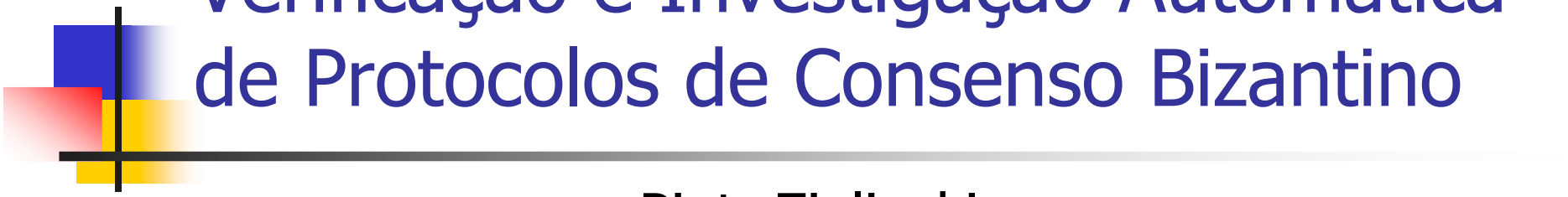
Automatic Verification and Discovery of Byzantine Consensus Protocols

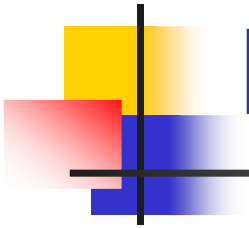
Verificação e Investigação Automática de Protocolos de Consenso Bizantino



Piotr Zielinski
Cavendish Laboratory, University of Cambridge, UK

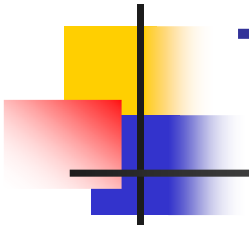
The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)
25 – 28 June 2007, Edinburg, UK

Apresentado por Jorge Tortato Júnior
Sistemas Distribuídos – Mestrado em Informática - UFPR



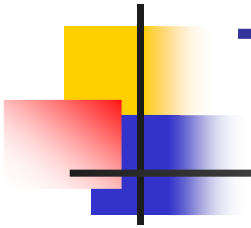
Roteiro

- Tema
- Trabalhos relacionados
- Modelos
- Verificação automática
- Investigação automática
- Resultados
- Conclusões
- Análise crítica do artigo



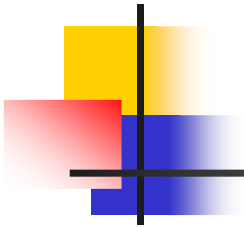
Tema

- Algoritmos distribuídos assíncronos de consenso tolerantes à falhas são difíceis de serem projetados/testados.
- Verificação automática acelera o trabalho e a verificação formal.
- Investigação automática encontra algoritmos baseados em uma especificação.
- Os dois últimos itens são abordados neste artigo.



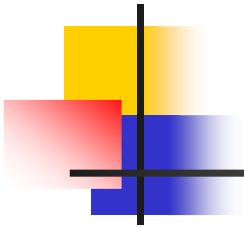
Trabalhos relacionados

- Vários algoritmos assíncronos de consenso foram propostos para modelos *crash-stop* e bizantinos.
- A verificação formal é extremamente trabalhosa, especialmente para modelos bizantinos.
- Verificação automática existentes para algoritmos distribuídos tolerantes à falhas é restritas aos modelos *crash-stop.*
- Verificação e geração automática pode ser vista em Bar-David e Taubenfeld para algoritmos distribuídos de exclusão mútua.



