula). Este permitiu criar eventos com sequências temporais aninhadas, assim como fixar uma ordem na transição de estados.

Outro recurso de validação que queremos mencionar é a possibilidade de, especificamente, avaliar cenários pela negativa. Dado que o *Alloy* assenta numa lógica de *model finding*, devolvendo o primeiro traço de execução que obedece às restrições, conseguimos definir um comando *run* cujo comportamento colide com o nosso modelo, colocando no seu escopo a palavra-chave *expect* 0. Assim, ao não encontrar uma instância que

satisfaça o referido cenário, o *Alloy* garante que aquele comportamento não é permitido face às restrições impostas na nossa especificação. À medida que a especificação ganha complexidade, este tipo de testes tornam-se úteis, pois garantem que eventuais alterações não permitem comportamentos indesejados.

A cresce que, também a ferramenta *Analyzer* mostrou-se muito útil na validação das especificações. Esta dispõe de funcionalidades que nos permitiram depurar e explorar os diversos traços de execução gerados, bem como avaliar cada relação, predicado ou variável em cada transição. Entre as diversas técnicas utilizadas destacamos: a possibilidade de visualizar a sequência de transições de estado (i.e., traço) e percorrer a mesma; o *New Fork* que permite alterar o traço de execução a partir de determinado estado; o *Theme* que permite destacar ou omitir, personalizando com cores e formas, uma relação, e o *Evaluator* que permite examinar qualquer expressão relacional da especificação.

Ao longo deste processo detetamos algumas inconsistências e erros subtils que sem a ajuda destas funcionalidades seriam difíceis de corrigir, bem como limitações nos próprios protocolos (ver Secção 5.4.4). Além disso, foi no decurso da validação da primeira versão da especificação do protocolo *Paxos* (*Paxos static messages*) que nos apercebemos que a exploração do traço de execução era impossível. Esta versão obrigava, primeiramente, a encontrar uma configuração inicial com um número suficiente de mensagens que nos permitisse analisar todas as restrições. Esta dificuldade motivou a criação de uma nova versão desta especificação com mensagens dinâmicas (*Paxos dynamic messages*), as quais, sendo geradas à medida que as fases do protocolo avançam, tornam mais simples aplicar os diversos mecanismos de exploração que a linguagem oferece.

Após termos concluído a formalização dos protocolos e validado diversos cenários que nos permitiram adquirir alguma confiança no modelo, o passo natural seguinte foi a sua verificação.

O *Alloy* oferece a opção de escolha entre diversos *solvers* e três estratégias de decomposição, bem como a seleção entre duas técnicas de verificação automática, limitada e ilimitada. A verificação das especificações centrou-se em analisar propriedades de *safety*, com vista a consolidar a confiança e segurança na correção das mesmas. Nos diversos protocolos aferimos, entre outras propriedades, se, na eventualidade de acordo, todos os participantes votam num único valor e se cada participante, por cada ronda do protocolo em questão, vota uma única vez.

No entanto, a verificação do protocolo *Paxos* com mensagens dinâmicas revelou-se demasiado morosa, pelo que foi necessário aperfeiçoar esta versão. A estratégia passou por abstrair a troca de mensagens (*Paxos no messages*). Estas continuam a existir, mas cada tipo de mensagem existente corresponde a um *singleton*, i.e., um conjunto que contém um único elemento. O foco deixa de ser o envio e a receção da mensagem, mas que mensagens o participante conhece, passando este a armazenar as mensagens por si enviadas. Com estas alterações foi possível verificar as diversas propriedades de *safety*.

De salientar que, pelas razões anteriormente expostas, estas diversas versões do *Paxos* foram transpostas para as especificações das variantes do *Vertical Paxos*. No entanto, o *Multi-Paxos* foi somente formalizado com duas versões (*static* e *dynamic messages*) porque o protocolo é tão complexo que entendemos que não iria beneficiar da terceira versão.

Posteriormente, averiguamos a escalabilidade e desempenho do *Analyzer* na verificação das referidas especificações. No que se refere à verificação automática limitada foram testados os *solvers* *Glucose41*, *Glucose*,

Lingeling, *MiniSat* e, em casos particulares, o *ElectrodX*. A técnica de verificação ilimitada foi efetuada com o solver *ElectrodX*.

