Um arcabouço em TLA+ para especificação e verificação de algoritmos distribuídos usando o modelo Heard-Of

Yuri de Souza Pazin[1], **Fernando Luis Dotti**[1]

[1]Programa de Pós-Graduação em Ciência da Computação
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

yuri.pazin@edu.pucrs.br,fernando.dotti@pucrs.br

Abstract. *Distributed fault-tolerant algorithms are essential in many critical systems, making their formal verification indispensable. The TLA+ has been increasingly adopted for this purpose, despite the challenges faced by developers, such as the inherent complexity of using formalisms and the choice of the appropriate abstraction level: too detailed models suffer from state space explosion, while simplified models can compromise the validity of formal verification.*
The Heard-Of model is an important abstraction for specifying and verifying round-based distributed algorithms, allowing both integrated reasoning about synchrony and failures and helping to mitigate state space explosion. The Heard-Of model represents the sets of messages that arrive at each process in a round, and not the order between them, leading to a significant reduction in the state space.
Given the importance of the TLA+ and the Heard-Of model, in this paper we propose a framework that provides the main abstractions of the Heard-Of model in TLA+. This framework is exemplified and evaluated using the Uniform Voting *algorithm, for which properties and their verifications through* model checking *are also discussed, evaluating the state space size and time taken for its construction.*

Resumo. *Algoritmos distribuídos tolerantes a falhas são essenciais em diversos sistemas críticos, tornando sua verificação formal indispensável. O TLA+ tem sido crescentemente adotado para esse fim, apesar dos desafios enfrentados pelos desenvolvedores, como a complexidade inerente ao uso de formalismos e a escolha do nível de abstração adequado: modelos muito complexos sofrem com a explosão do espaço de estados, enquanto modelos simplificados podem comprometer a validade da verificação formal.*
O modelo Heard-of é uma abstração importante para a especificação e verificação de algoritmos distribuídos baseados em rodadas, permitindo tanto o raciocínio integrado sobre sincronia e falhas quanto mitigar a explosão do espaço de estados. O modelo Heard-of representa os conjuntos de mensagens que chegam a cada processo em uma rodada, e não a ordem entre elas, levando a uma importante redução no espaço de estados.
Dada a importância de TLA+ e do modelo Heard-of, neste artigo propomos um framework que provê as abstrações principais do modelo Heard-of em TLA+. Este framework é exemplificado e avaliado com base no algoritmo de Uniform Voting*, para o qual também são discutidas propriedades e suas verificações por* model checking *avaliando o espaço de estados construido e tempo necessários para tal.*

1. Introdução

Algoritmos distribuídos tolerantes a falhas apresentam desafios significativos em seu projeto e análise. Em sistemas críticos que utilizam esses algoritmos, a aplicação de técnicas de verificação formal para assegurar sua corretude é fortemente recomendada por órgãos de fiscalização como a ANVISA [Agência Nacional de Vigilância Sanitária (ANVISA) 2020] e, em alguns casos, constitui até mesmo um requisito, como na norma DO-178C [RTCA 2011], amplamente adotada na indústria aeroespacial.

A técnica de verificação formal conhecida como *model checking* tem se mostrado uma abordagem poderosa para a análise de sistemas distribuídos tolerantes a falhas. Ela permite verificar, de maneira exaustiva, se modelos de estados finitos atendem a especificações formais, como propriedades de *safety* e *liveness*.

O TLA+ [Lamport 2002] é amplamente adotado tanto na indústria quanto na academia [Bögli et al. 2025] para especificação e verificação formal de sistemas distribuídos. A linguagem TLA+ possibilita a modelagem rigorosa de sistemas concorrentes e distribuídos, permitindo a verificação de suas propriedades críticas. Grandes empresas, como a *Amazon Web Services* (AWS), utilizam o TLA+ para identificar falhas em sistemas antes de sua implementação em escala [Newcombe et al. 2014], demonstrando sua relevância prática em ambientes reais e complexos.

Uma das razões para a popularidade do TLA+ é seu alto nível de abstração, que facilita a especificação de comportamentos e propriedades de sistemas distribuídos sem depender de detalhes específicos de implementação. A expressividade e o rigor do TLA+ demonstram seu poder como ferramenta de especificação formal de sistemas distribuídos. Outra questão importante neste contexto é o conceito de refinamento: o TLA+, assim como outras abordagens, suporta a noção de refinamento de um modelo. O refinamento relaciona um modelo abstrato e um concreto, com mais detalhes, mostrando que os comportamentos do concreto estão contidos no sistema abstrato especificado. Sucessivos passos de refinamento podem ser empregados para levar um modelo abstrato a uma implementação real, garantindo que as propriedades especificadas no modelo abstrato estão contidas na implementação.

