

Por que e pra que métodos formais?

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

13 de março de 2024



Conteúdo

Joost-Pieter Katoen

A Casa Branca

Leslie Lamport

Hillel Wayne

Introdução

Em todas as aulas dessa disciplina, eu vou estar, explicita ou implicitamente, tentando convencer vocês de que usar métodos formais para verificar sistemas complexos é importante.

Contudo, pessoas diferentes tem visões diferentes, e tem muita gente de respeito por aí falando da importância de métodos formais. Essa aula vai apresentar as respostas de **outras pessoas** à pergunta: **Por que e pra que métodos formais?**



Outline

Joost-Pieter Katoen

A Casa Branca

Leslie Lamport

Hillel Wayne

Joost-Pieter Katoen




- Cientista teórico da computação
- Professor renomado e líder do Grupo de Modelagem e Verificação de software na universidade RWTH Aachen University.
- Um dos autores do livro “Principles of Model Checking”, da bibliografia do curso

Impacto

O impacto que um *bug* pode ter varia muito de acordo com o tipo de sistema.

Vamos ver alguns exemplos de *bugs* que tiveram um impacto muito grande. Esses exemplos foram dados pelo professor Joost-Pieter Katoen em uma aula introdutória a *model checking* (KATOEN, 2013).

Overdose de radiação na Therac-25 (1985-87)

PATIENT NAME: John	BEAM TYPE: E	ENERGY (KeV):	10
TREATMENT MODE: FIX			
	ACTUAL	PRESCRIBED	
UNIT RATE/MINUTE	0.000000	0.000000	
MONITOR UNITS	200.000000	200.000000	
TIME (MIN)	0.270000	0.270000	
SANTRY ROTATION (DEG)	0.000000	0.000000	VERIFIED
COLLIMATOR ROTATION (DEG)	359.200000	359.200000	VERIFIED
COLLIMATOR X (CM)	14.200000	14.200000	VERIFIED
COLLIMATOR Y (CM)	27.200000	27.200000	VERIFIED
WEDGE NUMBER	1.000000	1.000000	VERIFIED
ACCESSORY NUMBER	0.000000	0.000000	VERIFIED
DATE: 2012-04-16	SYSTEM: BEAM READY	OP.MODE: TREAT	AUTO
TIME: 11:48:30	TREAT: TREAT PAUSE	X-RAY	175777
OP# ID: 633-tf3p	REASON: OPERATOR	COMMAND: 	

- Therac-25 era uma máquina de radioterapia controlada por computador
- 6 acidentes por overdose de radiação (~100x da dose)
 - 3 pacientes morreram
- Causa: software de controle entre dois modos de tratamento

Colapso da rede de telefonia da AT&T (1990)

- 9 horas de serviço interrompido em grande parte dos EUA.
- Causa: um `break` incorreto em `C` (deveria sair de um `if`, mas estava saindo de um `switch`), que fez os computadores reiniciarem em cadeia a cada 6s.
- Prejuízo: Centenas de milhões de dólares

Acidente do míssil Ariane 5 (1996)

BBC NEWS BRASIL

Notícias Brasil Internacional Economia Saúde Ciência Tecnologia Vídeos

As falhas numéricas que podem causar desastres

Chris Baraniuk
Da BBC Future

14 maio 2015

4 de junho de 1996 será para sempre lembrado como um dia sombrio para a Agência Espacial Europeia. O primeiro voo não tripulado do foguete Ariane 5, que decolou carregando quatro satélites científicos caríssimos, acabou 39 segundos depois do lançamento, em uma horrível bola de fogo e fumaça. Estima-se que a explosão causou um prejuízo de US\$ 370 milhões.

Não foi uma falha mecânica nem um ato de sabotagem. O desastre foi causado por um simples *bug* em um software, que fez cálculos errados ao se tornar sobrecarregado com números mais longos do que era capaz de suportar.