Após compararmos as diversas versões do protocolo *Paxos*, concluímos que as versões com mensagens estáticas e dinâmicas são completamente ineficientes. Aliás, as mesmas são sensíveis a pequenas variações de escopo e a segunda não escala. Ademais, observamos que as técnicas de decomposição paralela e híbrida ainda não se encontram totalmente estáveis pois apresentam erros de execução e resultados inconsistentes. Em breve pretendemos reportar oficialmente os *bugs* que encontramos no decurso deste trabalho.

A especificação do protocolo *Multi-Paxos* mostrou-se demasiado complexa, tendo sido impossível recolher dados quanto aos tempos de execução, quer porque o escopo excedia a capacidade de tradução do *Analyzer*, quer porque ultrapassava o limite máximo de tempo fixado.

No que se refere ao protocolo de reconfiguração vertical, as duas variantes do *Vertical Paxos* mostram resultados equivalentes. No entanto, a complexidade deste modelo não permitiu testes com grandes variações de escopo, pois a quantidade de variáveis envolvidas facilmente ultrapassa a capacidade de tradução do *Analyzer*.

Além disso, dos diversos testes realizados, conclui-se que os solvers *Glucose41* e *Glucose* são os mais eficientes, tendo o *MiniSat* o menor grau de escalabilidade.

Acresce que, para compreender todas as potencialidades do *ElectrodX* decidimos também avaliar a sua *performance* na vertente limitada. Para tirarmos ilações comparámo-lo, nas mesmas condições de processamento, com o solver *Glucose41*. Nesta experiência observamos que nos primeiros intervalos de passos, o *ElectrodX*, no geral, tem melhores resultados. Todavia, à medida que o intervalo de passos aumenta, essa tendência é contrariada. No que se refere à melhor estratégia de decomposição, neste caso, a paralela e a híbrida evidenciam-se, pois conseguem tirar partido do processamento paralelo das soluções.

No que se refere à verificação automática ilimitada, decidimos realizar uma análise comparativa do *ElectrodX* com o *TLA+*. Esta provou a superioridade do *TLA+* em termos de desempenho em todos as situações analisadas. No entanto, não podemos deixar de referir que o *ElectrodX* efetua uma verificação exaustiva, i.e., numa única análise examina todas as possíveis configurações no âmbito do escopo definido. Por seu turno, o *TLA+* somente analisa a configuração dada como parâmetro. Assim, o *ElectrodX* acaba por oferecer mais segurança no modelo, pois dispensa o utilizador da tarefa de pensar em todos os *corner cases*.

Face ao exposto, concluímos que o *Analyzer*, com a técnica de representação simbólica dos estados, pode não ser o mais apropriado para verificar protocolos de sistemas distribuídos. De facto, as técnicas de *model checking* como a implementada no *model checker TLC* do *TLA+* parecem ser mais apropriadas para este tipo de problema. Todavia, em algumas situações, como é o caso do aumento de boletins sem a possibilidade de execução paralela (i.e., só com um worker), a técnica de representação simbólica aliada ao mecanismo de *symmetry breaking* do *Analyzer* mostram-se mais eficientes.

Ademais, os mecanismos de validação do *Alloy*, a verificação exaustiva e as funcionalidades do *Analyzer* fazem desta linguagem uma das mais adequadas no estudo e compreensão dos comportamento do modelo, sendo extremamente úteis no desenvolvimento, validação e verificação do mesmo.

Por último, como trabalhos futuros propomos a implementação de uma versão simplificada do protocolo *Multi-Paxos* que permita avançar com o seu processo de verificação. Além disso, consideramos importante a criação de uma versão do *Vertical Paxos* que embeba um protocolo de [MER](#), como o *Multi-Paxos*.