Apesar das vantagens da verificação de sistemas distribuídos usando *model checking*, a verificação de algoritmos de consenso tolerantes a falhas usando esta abordagem apresenta desafios. Algoritmos de consenso normalmente geram um espaço de estado infinito em ambientes assíncronos, o que inviabiliza a utilização de *model checking* para verificar suas propriedades [Tsuchiya and Schiper 2007]. Esta limitação pode ser contornada restringindo o modelo verificado, por exemplo, limitando o número de passos da execução do algoritmo ou impondo restrições aos canais de comunicação. Esta técnica de verificação de modelos com restrições é denominada *bounded model checking* (BMC), e proporciona apenas uma verificação parcial de suas propriedades. No entanto, existem métodos de redução que possibilitam a representação simbólica de execuções infinitas de um algoritmo em um espaço de estados finito, permitindo a verificação completa das propriedades do modelo. Neste artigo, focamos no uso do modelo *Heard-Of* (HO) para especificação de algoritmos de consenso distribuído tolerantes a falhas benignas.

O modelo *Heard-Of* [Charron-Bost and Schiper 2009] oferece uma abstração que combina a sincronia e as falhas de um sistema distribuído em uma única estrutura: o conjunto *Heard-Of*. Em cada rodada de execução de um algoritmo distribuído, o conjunto *Heard-Of* de um processo especifica de quais processos ele recebeu mensagens. Esta abstração reduz significativamente o espaço de estados do sistema, pois considera apenas o efeito que falhas de transmissão têm em algoritmos de consenso, sem considerar a origem da falha.

As abstrações do modelo *Heard-Of* podem ser aplicadas juntamente com o conceito de refinamento de uma especificação. As execuções com um número infinito de rodadas de um algoritmo podem ser mapeadas para execuções com um espaço de estados finito. As propriedades do consenso deste sistema finito podem então ser verificadas automaticamente através de *model checking* [Chaouch-Saad et al. 2009].

Dada a crescente aplicação de TLA+ para especificação de algoritmos distribuídos e a necessidade de redução do espaço de estados para a viabilização da realização de *model checking* em algoritmos mais complexos, surge a principal contribuição deste trabalho: um *framework* em TLA+, constituído de uma parte genérica, que implementa o modelo *Heard-Of* e suas abstrações; e uma parte específica, onde um algoritmo de consenso distribuído e as condições do ambiente são especificadas. Com estes módulos, é possível realizar a verificação automática do algoritmo pela exploração exaustiva de seu espaço de estados pelo TLC.

O *framework* apresentado é ilustrado com um exemplo de aplicação: o algoritmo de *Uniform Voting* [Charron-Bost and Schiper 2009], modelado com as abstrações do modelo *Heard-Of* em TLA+. A análise de propriedades do consenso usando estas abstrações e o TLC também é discutida.

Embora existam *frameworks* que utilizam abstrações baseadas no modelo *Heard-Of*, nenhum deles modela o sistema distribuído sem definir explicitamente os conjuntos *Heard-Of* como variáveis de estado. Nosso *framework* adota uma abordagem diferente, na qual os conjuntos são construídos implicitamente, o que reduz significativamente o espaço de estados do sistema, sem perda de expressividade quanto às propriedades de consenso. Isso torna o processo de verificação dessas propriedades mais eficiente e viável, especialmente em cenários nos quais o ambiente modelado dá origem a grandes variações nos conjuntos *Heard-Of*.

A organização do texto é a seguinte: a fundamentação teórica na Seção 2; trabalhos relacionados na Seção 3; a Seção 4 apresenta a contribuição deste trabalho: o *framework* proposto, juntamente com sua aplicação ao exemplo; na Seção 5 apresentamos experimentos para a reprodução de resultados utilizando o *framework*; e, por fim, na Seção 6, seguem-se as considerações finais e os direcionamentos dos futuros trabalhos desta pesquisa.

2. Fundamentação teórica

Este artigo parte do pressuposto que o leitor está familiarizado com alguns conceitos de sistemas distribuídos, como por exemplo: processos comunicantes por mensagens; modelos de sincronia; modelos de falhas; o problema do consenso; noções de construção de algoritmos distribuídos.

Além dos conceitos de sistemas distribuídos, é importante que o leitor esteja familiarizado com os princípios de *model checking*. Isso inclui noções de especificação formal de sistemas, mais especificamente TLA+, noções de lógica temporal, a construção e exploração de espaços de estados, a verificação automática de propriedades e o problema da explosão do espaço de estados.

O Modelo *Heard-Of*: O modelo *Heard-Of* é um modelo computacional com a finalidade de facilitar a produção de provas formais de algoritmos de consenso distribuído tolerantes a falhas benignas. Apresentado em [Charron-Bost and Schiper 2009], as principais características do modelo são: (1) o grau de sincronia e o modelo de falhas são representados de forma unificada por meio de uma única abstração: *Os conjuntos Heard-Of*, e (2) o conceito de componente com falha (sendo esse um processo ou canal) é substituído pela noção de falhas de transmissão, ignorando a causa das falhas do sistema mas mantendo os efeitos destas falhas na transmissão de mensagens.