- Conversão de um float 64 bits pra um inteiro de 16 bits

Defeito de ponto flutuante (1994)



- Em inglês, *Pentium FDIV bug*
- Lançamento da família *Pentium* da Intel
- Uma das inovações era uma tabela para aumentar a velocidade de multiplicação de ponto flutuante
 - Ainda na fase de testes, descobriram um problema, mas acreditava-se ser muito raro
 - No fim, não era tão raro assim, e tiveram que fazer *recall* dos produtos
- Prejuízo: ~500 milhões de dólares e grande perda de reputação da Intel



Outline

Joost-Pieter Katoen

A Casa Branca

Leslie Lamport

Hillel Wayne

Documento da Casa Branca (HOUSE, 2024)

BACK TO THE BUILDING BLOCKS:

A PATH TOWARD SECURE AND MEASURABLE SOFTWARE

FEBRUARY 2024



Parágrafo 1

Even if engineers build with memory safe programming languages and memory safe chips, one must think about the vulnerabilities that will persist even after technology manufacturers take steps to eliminate the most prevalent classes. Given the complexities of code, **testing is a necessary but insufficient step** in the development process to fully reduce vulnerabilities at scale. If correctness is defined as the ability of a piece of software to meet a specific security requirement, then it is possible to **demonstrate correctness using mathematical techniques** called formal methods. These techniques, often used to prove a range of software outcomes, can also be used in a cybersecurity context and are **viable even in complex environments like space**. While formal methods have been studied for decades, their deployment remains limited; **further innovation in approaches to make formal methods widely accessible is vital to accelerate broad adoption**. Doing so enables formal methods to serve as another powerful tool to give software developers greater assurance that entire classes of vulnerabilities, even beyond memory safety bugs, are absent.

Parágrafo 2

While there are several types of formal methods that span a range of techniques and stages in the software development process, this report highlights a few specific examples. **Sound static analysis** examines the software for specific properties without executing the code. This method is effective because it can be used across many representations of software, including the source code, architecture, requirements, and executables. **Model checkers** can answer questions about a number of higher-level properties. These algorithms can be used during production; however, they are limited in their scaled use due to their computational complexity. **Assertion-based testing** is a formal statement of properties carried in the code that may be used to cross-check the code during testing or production. These generated proofs allow for **faults to be detected much earlier** and closer to the erroneous code, rather than tracing back from externally visible systems failures.

Parágrafo 3

There are two ways software engineers can use these techniques across software and hardware. First, formal methods can be incorporated directly into **the developer toolchain**. As the programmer builds, tests, and deploys software, the compiler can automate these mathematical proofs and verify that a security condition is met. Additionally, the developer can **use formally verified core components** in their software supply chain. By choosing provably secure software libraries, developers can ensure the components they are using are less likely to contain vulnerabilities.

Parágrafo 4

Formal methods can be incorporated throughout the development process to reduce the prevalence of multiple categories of vulnerabilities. Some emerging technologies are also well-suited to this technique. As questions arise about the safety or trustworthiness of a new software product, **formal methods can accelerate market adoption in ways that traditional software testing methods cannot.** They allow for proving the presence of an **affirmative requirement, rather than testing for the absence of a negative condition.**

Parágrafo 5

While memory safe hardware and formal methods can be **excellent complementary approaches** to mitigating undiscovered vulnerabilities, one of the most impactful actions software and hardware manufacturers can take is **adopting memory safe programming languages**. They offer a way to eliminate, not just mitigate, entire bug classes. This is a remarkable opportunity for the technical community to improve the cybersecurity of the entire digital ecosystem.



Outline

Joost-Pieter Katoen

A Casa Branca

Leslie Lamport

Hillel Wayne

Leslie Lamport



- Muitos trabalhos importantes na área de Sistemas Distribuídos
- Autor inicial do \LaTeX
- Recebeu um prêmio de Turing em 2013
- Criador de TLA e TLA+



Quem constrói casas sem antes desenhar plantas? (LAMPORT, 2015)

Writing is nature's way of letting you know how sloppy your thinking is. –Dick Guindon

Quem constrói casas sem antes desenhar plantas? (LAMPORT, 2015)

Writing is nature's way of letting you know how sloppy your thinking is. –Dick Guindon

Plantas nos permitem pensar com clareza sobre o que estamos construindo. Antes de escrever código, nós deveríamos escrever uma “planta” - em software, uma especificação.