Modelos - Sistema

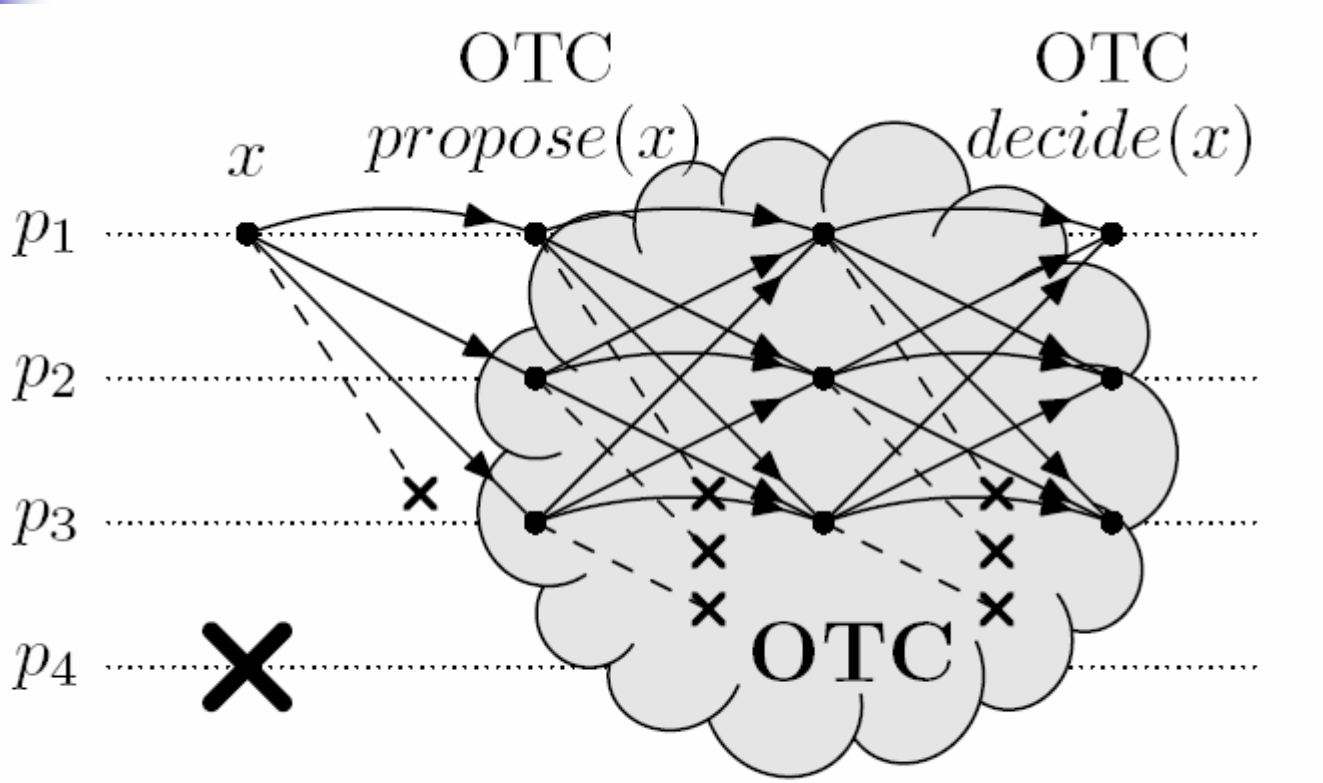
- Um modelo para ao algoritmos de consenso precisa ser definido.
  - Permitir a representação da maioria dos algoritmos conhecidos.
  - Flexível: contemplar falhas bizantinas, *crash-stop*, etc.
- Uso da abstração OTC (*Optmistically Termination Consensus)*.