A abstração central neste modelo são os conjuntos *Heard-Of*. Seja um conjunto Π de processos, $p \in \Pi$ e uma rodada r, o conjunto *Heard-Of* $HO(p, r)$ representa o conjunto de processos em Π dos quais p recebeu mensagem na rodada r.

A execução de um algoritmo distribuído no modelo HO ocorre em rodadas, cada uma composta por três etapas, onde cada processo p do conjunto de processos Π:

1. Aplica ao seu estado atual σ_p^r uma função S_p^r, definida pelo algoritmo, que computa as mensagens a serem enviadas e as envia para outros processos.
2. Recebe mensagens enviadas na rodada atual pelos processos pertencentes ao conjunto $HO(p, r)$. Mensagens recebidas de outras rodadas são descartadas.
3. Aplica uma função de transição de estado T_p^r, definida pelo algoritmo, usando seu estado atual σ_p^r e o vetor parcial das mensagens recebidas no round atual como parâmetros. O resultado da aplicação da função define a transição realizada pelo processo e, por consequência desta transição, seu estado na próxima rodada $\sigma_p^{(r+1)}$.

A abstração do conjunto *Heard-Of* utilizada na segunda etapa da computação possibilita simular falhas de transmissão da seguinte maneira: Se um processo p não recebe a mensagem de q em uma rodada r, seu conjunto *Heard-Of* $HO(p, r)$ não contêm q. Assim, o modelo HO consegue abstrair completamente a origem das falhas de comunicação.

Os conjuntos *Heard-Of* de cada computação são originados por um *Predicado de Comunicação*, este predicado modela o comportamento do ambiente em que os processos executam o algoritmo. O predicado estabelece condições invariantes no tempo sobre os possíveis conjuntos HO da computação, originando uma coleção de conjuntos HO que repeitam dado predicado de comunicação. Dois exemplos de predicado que serão utilizados nas demonstrações deste artigo são:

No-Split: A intersecção dos conjuntos HO de quaisquer dois processos do sistema distribuído é diferente do conjunto vazio.

$$\mathcal{P}_{nosplit} :: \forall r > 0, \forall p, q \in \Pi^2 : HO(p, r) \cap HO(q, r) \neq \emptyset,$$

Space Uniform: Os conjuntos HO de quaisquer dois processos do sistema distribuído são iguais.

$$\mathcal{P}_{sp\_unif} :: \forall r > 0, \forall p, q \in \Pi^2 : HO(p, r) = HO(q, r),$$

Com isto, tanto o modelo de sincronia e o modelo de falhas do sistema são representados pelo predicado de comunicação. Predicados análogos aos modelos clássicos de sincronia e falhas podem ser obtidos na literatura [Hutle and Schiper 2007] ou construídos de acordo com a necessidade de modelar ambientes mais complexos.

3. Trabalhos Relacionados

Nesta seção, apresentamos uma revisão dos principais trabalhos acadêmicos que utilizam abstrações do modelo *Heard-Of* no contexto de verificação automática de algoritmos de consenso distribuídos tolerantes a falhas benignas. A revisão é organizada cronologicamente, destacando as abordagens empregadas para reproduzir o modelo *Heard-Of* em ferramentas de *model checking*, com ênfase nas contribuições que mais se aproximam ou influenciam a proposta deste trabalho.

No trabalho [Tsuchiya and Schiper 2007], os autores apresentam sua abordagem para especificação de algoritmos de consenso baseados no modelo *Heard-Of* através da ferramenta *NuSMV* [Cimatti et al. 2002]. A principal contribuição desse trabalho é demonstrar, pela primeira vez na literatura, que técnicas de *model checking* podem ser utilizadas para verificar algoritmos de consenso assíncronos. Para isso, os autores adotam uma modelagem simbólica que utiliza as abstrações do modelo *Heard-Of* como variáveis explícitas do sistema. Esta abordagem permite a verificação de propriedades do consenso, como *agreement* e *termination*, através da exploração exaustiva do espaço de estados gerado pelo algoritmo e os predicados especificados. Apesar do trabalho demonstrar a capacidade de abstração que o modelo HO proporciona, os métodos utilizados possuem limitações quanto à modularidade e escalabilidade: A especificação do algoritmo requer modificações significativas no código para alterar o número de processos e/ou o predicado de comunicação do sistema a ser verificado, além disso, a modelagem explicita dos conjuntos *Heard-Of* gera uma explosão do espaço de estados com o aumento do número de processos do sistema.

Em [Chaouch-Saad et al. 2009], com co-autoria de Bernadette Charron-Bost, autora principal do Modelo *Heard-Of* [Charron-Bost and Schiper 2009], explora-se a verificação por *model checking* de algoritmos de consenso por rodada expressos no modelo HO usando TLA+. Os principais resultados apresentados são:

1. O sistema a ser verificado mantêm suas propriedades locais[1] quando descrito apenas pelo estado local de cada processo.
2. As propriedades do consenso podem ser expressas como propriedades locais.
3. Uma especificação na linguagem TLA+ contendo: o modelo *Heard-Of*, as propriedades do consenso e um algoritmo distribuído especificado no modelo HO.
4. Resultados da verificação formal do algoritmo especificado em TLA+ usando o *model checker* TLC.