BIBLIOGRAFIA

- Marcos Kawazoe Aguilera. Stumbling over consensus research: Misunderstandings and issues. In Bernadette Charron-Bost, Fernando Pedone, and André Schiper, editors, *Replication: Theory and Practice*, volume 5959 of *Lecture Notes in Computer Science*, pages 59–72. Springer, 2010. doi: 10.1007/978-3-642-11294-2_4. URL https://doi.org/10.1007/978-3-642-11294-2_4.
- alloy, 2011. URL <https://alloytools.org/alloy6.html>. Accessed: 2022-01-02.
- Asad-ur-rehman, Rui L. Aguiar, and João Paulo Barraca. Fault-tolerance in the scope of cloud computing. *IEEE Access*, 10:63422–63441, 2022. doi: 10.1109/ACCESS.2022.3182211. URL <https://doi.org/10.1109/ACCESS.2022.3182211>.
- Julien Brunel, David Chemouil, Alcino Cunha, and Nuno Macedo. The electrum analyzer: model checking relational first-order temporal specifications. In Marianne Huchard, Christian Kästner, and Gordon Fraser, editors, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 884–887. ACM, 2018. doi: 10.1145/3238147.3240475. URL <https://doi.org/10.1145/3238147.3240475>.
- Julien Brunel, David Chemouil, Alcino Cunha, and Nuno Macedo. Simulation under arbitrary temporal logic constraints. In Rosemary Monahan, Virgile Prevosto, and José Proen  a, editors, *Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM 2019, Porto, Portugal, 7th October 2019*, volume 310 of *EPTCS*, pages 63–69, 2019. doi: 10.4204/EPTCS.310.7. URL <https://doi.org/10.4204/EPTCS.310.7>.
- Julien Brunel, David Chemouil, Alcino Cunha, and Nuno Macedo. Structural design with alloy - formal software design with alloy 6, 2021. URL <https://haslab.github.io/formal-software-design/index.html>. Accessed: 2021-12-30.
- Christian Cachin, Rachid Guerraoui, and Lu  s E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming* (2. ed.). Springer, 2011. ISBN 978-3-642-15259-7. doi: 10.1007/978-3-642-15260-3. URL <https://doi.org/10.1007/978-3-642-15260-3>.
- Wei Cao, Zhenjun Liu, Peng Wang, Sen Chen, Caifeng Zhu, Song Zheng, Yuhui Wang, and Guoqing Ma. Polarfs: An ultra-low latency and failure resilient distributed file system for shared storage cloud database. *Proc. VLDB Endow.*, 11(12):1849–1862, 2018. doi: 10.14778/3229863.3229872. URL <http://www.vldb.org/pvldb/vol11/p1849-cao.pdf>.
- Wei Cao, Yang Liu, Zhushi Cheng, Ning Zheng, Wei Li, Wenjie Wu, Linqiang Ouyang, Peng Wang, Yijing Wang, Ray Kuan, Zhenjun Liu, Feng Zhu, and Tong Zhang. POLARDB meets computational storage: Efficiently

support analytical workloads in cloud-native relational database. In Sam H. Noh and Brent Welch, editors, *18th USENIX Conference on File and Storage Technologies, FAST 2020, Santa Clara, CA, USA, February 24-27, 2020*, pages 29–41. USENIX Association, 2020. URL <https://www.usenix.org/conference/fast20/presentation/cao-wei>.

Saksham Chand, Yanhong A. Liu, and Scott D. Stoller. Formal verification of multi-paxos for distributed consensus. In *FM 2016: Formal Methods*, pages 119–136. Springer International Publishing, 2016. doi: 10.1007/978-3-319-48989-6_8. URL https://doi.org/10.1007%2F978-3-319-48989-6_8. Submitted on 4 Jun 2016 (version 1), last revised 11 Nov 2019 (this version, version 4)].

Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996. doi: 10.1145/226643.226647. URL <https://doi.org/10.1145/226643.226647>.

Tushar Deepak Chandra, Robert Griesemer, and Joshua Redstone. Paxos made live: an engineering perspective. In Indranil Gupta and Roger Wattenhofer, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 398–407. ACM, 2007. doi: 10.1145/1281100.1281103. URL <https://doi.org/10.1145/1281100.1281103>.

Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. ISBN 978-3-319-10574-1. doi: 10.1007/978-3-319-10575-8. URL <https://doi.org/10.1007/978-3-319-10575-8>.

Alcino Cunha and Nuno Macedo. Validating the hybrid ERTMS/ETCS level 3 concept with electrum. In Michael J. Butler, Alexander Raschke, Thai Son Hoang, and Klaus Reichl, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 6th International Conference, ABZ 2018, Southampton, UK, June 5-8, 2018, Proceedings*, volume 10817 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2018. doi: 10.1007/978-3-319-91271-4_21. URL https://doi.org/10.1007/978-3-319-91271-4_21.

Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. volume 32, pages 374–382, 1985. doi: 10.1145/3149.214121. URL <https://doi.org/10.1145/3149.214121>.