Modelo - Sistema

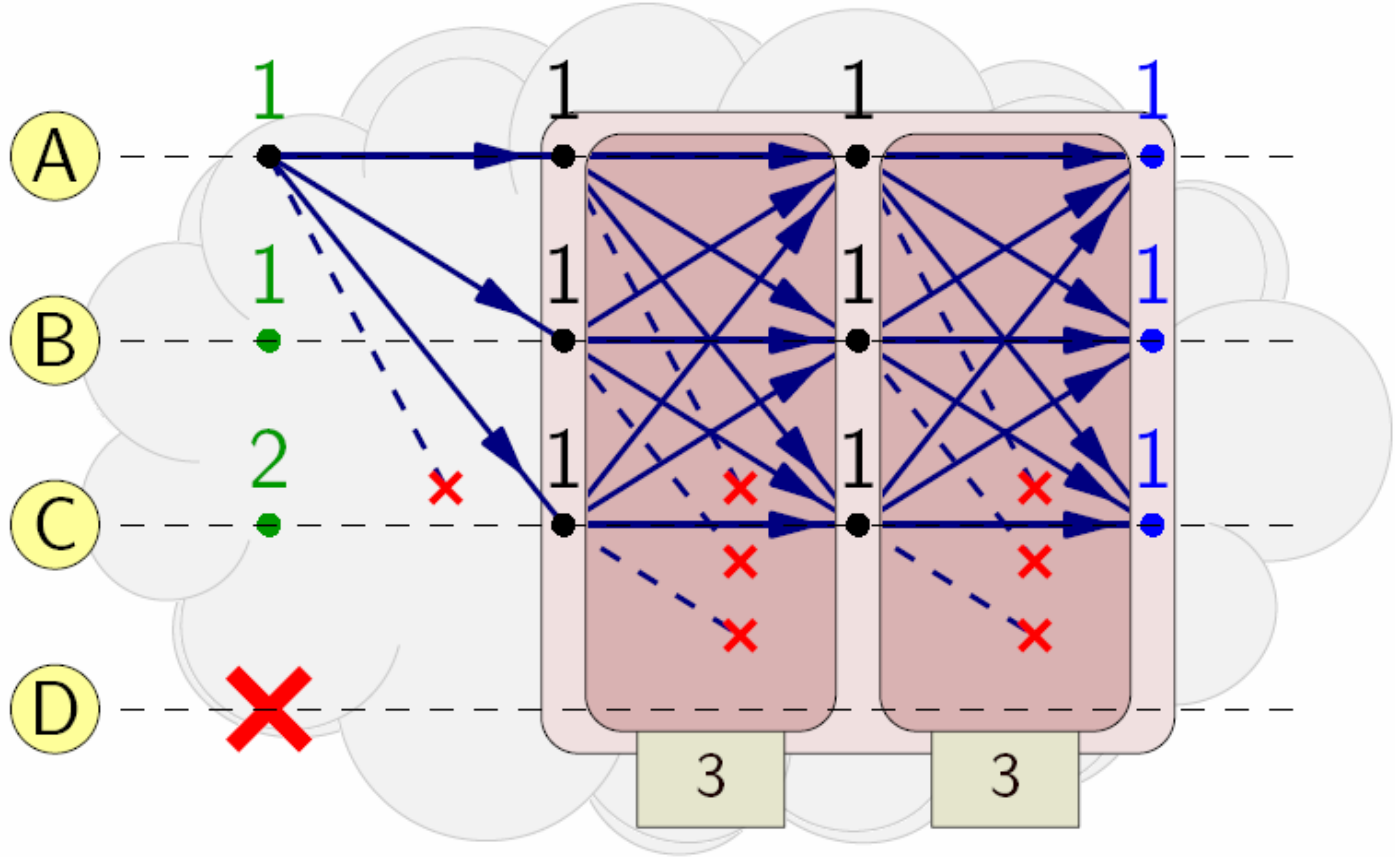
- Processos: P={p1, ..., pn}.
- Falhas: conj. F, sendo conj. M maliciosas.
- $M \subseteq F \subseteq P$
- Canais de comunicação: assíncronos e confiáveis.
- Propriedades de consenso (Bizantino):
  - Validade: algum $p_i \in P$ propõe uma solução.
  - Acordo: todos $p_i \in P$ e $p_i \notin F$ decidem igualmente.
  - Término: todos $p_i \in P$ e $p_i \notin F$ decidem.