Em [Barbosa et al. 2021], os autores propõem um *framework* reutilizável para verificação formal de algoritmos baseados em rodadas utilizando o *model checker* SPIN. O *framework* é dividido em uma parte genérica, que modela a computação em rodadas

[1]Propriedades que podem ser verificadas pela evolução das variáveis internas dos processos do sistema durante a execução do algoritmo

baseada no modelo HO; e uma parte específica, que modela o algoritmo, especificado pelas suas funções de envio de mensagem e transição de estado. Além disso, uma metodologia de tradução para SMT é proposta para permitir o uso de verificadores simbólicos. Embora as computações por rodada sejam baseadas no modelo HO, nota-se a ausência de suas principais abstrações: os conjuntos *Heard-Of* e os predicados de comunicação. Apesar disso, sua enfase em modularidade e a separação em parte genérica e específica inspirou a estrutura de nosso *framework*.

Finalmente, em [Zhai et al. 2023], é apresentado o *HOME*, um *framework* de modelagem e verificação formal de algoritmos baseados no modelo *Heard-Of*. Diferentemente de *frameworks* anteriores que se restringem a falhas benignas, o *HOME* também possibilita a modelagem de falhas bizantinas [Lamport et al. 1982]. Neste *framework*, a especificação do algoritmo e o predicado se dá através da linguagem *HOML*, que é baseada nas abstrações originais do modelo HO. A verificação é realizada através da tradução automática do sistema especificado em um problema satisfatibilidade booleana (SAT), e apresenta um desempenho superior quando comparado aos *frameworks* anteriores. Apesar de suas vantagens, o *HOME* possui algumas restrições: (1) A linguagem *HOML* é restrita a especificação de algoritmos de *threshold*; (2) O *framework* não conta com a verificação explicita da propriedade de *termination* dos algoritmos; (3) A verificação de *liveness* do sistema é restrita a um problema de alcançabilidade de estados de univalência; (4) É necessário definir os estados de univalência do algoritmo para realizar sua verificação.

Esses trabalhos fornecem uma base para o desenvolvimento do *framework* proposto neste artigo. Apesar de todos explorarem abstrações semelhantes ou complementares ao modelo *Heard-Of*, nenhum dos *framewokrs* mencionados modelam o sistema distribuído a ser verificado sem definir explicitamente os conjuntos *Heard-Of*. Ao utilizarmos esta abordagem, nosso *framework* em particular apresenta uma redução significativa no espaço de estados gerado, sem perda de expressividade na verificação das propriedades do consenso. Tais consequências serão melhor explorado na Seção 4.

4. Apresentação do Framework

Usaremos o algoritmo de consenso *Uniform Voting* como exemplo para demonstração das funcionalidades do *framework* proposto. Este exemplo é idêntico ao proposto no artigo original onde o modelo HO é apresentado [Charron-Bost and Schiper 2009] e já é apresentado usando as abstrações do modelo *Heard-Of*, ou seja, é expresso através das funções S e T. Além disso, como o uso de rodadas é inerente ao modelo *Heard-Of*, usa-se r como a rodada na apresentação do algoritmo, sendo que a definição e avanço de r faz parte do *framework* que oferece o funcionamento em rodadas descrito anteriormente.

Algoritmo 1: *Uniform Voting*

1: **Inicialização:**
2: $x_p \in V$ ▷ é o valor inicial de p
3: $v_p := Null$
4: $d_p := Null$
5: $r \in \mathbb{N}$ ▷ rodada atual

6: **Se r é impar:**
7: S_p:
8: envia $\langle x_p \rangle$ Para todos os processos em Π
9: T_p:
10: $x_p :=$ menor $\langle x_i \rangle$ recebido
11: $v_p :=$ **IF** todas as mensagens são iguais a $\langle x_i \rangle$
THEN x_i
ELSE v_p
12: $d_p := d_p$

13: **Se r é par:**
14: S_p:
15: envia $\langle x_p, v_p \rangle$ para todos os processos em Π
16: T_p:
17: $x_p :=$ **IF** pelo menos uma mensagem $\langle *, v \rangle$ com $v \neq Null$ é recebida
THEN v
ELSE menor x_i das mensagens recebidas
18: $v_p := Null$
19: $d_p :=$ **IF** todas as mensagens recebidas são iguais a $\langle *, v \rangle$ com $v \neq Null$
THEN v
ELSE d_p

Nosso exemplo considera um sistema distribuído composto por um conjunto de 3 processos $\Pi = \{p_1, p_2, p_3\}$ que executam o algoritmo *uniform voting* na primeira rodada da computação $r = 1$, Usaremos $V = \{0, 1\}$ como possíveis valores iniciais (consenso binário).