Eli Gafni and Leslie Lamport. Disk paxos. In *Distributed Computing: 14th International Conference, DISC 2000, Maurice Herlihy, editor. Lecture Notes in Computer Science number 1914, Springer-Verlag, (2000) 330-344.*, volume 16, pages 1–20, May 2003. URL <https://www.microsoft.com/en-us/research/publication/disk-paxos/>.

Rachid Guerraoui and Nancy A. Lynch. A general characterization of indulgence. In Ajoy Kumar Datta and Maria Gradinariu, editors, *Stabilization, Safety, and Security of Distributed Systems, 8th International Symposium, SSS 2006, Dallas, TX, USA, November 17-19, 2006, Proceedings*, volume 4280 of *Lecture Notes in Computer Science*, pages 16–34. Springer, 2006. doi: 10.1007/978-3-540-49823-0_2. URL https://doi.org/10.1007/978-3-540-49823-0_2.

- Rachid Guerraoui and Michel Raynal. A generic framework for indulgent consensus. In *23rd International Conference on Distributed Computing Systems (ICDCS 2003), 19-22 May 2003, Providence, RI, USA*, pages 88–95. IEEE Computer Society, 2003. doi: 10.1109/ICDCS.2003.1203455. URL <https://doi.org/10.1109/ICDCS.2003.1203455>.
- Rachid Guerraoui, Michel Hurfin, Achour Mostéfaoui, Rui Carlos Oliveira, Michel Raynal, and André Schiper. Consensus in asynchronous distributed systems: A concise guided tour. In Sacha Krakowiak and Santosh K. Srivastava, editors, *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*, volume 1752 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 1999. doi: 10.1007/3-540-46475-1_2. URL https://doi.org/10.1007/3-540-46475-1_2.
- Heidi Howard and Richard Mortier. Paxos vs raft: Have we reached consensus on distributed consensus? *CoRR*, abs/2004.05074, 2020. URL <https://arxiv.org/abs/2004.05074>.
- Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. Flexible Paxos: Quorum Intersection Revisited. In Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone, editors, *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*, volume 70 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-031-6. doi: 10.4230/LIPIcs.OPODIS.2016.25. URL <http://drops.dagstuhl.de/opus/volltexte/2017/7094>.
- Michael Isard. Autopilot: automatic data center management. *ACM SIGOPS Oper. Syst. Rev.*, 41(2):60–67, 2007. doi: 10.1145/1243418.1243426. URL <https://doi.org/10.1145/1243418.1243426>.
- Daniel Jackson. Nitpick: A checkable specification language. In *Workshop on Formal Methods in Software Practice, San Diego, CA, January 1996*, pages 60–69. Proc. First ACM SIGSOFT, 1996. URL <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.5678&rep=rep1&type=pdf>.
- Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002. doi: 10.1145/505145.505149. URL <https://doi.org/10.1145/505145.505149>.
- Daniel Jackson. *Software Abstractions - Logic, Language, and Analysis - Revised edition*. MIT Press, 2012. ISBN 978-0-262-52890-0.
- Daniel Jackson. Alloy: a language and tool for exploring software designs. *Commun. ACM*, 62(9):66–76, 2019. doi: 10.1145/3338843. URL <https://doi.org/10.1145/3338843>.
- Daniel Jackson, Somesh Jha, and Craig Damon. Isomorph-free model enumeration: A new method for checking relational specifications. *ACM Trans. Program. Lang. Syst.*, 20(2):302–343, 1998. doi: 10.1145/276393.276396. URL <https://doi.org/10.1145/276393.276396>.
- Jan Z. Konczak, Paweł T. Wojciechowski, Nuno Santos, Tomasz Zurkowski, and André Schiper. Recovery algorithms for paxos-based state machine replication. *IEEE Trans. Dependable Secur. Comput.*, 18(2):623–640, 2021. doi: 10.1109/TDSC.2019.2926723. URL <https://doi.org/10.1109/TDSC.2019.2926723>.