Modelos – OTC Gráfico



A run of Byzantine Consensus

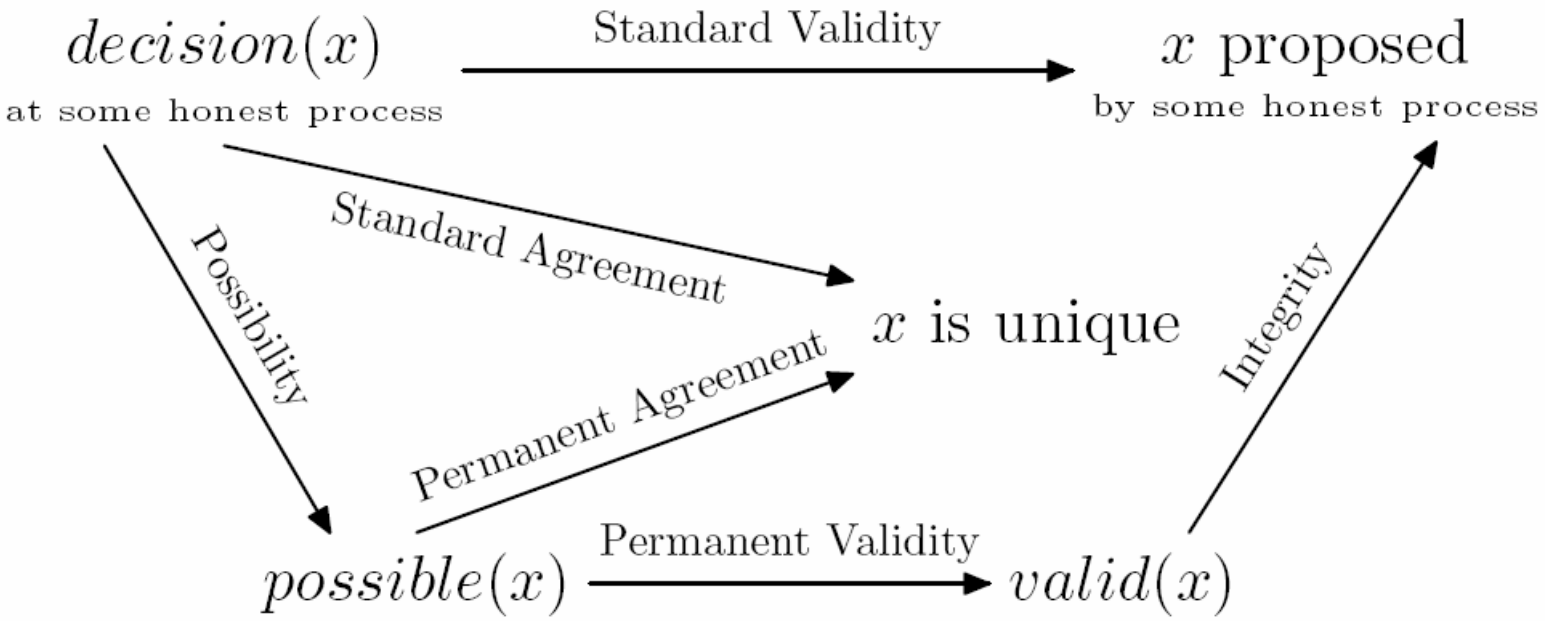
Modelos - OTC Composição



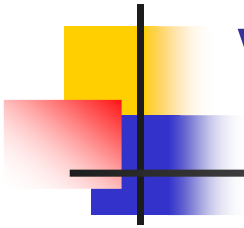
Modelos – Propriedades OTC

- Primitivas: *propose(x)*, *stop*, *decision(x)*, *valid(x)* e *possible(x)*.
- Propriedades OTC:
  - **P1. Integridade**: se v*alid(x) em* p_i, então *propose(x)* foi executado por um p_k.
  - **P2. Possibilidade**: se *decision(x)* em p_i, *possible(x)* em P.
  - **P3. Validade Permanente**: *possible(x)* => *valid(x).*
  - **P4. Acordo Permanente**: *possible(x)* é valido ao menos em um p_i.
  - **P5. Término otimista (X,C,k)**: se conjunto X propõe x, e todos C processos são corretos, *decision(x)* acontece em C após k rodadas.