O estado inicial de cada processo nas computações do modelo HO é definido pelo algoritmo, no caso do *Uniform Voting*, as linhas 1 à 4 definem estes estados iniciais. Em nosso *framework*, usamos uma função que armazena o estado individual de todos os processos, chamada ***ProcState(p)***, onde a aplicação da função para qualquer $p \in \Pi$ retorna uma função cujo domínio é as variáveis internas do processo, que, no exemplo, serão x, v e d. Tomaremos então um estado inicial arbitrário resultando em *ProcState* sendo igual a:

```
ProcState(p) == [ p1 = [ x = 1 ; v = Null ; d = Null ] ;
                  p2 = [ x = 1 ; v = Null ; d = Null ] ;
                  p3 = [ x = 0 ; v = Null ; d = Null ] ]
```

Definido o estado inicial do sistema, nosso *framework* realiza 3 etapas de computação análogas as etapas do modelo HO, são elas:
Envio: Cada processo computa e envia as mensagens definidas pelo algoritmo de acordo com a função S. Em nosso *framework*, usamos o gerador de função $[\ X \mapsto f(x_i)\ ]$ uma ferramenta presente na linguagem do TLA+, o que gera uma função que mapeia cada elemento de um conjunto $X = \{x_1, x_2, x_3, \ldots, x_i\}$ ao resultado da aplicação da função usando o elemento como argumento $f(x_i)$. Em nosso caso, o conjunto Π (representado na linguagem TLA+) sera mapeado à mensagem resultante da aplicação da função S_p^r. Definimos este operador de *Send*:

```
Send(ProcState) == [p \in P |-> Algorithm(ProcState[p])!S]
```

O vetor obtido representa as mensagens enviadas por cada processo no round atual. Sendo assim, este operador em nosso exemplo resulta em:

```
Send(ProcState) == [ p1= [ x= 1 ] ;
                     p2= [ x= 1 ] ;
                     p3= [ x= 0 ] ]
```

Recebimento: Na segunda etapa, os processos recebem as mensagens dos elementos presentes em seu conjunto HO_p, ou um valor $Null$ se o elemento não fizer parte do conjunto. A origem dos conjuntos HO utilizados nas computações do modelo e sua computação através de um predicado de comunicação serão abordadas posteriormente neste artigo. Por enquanto, usaremos um vetor de conjuntos *Heard-Of* didaticamente escolhido para a etapa atual da computação. Sendo este:

```
HO(p) == [ p1 = { p1, p2 } ;
           p2 = { p2, p3 } ;
           p3 = { p1, p3 } ]
```

As mensagens recebidas são calculadas pelo operador*Receive*, mapeando cada processo p e q para o resultado de *Send(ProcState)[q]*, ou $Null$ caso q não seja um elemento de $HO[p]$:

```
Receive(ProcState,HO) ==
    [p \in P |->
        [q \in DOMAIN ProcState |->
            IF q \in HO[p]
            THEN Send(ProcState)[q]
            ELSE NULL                ]]
```

O resultado da aplicação deste operador resulta em uma função de transmissão, que armazena a mensagem que foi recebida por p de q, com o valor $Null$ em caso de falha de transmissão. O resultado da aplicação em nosso exemplo resulta em:

```
Receive(ProcState,HO) ==
    [ p1= [ p1= [ x= 1 ] ; p2= [ x= 1 ] ; p3=   Null    ] ;
      p2= [ p1=   Null   ; p2= [ x= 1 ] ; p3= [ x = 0 ] ] ;
      p3= [ p1= [ x= 1 ] ; p2=   Null   ; p3= [ x = 0 ] ] ]
```

Transição de estado: Na última etapa, cada processos aplica a função T_p^r usando suas variáveis internas e o vetor parcial das mensagens recebidas para calcular seu estado na próxima rodada. A função *Transition* presente em nosso *framework* realiza esta tarefa com o resultado do operador *Receive[p]* para obter apenas as mensagens recebidas pelo processo p:

```
Transition(ProcState,HO) ==
    [p \in DOMAIN Procstate |->
    Algorithm(ProcState[p])!T(Receive(ProcState,HO)[p])]
```

O resultado deste operador em nosso exemplo resulta em:

```
Transition(ProcState,HO) ==
    [ p1 = [ x = 1 ; v = 1    ; d = Null ] ;
      p2 = [ x = 0 ; v = Null ; d = Null ] ;
      p3 = [ x = 0 ; v = Null ; d = Null ] ]
```

Calculando os Conjuntos Heard-Of: Como já discutido anteriormente, os conjuntos HO são derivados de um predicado de comunicação que representa o modelo de falhas e sincronia do sistema. Revisitando o predicado de comunicação *Space Uniform* apresentado anteriormente:

$$\mathcal{P}_{sp\_unif} :: \forall r > 0, \forall p, q \in \Pi^2 : HO(p, r) = HO(q, r),$$

Este predicado pode ser traduzido diretamente para a linguagem TLA+:

```
SpUniform(P,HO) == \A p,q \in P: HO[p] = HO[q]
```