Usando a comparação com plantas

- Muitas pessoas argumentam que especificar software é uma perda de tempo
 - Por exemplo: especificações são inúteis porque não podemos gerar todo o **código** a partir dela
 - Isso é como dizer que desenhar plantas é inútil porque ainda precisa-se de **construtores** para construir.

Usando a comparação com plantas

- Muitas pessoas argumentam que especificar software é uma perda de tempo
 - Por exemplo: especificações são inúteis porque não podemos gerar todo o **código** a partir dela
 - Isso é como dizer que desenhar plantas é inútil porque ainda precisa-se de **construtores** para construir.
- Alguns argumentam que essa analogia não é válida porque é mais fácil **mudar código** do que **mudar uma construção**
 - **Não!** Pode ser muito difícil mudar código, principalmente sem introduzir bugs. Especialmente sem especificações.



Escrevendo especificações

*But few engineers write specs because they have little **time to learn how** on the job, and they are unlikely to have learned in school. Some graduate schools teach courses on specification languages, but few teach how to use specification **in practice**. It's hard to draw blueprints for a skyscraper without ever having drawn one for a toolshed.*



Escrevendo especificações

*But few engineers write specs because they have little **time to learn how** on the job, and they are unlikely to have learned in school. Some graduate schools teach courses on specification languages, but few teach how to use specification **in practice**. It's hard to draw blueprints for a skyscraper without ever having drawn one for a toolshed.*

Uma dica é evitar usar o código como base para escrever especificações. Arquitetos não fazem as plantas usando tijolos.



Outline

Joost-Pieter Katoen

A Casa Branca

Leslie Lamport

Hillel Wayne

Hillel Wayne



- Autor do site Learn TLA+, do livro Practical TLA+ and da documentação do Alloy (linguagem de especificação)
- Trabalha como consultor pra empresas, ensinando times a usarem TLA+
 - Muita experiência prática

Hillel's Pitch (WAYNE, 2019)

*Formal methods are an incredibly powerful tool. The biggest barrier to using them, in my opinion, is **education**. FM requires a different mindset from coding and sometimes people have trouble **building the intuition**. There's also an implicitly-assumed set of math skills that are easy to learn but hard to realize you need to learn.*



Achando bugs rapidamente (WAYNE, 2020)

Quanto antes os bugs são encontrados, menos dano eles causam.

Achando bugs rapidamente (WAYNE, 2020)

Quanto antes os bugs são encontrados, menos dano eles causam.

- AWS (CHRIS NEWCOMBE, 2014)
 - Modelaram DynamoDB e S3 (entre outros)
 - Encontraram bugs complexos em ambos, um deles requeria 35 passos para reprodução
 - Testes, QA e revisão de código não foram suficientes
 - Também conseguiram fazer otimizações agressivas com mais confiança



Achando bugs rapidamente (WAYNE, 2020)

Quanto antes os bugs são encontrados, menos dano eles causam.

- AWS (CHRIS NEWCOMBE, 2014)
 - Modelaram DynamoDB e S3 (entre outros)
 - Encontraram bugs complexos em ambos, um deles requeria 35 passos para reprodução
 - Testes, QA e revisão de código não foram suficientes
 - Também conseguiram fazer otimizações agressivas com mais confiança
- eSparkLearning (WAYNE, 2017)
 - Precisaram modificar o sistema para atender um cliente grande
 - Dois dias investidos em uma especificação em TLA+
 - Bugs significantes encontrados, que causariam a perda desse cliente
 - Estimativa de \$300k/ano economizados

Economizando no desenvolvimento e manutenção

Formalizando designs, é possível simplificar os sistemas antes de começar a desenvolver.

Economizando no desenvolvimento e manutenção

Formalizando designs, é possível simplificar os sistemas antes de começar a desenvolver.

- OpenComRTOS (VERHULST et al., 2011)
 - Real-time operating system (RTOS)
 - Modelar ajudou com que desenvolvedores júniores (menos experientes) pudessem contribuir para o sistema complexo

Economizando no desenvolvimento e manutenção

Formalizando designs, é possível simplificar os sistemas antes de começar a desenvolver.