Modelos - Propriedades OTC



OTC properties graphically

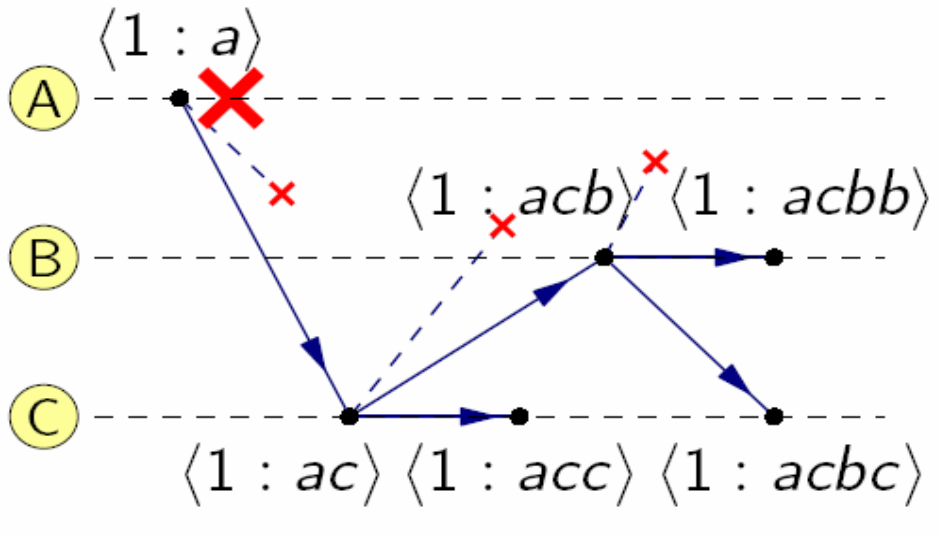


Verificação automática

- Considera apenas a latência da primeira rodada (coordenador honesto), desconsiderando a ordem das mensagens.
- Consiste em comprovar as propriedades P3, P4 e P5 a partir de P1 e P2. Elas garantem a recuperação de estado se uma falha ocorrer!
- P5 define os requisitos do protocolo. Ex.:
  - Decidir em uma rodada se não existem erros OT(P,P,1)
  - Decidir em duas rodadas se houver uma falha OT(P/{pi},P/{pi},2)
- Baseia-se em eventos e estados.

Verificação automática – Estados e eventos

- A recepção de uma mensagem é um evento.
- Adiciona o estado corrente e o id do processo (evento) ao voto e faz o *multicast.* Descarta conflitos => votos diferentes do mesmo caminho. Ex.: <1:p1>, <2:p1>.



Verificação automática – Estados e eventos

```
1   state ← ∅                                          { the empty set }
2   when a process executes propose(x) do
3     incorporate ⟨x : ε⟩ into the state               { ε: empty seq. }
4   when a process receives ⟨x : q1 ... qi−1⟩ from qi do
5     incorporate ⟨x : q1 ... qi⟩ into the state
6   function incorporate ⟨x : q1 ... qi⟩ into the state is
7     if ⟨y : q1 ... qi⟩ ∉ state for any y then
8       insert ⟨x : q1 ... qi⟩ into state
9       broadcast ⟨x : q1 ... qi⟩
10  when a process executes stop do
11    for all sequences q1 ... qi do                   { including ε }
12      incorporate ⟨⊥ : q1 ... qi⟩ into the state
```

Evolution of states

Verificação automática - Formalismo

- Consistência dos estados:
  - Conflitos => Votos ≠ do mesmo caminho

$$conflict(S) \stackrel{\text{def}}{=} \{\, \alpha \mid \exists\, x \neq y : \langle x : \alpha \rangle \in S \wedge \langle y : \alpha \rangle \in S \,\}$$

$$conflict(\{\langle 1 : p_1 p_2 \rangle, \langle 2 : p_2 \rangle, \langle 2 : p_1 p_2 \rangle\}) = \{p_1 p_2\}$$

  - Inferências => Cjto. eventos possíveis

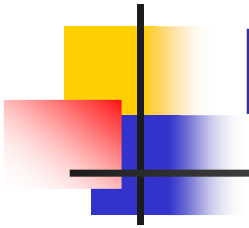
$$prefs(q_1 \ldots q_i, M) \stackrel{\text{def}}{=} \{\, q_1 \ldots q_j \mid q_{j+1}, \ldots, q_i \notin M \,\}$$

$$infer(S, M) = \langle x : prefs(q_1 \ldots q_i, M) \rangle$$

  - Consistência => conflitos devidos à processos maliciosos

$$conflict(infer(S, M)) \subseteq \alpha M$$

Verificação automática - Formalismo



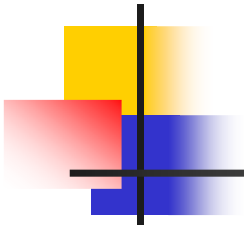
- Acordo Permanente:
  - A1: consistência de estados
  - A2: Estado S é completo: processo recebe todos os eventos produzidos por todos os processos corretos
  - A3: para todo $z \in \{x,y\}$, eventos de decisão $<z,Dz>$ são consistentes com S, ou seja, *x* e *y* podem ser decisões, o que invalida o algoritmo sendo analisado.