Por definição, predicados de comunicação retornam apenas valores booleanos para um conjunto HO. A coleção de todos os conjuntos HO que respeitam um predicado é definida como uma *HOcollection*. Nosso *framework* obtêm esta coleção filtrando o espaço de função $\{\Pi \rightarrow 2^{\Pi}\}$ [2], mantendo os elementos HO que respeitam tal predicado, o operador correspondente é definido na linguagem TLA+ como:

```
HOcollection(P,Predicate(_,_)) ==
    {HO \in [P -> SUBSET P]: Predicate(P,HO)}
```

Ao revisitar nosso sistema distribuído usado como exemplo anteriormente, o resultado da aplicação deste operador com o predicado *Space Uniform*, é uma coleção de 8 possíveis HO, são eles:

```
HOcollection(P,SpUniform) ==
    { [p1= {}        ; p2= {}        ; p3= {}        ],
      [p1= {p1}      ; p2= {p1}      ; p3= {p1}      ],
      [p1= {p2}      ; p2= {p2}      ; p3= {p2}      ],
      [p1= {p3}      ; p2= {p3}      ; p3= {p3}      ],
      [p1= {p1,p2}   ; p2= {p1,p2}   ; p3= {p1,p2}   ],
      [p1= {p1,p3}   ; p2= {p1,p3}   ; p3= {p1,p3}   ],
      [p1= {p2,p3}   ; p2= {p2,p3}   ; p3= {p2,p3}   ],
      [p1= {p1,p2,p3}; p2= {p1,p2,p3}; p3= {p1,p2,p3} ]}
```

Já a aplicação do operador utilizando o predicado *No Split* em nosso exemplo resultaria em um conjunto de 175 vetores HO.

Verificação das propriedades do Consenso: No problema clássico do consenso, cada processo $p \in \Pi$ de um sistema distribuído começa com um valor inicial x_p, escolhido a partir de um conjunto fixo de valores V e uma variável que corresponde a decisão do processo d_p com valor inicial $NULL$.
Para resolver o problema do consenso, os processos precisam entrar em acordo e tomar uma decisão definitiva sobre um dos valores iniciais. Onde a decisão de cada processo correto deve respeitar as propriedades de *Termination*, *Validity*, *Integrity* e *Agreement* [Van Steen 2002].
Nas computações do modelo HO, as ideias de processo correto ou processo com falhas não existe, portanto, nenhum processo está isento de tomar uma decisão. Mesmo assim, com o predicado adequado, é possível modelar sistemas com processos que apresentem falhas benignas.

[2] Sendo $\{A \rightarrow B\}$ o conjunto de todas as funções que mapeiam os elementos do conjunto A para elementos do conjunto B. Na linguagem TLA+ esta operação tem a notação `[A -> B]`

As propriedades do consenso a serem verificadas em nosso *framework*, em sua versão em linguagem natural, reescrita em termos da linguagem TLA+ e especificada diretamente no *framework*, são:
***Agreement*:** Os processos decidem o mesmo valor. Para todo processo $p, q \in \Pi$ o valor de sua variaveis d_i são iguais, ou, d_q é nulo, ou, d_q é nulo.

```
Agreement ==
  \A p,q \in P: \/ ProcState[p]["d"] = ProcState[q]["d"]
                \/ ProcState[p]["d"] = NULL
                \/ ProcState[q]["d"] = NULL
```

***Termination*:** Decorrido um tempo, todos os processos decidem. Em algum momento da execução do algoritmo, o valor de d, para todo processo $p \in \Pi$, é diferente de $NULL$.

```
Termination == <> (\A p \in P: ProcState[p]["d"] # NULL )
```

Construção do espaço de estados no *framework*: Em [Chaouch-Saad et al. 2009] apresenta dois tipos de execuções de algoritmos especificados no modelo HO: *fine-grained* e *coarse-grained*.

As execuções do tipo *fine-grained* utilizam todas as abstrações do modelo HO como variáveis de estado, obtendo assim uma especificação mais fiel ao modelo HO original. Neste tipo de execução, cada estado do sistema seria modelado usando como uma de suas variáveis de estado a rodada atual r, gerando um sistema com infinitos estados a serem explorados, impossibilitando a verificação completa das propriedades usando *model checking*.

As execuções do tipo *coarse-grained* são especificadas sem utilizar diretamente a rodada atual r como variável de estado. Para isso, a rodada atual é abstraída, utilizando a fase atual do algoritmo r para representar a causalidade dos estados do algoritmo. Para manter a notação utilizada no artigo sendo discutido, também usaremos r como a fase atual do algoritmo na especificação em TLA+. Os sistemas modelados pela execuções do tipo *coarse-grained* são finitos e, além disso, mantêm as propriedades locais que serão verificadas, tornando esse tipo de execução ideal para modelar os sistemas a serem verificados por *model checking*.