- Markus Alexander Kuppe, Novemser, Kirill Melhesedek, and Igor Konnov. Paxos.tla. URL <https://github.com/tlaplus/Examples/blob/master/specifications/Paxos/Paxos.tla>. Accessed: 2022-01-02.
- Leslie Lamport. If you're not writing a program, don't use a programming language. URL <http://bulletin.eatcs.org/index.php/beatcs/article/view/539>. Accessed: 2022-01-12.
- Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Trans. Software Eng.*, 3(2):125–143, 1977. doi: 10.1109/TSE.1977.229904. URL <https://doi.org/10.1109/TSE.1977.229904>.
- Leslie Lamport. The temporal logic of actions. Technical Report 79, May 1994. URL <https://www.microsoft.com/en-us/research/publication/the-temporal-logic-of-actions/>. ACM Transactions on Programming Languages and Systems 16.
- Leslie Lamport. The part-time parliament. May 1998. URL <https://www.microsoft.com/en-us/research/publication/part-time-parliament/>. ACM SIGOPS Hall of Fame Award in 2012.
- Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, December 2001. URL <https://www.microsoft.com/en-us/research/publication/paxos-made-simple/>.
- Leslie Lamport. *Specifying Systems: the TLA+ language and tools for hardware and software engineers*. Pearson Education, Inc, 2003. ISBN 0-321-14306-X. pages 1 and 225.
- Leslie Lamport. Lower bounds for asynchronous consensus. Technical Report MSR-TR-2004-72, July 2004. URL <https://www.microsoft.com/en-us/research/publication/lower-bounds-for-asynchronous-consensus/>.
- Leslie Lamport. Generalized consensus and paxos. Technical Report MSR-TR-2005-33, March 2005. URL <https://www.microsoft.com/en-us/research/publication/generalized-consensus-and-paxos/>.
- Leslie Lamport. Fast paxos. *Distributed Computing*, 19:79–103, October 2006. URL <https://www.microsoft.com/en-us/research/publication/fast-paxos/>.
- Leslie Lamport. Computation and state machines. April 2008. URL <https://www.microsoft.com/en-us/research/publication/computation-state-machines/>.
- Leslie Lamport and Mike Massa. Cheap paxos. In *International Conference on Dependable Systems and Networks (DSN 2004)*, June 2004. URL <https://www.microsoft.com/en-us/research/publication/cheap-paxos/>.
- Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. Technical Report MSR-TR-2009-63, May 2009a. URL <https://www.microsoft.com/en-us/research/publication/vertical-paxos-and-primary-backup-replication/>.
- Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. In Sri-kanta Tirthapura and Lorenzo Alvisi, editors, *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009*, pages 312–

313. ACM, 2009b. doi: 10.1145/1582716.1582783. URL <https://dahliamalkhi.files.wordpress.com/2015/12/verticalpaxosba-podc2009.pdf>.
- Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Reconfiguring a state machine. *SIGACT News*, 41(1):63–73, 2010. doi: 10.1145/1753171.1753191. URL <https://lamport.azurewebsites.net/pubs/reconfiguration-tutorial.pdf>.
- Barbara Liskov and James Cowling. Viewstamped replication revisited. Technical Report MIT-CSAIL-TR-2012-021, MIT, July 2012. <http://pmg.csail.mit.edu/papers/vr-revisited.pdf>.
- Jacob R. Lorch, Atul Adya, William J. Bolosky, Ronnie Chaiken, John R. Douceur, and Jon Howell. The SMART way to migrate replicated stateful services. In Yolande Berbers and Willy Zwaenepoel, editors, *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006*, pages 103–115. ACM, 2006. doi: 10.1145/1217935.1217946. URL <https://doi.org/10.1145/1217935.1217946>.
- Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. ISBN 1-55860-348-4.
- Nuno Macedo, Julien Brunel, David Chemouil, Alcino Cunha, and Denis Kuperberg. Lightweight specification and analysis of dynamic systems with rich configurations. In Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su, editors, *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 373–383. ACM, 2016. doi: 10.1145/2950290.2950318. URL <https://doi.org/10.1145/2950290.2950318>.
- Nuno Macedo, Alcino Cunha, and Eduardo Pessoa. Exploiting partial knowledge for efficient model analysis. In Deepak D’Souza and K. Narayan Kumar, editors, *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, volume 10482 of *Lecture Notes in Computer Science*, pages 344–362. Springer, 2017. doi: 10.1007/978-3-319-68167-2_23. URL https://doi.org/10.1007/978-3-319-68167-2_23.
- Dahlia Malkhi, Leslie Lamport, and Lidong Zhou. Stoppable paxos. Technical Report MSR-TR-2008-192, April 2008. URL <https://www.microsoft.com/en-us/research/publication/stoppable-paxos/>.
- Aleksandar Milicevic, Derek Rayside, Kuat Yessenov, and Daniel Jackson. Unifying execution of imperative and declarative code. In Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic, editors, *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 511–520. ACM, 2011. doi: 10.1145/1985793.1985863. URL <https://doi.org/10.1145/1985793.1985863>. <http://people.csail.mit.edu/aleks/website/papers/icse11-squander.pdf>.
- Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In Michael Kaminsky and Mike Dahlin, editors, *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP ’13, Farmington, PA, USA, November 3-6, 2013*, pages 358–372. ACM, 2013. doi: 10.1145/2517349.2517350. URL <https://doi.org/10.1145/2517349.2517350>.

- Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, 2015. doi: 10.1145/2699417. URL <https://doi.org/10.1145/2699417>. <https://www.amazon.science/publications/how-amazon-web-services-uses-formal-methods>.
- Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 305–319, Philadelphia, PA, June 2014. USENIX Association. ISBN 978-1-931971-10-2. URL <https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro>.
- Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. doi: 10.1145/322186.322188. URL <http://doi.acm.org/10.1145/322186.322188>.
- Roberto De Prisco, Butler W. Lampson, and Nancy A. Lynch. Revisiting the PAXOS algorithm. *Theor. Comput. Sci.*, 243(1-2):35–91, 2000. doi: 10.1016/S0304-3975(00)00042-6. URL [https://doi.org/10.1016/S0304-3975\(00\)00042-6](https://doi.org/10.1016/S0304-3975(00)00042-6).
- Derek Rayside, Felix Sheng-Ho Chang, Greg Dennis, Robert Seater, and Daniel Jackson. Automatic visualization of relational logic models. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 7, 2007. doi: 10.14279/tuj.eceasst.7.94. URL <https://doi.org/10.14279/tuj.eceasst.7.94>.
- Fred B. Schneider. The state machine approach: A tutorial. In *Fault-Tolerant Distributed Computing*, volume 448 of *Lecture Notes in Computer Science*, pages 18–41. Springer, 1986.
- Fred B. Schneider. What good are models and what models are good? In S. Mullender, editor, *Distributed Systems*, volume 1, chapter 2, pages 17–25. Addison-Wesley, Boston, MA, USA, 2 edition, Maio de 1993. URL <https://www.cs.cornell.edu/fbs/publications/MullenderChptr2.pdf>.
- Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647. Springer, 2007. doi: 10.1007/978-3-540-71209-1_49. URL https://doi.org/10.1007/978-3-540-71209-1_49. <https://homes.cs.washington.edu/emina/pubs/kodkod.tacas07.pdf>.
- David Turner. Unbounded pipelining in dynamically reconfigurable paxos clusters. 2016. URL <http://tessanddave.com/paxos-reconf-latest.pdf>.
- Dirk van Dalen. *Logic and Structure*. Springer, fifth edition, 2013. ISBN 978-1-4471-4557-8. doi: 10.1007/978-1-4471-4558-5. pages 66-67.
- Robbert van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3):42:1–42:36, 2015. doi: 10.1145/2673577. URL <https://doi.org/10.1145/2673577>.

- Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems*. Maarten van Steen, third edition, 2018. ISBN 978-90-815406-2-9 (digital version).
- Marko Vukolic. The origin of quorum systems. *Bull. EATCS*, 101:125–147, 2010. URL <http://eatcs.org/beatcs/index.php/beatos/article/view/183>.
- Marko Vukolic. *Quorum Systems: With Applications to Storage and Consensus*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012. doi: 10.2200/S00402ED1V01Y201202DCT009. URL <https://doi.org/10.2200/S00402ED1V01Y201202DCT009>.
- Michael Whittaker, Neil Giridharan, Adriana Szekeres, Joseph M. Hellerstein, Heidi Howard, Faisal Nawab, and Ion Stoica. Matchmaker paxos: A reconfigurable consensus protocol [technical report]. *CoRR*, abs/2007.09468, 2020. URL <https://arxiv.org/abs/2007.09468>.
- Michael J. Whittaker, Aleksey Charapko, Joseph M. Hellerstein, Heidi Howard, and Ion Stoica. Read-write quorum systems made practical. In *PaPoC@EuroSys 2021, 8th Workshop on Principles and Practice of Consistency for Distributed Data, Online Event, United Kingdom, April 26, 2021*, pages 7:1–7:8. ACM, 2021. doi: 10.1145/3447865.3457962. URL <https://doi.org/10.1145/3447865.3457962>. <https://mwhittaker.github.io/publications/quoracle.pdf>.