- OpenComRTOS (VERHULST et al., 2011)
 - Real-time operating system (RTOS)
 - Modelar ajudou com que desenvolvedores júniores (menos experientes) pudessem contribuir para o sistema complexo
- Cockroach Labs (VANBENSCHOTEN, 2019)
 - Modelo em TLA+ para uma otimização de commits paralelos
 - Encontraram um bug que precisaria de mais de 10 horas para debugar
 - O modelo deu confiança de que a solução para o bug funcionava

Achando bugs em sistemas legado

Mesmo em sistemas legados, é muito útil achar bugs antes que os usuários os achem.



Achando bugs em sistemas legado

Mesmo em sistemas legados, é muito útil achar bugs antes que os usuários os achem.

- Rackspace (PARLAR, 2019)
 - Analisando um sistema em produção com Alloy, acharam um bug tão severo que tiveram que refazer **um ano** de trabalho. Se tivessem usado métodos formais desde o início, poderiam ter salvado esse ano.

Achando bugs em sistemas legado

Mesmo em sistemas legados, é muito útil achar bugs antes que os usuários os achem.

- Rackspace (PARLAR, 2019)
 - Analisando um sistema em produção com Alloy, acharam um bug tão severo que tiveram que refazer **um ano** de trabalho. Se tivessem usado métodos formais desde o início, poderiam ter salvado esse ano.
- Elasticsearch (HOWELL, 2018)
 - Em três dias de modelagem de algumas partes da engine do Elasticsearch, um problema significante foi encontrado.
 - Três meses depois de arrumarem o problema, alguém encontrou e reportou o exato problema em uma versão antiga.



Demo (Adaptada)

- Estou doando meu sofá, alguém quer?
- Quem quiser, me manda um e-mail essa semana e é seu

Demo (Adaptada)

- Estou doando meu sofá, alguém quer?
- Quem quiser, me manda um e-mail essa semana e é seu

Mais detalhes:

- Um sofá tem um dono
- O dono do sofá pode ofertá-lo a outra pessoa. Quem recebe a oferta pode aceitá-la, e assim o sofá passa a ser dessa pessoa, ou rejeitá-la, e assim nada acontece
- A aceitação/rejeição é assíncrona. O dono do sofá pode oferecê-lo para várias pessoas e a pessoa pode esperar alguns dias antes de aceitar ou rejeitar a oferta

Demo (Adaptada)

- Estou doando meu sofá, alguém quer?
- Quem quiser, me manda um e-mail essa semana e é seu

Mais detalhes:

- Um sofá tem um dono
- O dono do sofá pode ofertá-lo a outra pessoa. Quem recebe a oferta pode aceitá-la, e assim o sofá passa a ser dessa pessoa, ou rejeitá-la, e assim nada acontece
- A aceitação/rejeição é assíncrona. O dono do sofá pode oferecê-lo para várias pessoas e a pessoa pode esperar alguns dias antes de aceitar ou rejeitar a oferta

Temos um problema aqui, conseguem ver?

Demo - bug

- 1 Gabriela tem o sofá
- 2 Gabriela oferece o sofá pra Alice
- 3 Gabriela oferece o sofá pro Bob
- 4 Alice aceita. O sofá agora é da Alice
- 5 Bob aceita. O sofá agora é do Bob

A Alice não ofereceu o sofá para o Bob, mas o sofá era dela e passou a ser do Bob.

Demo - características do bug

- É complexo: envolve três pessoas e quatro passos. Difícil de especificar com testes unitários.
- É sutil: O único sintoma é que a Alice ficou sem sofá. Uma pessoa testando o OLX não ia perceber isso.
- É perigoso: Viola um requisito principal do sistema. A Alice (e quem ouvir a história dela) vai deixar de confiar em mim.