Com estas ideias em mente, nosso *framework* modela o estado do sistema a ser verificado usando apenas duas variáveis de estado: a fase atual do algoritmo r e a função $ProcState(p)$, que armazena o estado local de cada processo durante a execução do algoritmo especificado. Os conjuntos HO que serão usados na computação não fazem parte das variáveis de estado do sistema, e são definidas separadamente pela função $HOset = HOcollection(\Pi, \mathcal{P})$, onde $\mathcal{P}$ é o predicado de comunicação desejado.

Estado inicial: O estado inicial do sistema é especificado da seguinte maneira:

```
SpecInit == /\ r = 0
            /\ ProcState \in [P -> InitStates]
```

Com o algoritmo iniciando em sua fase 0 e os estados iniciais possíveis definidos pela função $InitStates$, que faz parte da especificação do algoritmo distribuído a ser verificado. Continuando com nosso exemplo, no algoritmo *Uniform Voting*, $InitStates$ é especificada:

```
InitStates == [ x : V , v : {NULL} , d : {NULL} ]
```

A notação da função em TLA+ é apenas um atalho para definir um conjunto de todas as funções com domínio $\{x, v, d\}$ onde cada variável pode ser mapeada para um elemento dos conjuntos $\{V, \{NULL\}, \{NULL\}\}$, como definimos $V = \{0, 1\}$ no inicio do exemplo, a função $InitStates$ resulta no conjunto:

```
InitStates == { [x= 0 ; v= NULL ; d= NULL] ,
                [x= 1 ; v= NULL ; d= NULL] }
```

Assim, a expressão `[P -> InitStates]` gera 8 possíveis valores para $ProcState$ resultando em 8 estados iniciais.

Próximo estado: O cálculo do próximo estado do sistema é especificado como:

```
SpecNext == /\ r' = (r + 1) % 2
            /\ ProcState' \in StateSet(ProcState,HOset)
```

Como o algoritmo que estamos usando de exemplo utiliza duas fases, a variável r alterna entre os valores 0 e 1 a cada passo da execução. O próximo estado dos processos é calculado pela função $StateSet$ especificada:

```
StateSet(ProcState,HOset) ==
    {Transition(ProcState,HO): HO \in HOset}
```

A aplicação desta função resulta em um conjunto contendo todos os possíveis próximos $ProcState$ da execução, um para cada elemento do conjunto $HOset$. Com esta abordagem, os conjuntos HO deixam de fazer parte das variáveis de estado que descrevem o sistema e são tratados como as transições que o sistema pode realizar, esta abordagem reduz significativamente o espaço de estados do modelo, principalmente quando o predicados utilizado resulta em coleções HO com muitos elementos. O sistema então é modelado usando $SpecInit$ como estado inicial, $SpecNext$ como transições e uma condição de *weak fairness* para $SpecNext$:

```
Spec == /\ SpecInit
        /\ [][SpecNext]_<<ProcState,r>>
        /\ WF_<<ProcState,r>>(SpecNext)
```

5. Verificação do algoritmo Uniform Voting utilizando o framework

Nesta seção do artigo iremos realizar a verificação formal por *model checking* do algoritmo *Uniform Voting* utilizando o *framework* apresentado até então. O objetivo deste experimento é demonstrar a efetividade do *framework* em verificar propriedades do consenso em algoritmos distribuídos tolerantes a falhas benignas especificados através das abstrações do modelo HO. Para atingir tal objetivo, iremos usar o *framework* proposto para replicar resultados apresentados no artigo onde o modelo HO é apresentado.

O resultado obtido em [Charron-Bost and Schiper 2009], usa o modelo HO, o algoritmo *Uniform Voting*, o predicado **No-Split** e o predicado **Space Uniform** resultando nas seguintes conclusões:

1. O algoritmo *Uniform Voting* garante a propriedade *Agreement* em qualquer execução onde o predicado **No-Split** é respeitado pelos conjuntos HO
2. O algoritmo *Uniform Voting* garante a propriedade *Termination* em qualquer execução onde: o predicado **No-Split** é respeitado pelos conjuntos HO e pelo menos uma passo da execução respeite o predicado **Space Uniform**

Agora iremos demonstrar os resultados obtidos usando nosso *framework* para reproduzir as situações mencionadas. Todas as medidas foram feitas em um notebook com sistema operacional Windows 11 com um processador Intel i5 11ª geração de 2.40GHz e 8.00 GB de memória RAM

Experimento 1.1: Criamos um novo modelo para ser verificado pelo TLC com a especificação: `Temporal Formula = Spec`, Model Values:`V={0,1,2}` e com as invariantes: `Agreement`.

O predicado **No-Split** é especificado em TLA+:

```
NoSplit(P,HO) == \A p,q \in P: (HO[p] \intersect HO[q]) # {}
```

Este predicado então é usado na função $HOset$:

```
HOset == HOcollection(P,NoSplit)
```

É executada a verificação do modelo pelo TLC.

Resultados 1.1: A verificação termina sem erros, o número de estados encontrados e estados distintos gerados por este experimento, em relação ao número de processos, são apresentados na tabela abaixo, e são comparados com os resultados dos experimentos obtidos em [Chaouch-Saad et al. 2009].