Verificação automática - Formalismo

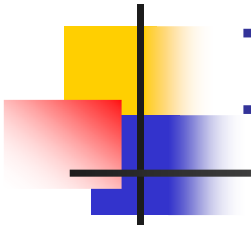
```
1  function PermanentAgreement(OTs) is
2    for all D_x, D_y corresponding to OTs (3) do
3      for all F ∈ F and M, M_x, M_y ∈ M do
4        if M ⊆ F then
5          compute the least fixpoints S_x and S_y of (7)
6          if computed S_x and S_y satisfy (8) then
7            return FALSE
8    return TRUE
```

Testing Permanent Agreement

Verificação automática - Desempenho



- Considera apenas a primeira rodada (otimista).
- Verifica apenas os estados mínimos dependendo do número *k* de passos necessários.
- Procura por condições inválidas.
- Utiliza o algoritmo de Tarski para diminuir a ordem do algoritmo: de $O(3^{n^k})$ para $O(n^k)$.



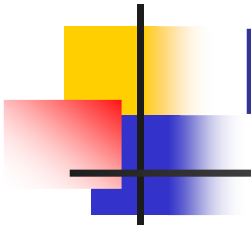
Investigação automática

- O desempenho da verificação permite que novos algoritmos sejam procurados automaticamente.
- Para todas as combinações os conjuntos possíveis de P, F e M.
- Recursivamente adiciona condições OT.
- Ordena as condições OT para evitar que o mesmo conjunto de condições seja analisado mais de uma vez.
- Analisa se uma condição OT é dominante, ou seja, analisa apenas as condições mais restritivas.

Investigação automática

```
1  function OTCSearch(𝒯) is
2    if PermValidity(𝒯) and PermAgreement(𝒯) then
3      output 𝒯
4      for all possible OT conditions T = (X, C, k) do
5        if T is greater (">OT") than all elements of 𝒯 and
6          T does not dominate any element of 𝒯 and
7          T is not dominated by any element of 𝒯 then
8            OTCSearch(𝒯 ∪ {T})
```

Discovering new OTC protocols



Resultados

n	failures	algs tested	found	time
3	1 crash-stop	360	1	0.03 sec
4	1 crash-stop	8,512	2	0.33 sec
5	1 crash-stop	341,312	3	0.83 sec
5	2 crash-stop	32,620,109	6	61.52 sec
4	1 malicious	47,990	7	0.41 sec
5	1 malicious	11.9 billion	6	39.4 hours

Resultados

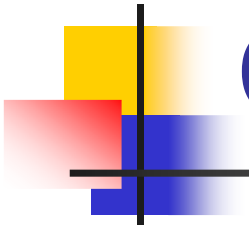
- Ex.: 3 processos e uma falha.

$$\mathcal{F} = \{\emptyset, \{p_1\}, \{p_2\}, \{p_3\}\}, \quad \mathcal{M} = \{\emptyset\}$$

- Condições OT

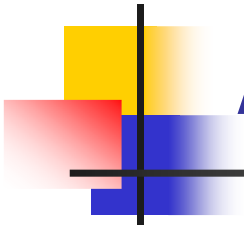
$$\left\{ \begin{array}{c} \langle \{p_1, p_2\}, \{p_1, p_2\}, 1 \rangle \\ \langle \{p_1\}, \{p_1, p_2\}, 2 \rangle \\ \langle \{p_1\}, \{p_1, p_3\}, 2 \rangle \end{array} \right\} \qquad \left\{ \begin{array}{c} \langle \{p_1, p_2\}, \{p_1, p_2\}, 1 \rangle \\ \langle \{p_1, p_3\}, \{p_1, p_3\}, 2 \rangle \\ \langle \{p_2, p_3\}, \{p_2, p_3\}, 2 \rangle \end{array} \right\}$$

- Traçado



Conclusões

- Foi apresentado um método para verificação e investigação de protocolos de consenso.
- O tempo de processamento e o tamanho dos dados necessários foi reduzido através de otimizações e do uso de um modelo, o OTC.
- Permite a análise e descoberta de erros em protocolos de forma rápida e automatizada.
- Protocolos descobertos automaticamente podem ser usados para o desenvolvimento de protocolos mais genéricos.
- Protocolos tolerantes a falhas parecem ser uma área de pesquisa promissora para a geração automática de código.



Análise crítica do artigo

- A investigação/verificação automática de protocolos parece ser uma área promissora de pesquisas.
- Apesar dos bons resultados muito trabalho ainda precisa ser desenvolvido.
- Apesar das otimizações, o tempo de investigação para um número maior de processos e com um número maior de falhas tende a crescer muito.
- Generalizações para automação de protocolos com um número genérico de processos ainda não foi atingido.