Demo - características do bug

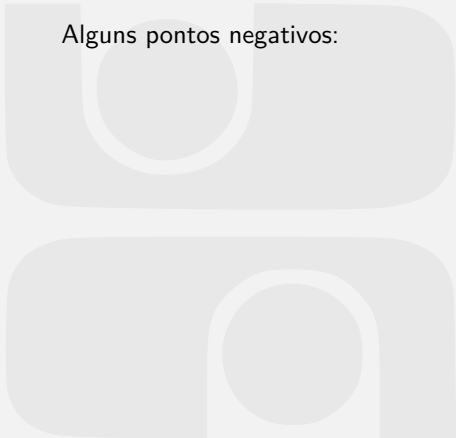
- É complexo: envolve três pessoas e quatro passos. Difícil de especificar com testes unitários.
- É sutil: O único sintoma é que a Alice ficou sem sofá. Uma pessoa testando o OLX não ia perceber isso.
- É perigoso: Viola um requisito principal do sistema. A Alice (e quem ouvir a história dela) vai deixar de confiar em mim.

Em (WAYNE, 2020), o Hillel apresenta uma especificação em TLA+ de 15 linhas para um sistema análogo a esse e encontra o bug usando um model checker.



Quando não usar métodos formais

Alguns pontos negativos:





Quando não usar métodos formais

Alguns pontos negativos:

- Não é possível obter a **implementação** a partir da especificação. Se você precisa ter absoluta certeza que a implementação corresponde à implementação, muito mais recursos são necessários.

Quando não usar métodos formais

Alguns pontos negativos:

- Não é possível obter a **implementação** a partir da especificação. Se você precisa ter absoluta certeza que a implementação corresponde à implementação, muito mais recursos são necessários.
- Escrever especificações é útil para sistemas complexos. Se você consegue manter o sistema inteiro na sua cabeça, pode não ser benéfico escrever uma especificação.
 - “Rule of thumb”: Especificar coisas que levam menos de uma semana pra desenvolver não vale a pena.

Quando não usar métodos formais

Alguns pontos negativos:

- Não é possível obter a **implementação** a partir da especificação. Se você precisa ter absoluta certeza que a implementação corresponde à implementação, muito mais recursos são necessários.
- Escrever especificações é útil para sistemas complexos. Se você consegue manter o sistema inteiro na sua cabeça, pode não ser benéfico escrever uma especificação.
 - “Rule of thumb”: Especificar coisas que levam menos de uma semana pra desenvolver não vale a pena.
- Especificações não são a melhor ferramenta pra encontrar erros simples de implementação, como null-checks.

Referências I

CHRIS NEWCOMBE, F. Z. Tim Rath. **Use of formal methods at amazon web services**. Disponível em: <<https://lamport.azurewebsites.net/tla/formal-methods-amazon.pdf>>.

HOUSE, T. W. **Back to the building blocks: A path toward secure and measurable software**. Disponível em:
<<https://www.whitehouse.gov/wp-content/uploads/2024/02/Final-ONCD-Technical-Report.pdf>>.

HOWELL, K. **Possible to index duplicate documents with same id and routing id**. Disponível em: <<https://github.com/elastic/elasticsearch/issues/31976#ISSUECOMMENT-404722753>>.

KATOEN, J.-P. **Introduction to model checking**. Disponível em:
<<https://moves.rwth-aachen.de/wp-content/uploads/SS16/mc/lec1.pdf>>.

Referências II

LAMPORT, L. Who builds a house without drawing blueprints?

Commun. acm, v. 58, n. 4, p. 38–41, Mar. 2015.

PARLAR, J. **Finding bugs without running or even looking at code.**

Disponível em: <https://www.youtube.com/watch?v=FvNR1E4E9QQ&ab_channel=StrangeLoopConference>.

VANBENSCHOTEN, N. **Parallel commits: An atomic commit protocol for globally distributed transactions.** Disponível em:

<<https://www.cockroachlabs.com/blog/parallel-commits/>>.

VERHULST, E. et al. **Formal development of a network-centric rtos.** [s.l: s.n.].

WAYNE, H. **Formal methods in practice: Using tla+ at espark learning.** Disponível em:



Referências III

`<https://medium.com/espark-engineering-blog/
formal-methods-in-practice-8f20d72bce4f>.`

WAYNE, H. **Using formal methods at work**. Disponível em:

`<https://www.hillelwayne.com/post/using-formal-methods/>.`

WAYNE, H. **The business case for formal methods**. Disponível em:

`<https://www.hillelwayne.com/post/business-case-formal-methods/>.`

Por que e pra que métodos formais?

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

13 de março de 2024