Tabela 1. Resultados obtidos em 1.1

	Experimento 1.1		*Chaouch-Saad*	
Processos	3	4	3	4
Estados	1350	13,749	21,351	5,865,770
Distintos	122	332	122	887

Experimento 1.2: Usando o mesmo modelo criado no Experimento 1.1, adicionamos a propriedade temporal $Termination$ à especificação, e então, é executada a verificação do modelo pelo TLC.

Resultados 1.2: A verificação termina apontando a violação de uma propriedade temporal, isso mostra que a propriedade $Termination$ não é garantida pelo algoritmo *Uniform Voting* e o predicado **No-Split**. Realizando a mesma verificação substituindo a propriedade $Termination$ por sua negação ($\neg Termination$) também resulta na violação de uma propriedade temporal, demonstrando que existem execuções em que o algoritmo termina.

Experimento 2.1: A especificação anterior é modificada para conter o predicado `SpUniform(P,HO)` (**Space Uniform**) e as seguintes definições adicionais:

```
HOuniform == HOcollection(P,SpUniform)

UniformRound == StateSet(ProcState,HOuniform)

Liveness == ProcState \in UniformRound ⇝ Termination
```

Resultados 2.1: A verificação termina sem erros e com o mesmo número de estados apresentados no experimento 1.1, embora sua verificação com três processos tenha levado 40 segundos.

6. Considerações finais

A principal contribuição deste artigo é um *framework* na linguagem TLA+ que utiliza *model checking* para verificar as propriedades do consenso de algoritmos distribuídos tolerantes a falhas benignas. Esses algoritmos são especificados no modelo *Heard-Of*. O *framework* permite a especificação de diferentes predicados, o que possibilita a verificação formal das condições que um algoritmo distribuído garante as propriedades do consenso.

O *framework* desenvolvido também apresenta uma abordagem diferente em relação a outros *frameworks* que também usam o modelo *Heard-Of*: Nossa especificação do sistema não utiliza os conjuntos HO como variável de estado. Este modelo de sistema apresenta um espaço de estados consideravelmente menor quando comparados a um modelo que usa os conjuntos HO na sua especificação. Apesar da redução, o modelo especificado precisa de um estudo mais aprofundado quanto ao tempo de verificação das propriedades do modelo.

Em trabalhos futuros, pretendemos verificar algoritmos distribuídos mais concretos como *PBFT* [Castro et al. 1999]. Além disso, planejamos expandir o *framework* para permitir a verificação de algoritmos distribuídos tolerantes a falhas bizantinas [Lamport et al. 1982] pelo uso do modelo *Altered Heard-Of* [Biely et al. 2007].

7. Agradecimentos

O presente trabalho foi realizado com apoio do CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil.

Referências

Agência Nacional de Vigilância Sanitária (ANVISA) (2020). *Guia para Validação de Sistemas Computadorizados - Guia nº 33/2020*. Agência Nacional de Vigilância Sanitária.

Barbosa, R., Fonseca, A., and Araujo, F. (2021). Reductions and abstractions for formal verification of distributed round-based algorithms. *Software Quality Journal*, 29(3):705–731.

Biely, M., Widder, J., Charron-Bost, B., Gaillard, A., Hutle, M., and Schiper, A. (2007). Tolerating corrupted communication. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 244–253.

Bögli, R., Lerena, L., Tsigkanos, C., and Kehrer, T. (2025). A systematic literature review on a decade of industrial tla+ practice. In *International Conference on Integrated Formal Methods*, pages 24–34. Springer.

Castro, M., Liskov, B., et al. (1999). Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186.

Chaouch-Saad, M., Charron-Bost, B., and Merz, S. (2009). A reduction theorem for the verification of round-based distributed algorithms. In *International Workshop on Reachability Problems*, pages 93–106. Springer.

Charron-Bost, B. and Schiper, A. (2009). The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). Nusmv 2: An opensource tool for symbolic model checking. In *Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, 2002 Proceedings 14*, pages 359–364. Springer.

Hutle, M. and Schiper, A. (2007). Communication predicates: A high-level abstraction for coping with transient and dynamic faults. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 92–101. IEEE.

Lamport, L. (2002). *Specifying systems: the TLA+ language and tools for hardware and software engineers*.

Lamport, L., Shostak, R. E., and Pease, M. C. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4:382–401.

Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., and Deardeuff, M. (2014). Use of formal methods at amazon web services. *See http://research. microsoft. com/en-us/um/people/lamport/tla/formal-methods-amazon. pdf*, page 16.

RTCA (2011). DO-178C: Software Considerations in Airborne Systems and Equipment Certification. Supersedes DO-178B (1992).

Tsuchiya, T. and Schiper, A. (2007). Model checking of consensus algorithms. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 137–148. IEEE.

Van Steen, M. (2002). Distributed systems principles and paradigms. *Network*, 2(28):1.

Zhai, S., Li, X., and Ge, N. (2023). Home: Heard-of based formal modeling and verification environment for consensus protocols. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 16–20. IEEE.