Métodos Formais
2023.2

Informações Gerais Sobre o Curso
Introdução a Métodos Formais

Área de Teoria DCC/UFMG

O professor

- **Haniel Barbosa**

  hbarbosa@dcc.ufmg.br `https://hanielbarbosa.com/`

- Formação:

2017:	Doutorado em Ciência da Computação (Université de Lorraine, França)
2012:	Mestrado em Ciência da Computação (UFRN)
2010:	Bacharelado em Ciência da Computação (UFRN)

- Experiência profissional:

2019-...:	Professor adjunto (UFMG)
2017-2019:	Professor assistente visitante (University of Iowa, EUA)
2017-2019:	Pesquisador pós-doutor (University of Iowa, EUA)
2013:	Professor substituto (UFRN)
2012:	Estágio (Clearsy, França)
2010:	Estágio (AeS - Acesso e Segurança, Brasil)

- Interesses de pesquisa:
  - automatização de raciocínio lógico,
  - satisfatibilidade módulo teorias,
  - verificação formal,
  - assistentes de demonstração

Bibliografia

- A disciplina não possui um livro-texto.

- Materiais de leitura, entre notas de aula, tutoriais, capítulos de livros e artigos, serão passados durante o semestre e serão disponibilizados na página da disciplina:

  `https://hanielbarbosa.com/teaching/ufmg/2023-2/fm`

Métodos de avaliação

- **Provas**: 40% da nota final.
  - Cada prova 20% da nota
- **Projetos**: 60% da nota final
  - Um mini-projeto em Alloy (15%)
  - Um mini-projeto em Dafny (15%)
  - Um projeto em Alloy que vocês devem propor (30%)

Comunicação

- Para material didático, informações gerais, e calendários, acesse o site da disciplina.

  e também o Moodle da disciplina:

  `https://virtual.ufmg.br/20232/course/view.php?id=11656`

  - Grupos de discussões e avisos urgentes (como eventuais cancelamentos de aula de última hora) também ocorrem no Moodle da disciplina.

- E-mails sobre a disciplina devem iniciar o campo "assunto" / "*subject*" com o indicativo **[FM]** para facilitar a organização das mensagens.

Objetivos do curso

1. Aprender sobre métodos formais
2. Entender como métodos formais podem auxiliar na construção de sistemas de alta qualidade
3. Aprender sobre modelagem formal e linguagens de especificação
4. Escrever e entender especificações formais de requisitos
5. Aprender as principais abordagens em verificação formal de sistemas
6. Saber que métodos formais usar em que cenários
7. Utilizar ferramentas automáticas e interativas para verificar especificações e código

Organização do curso

- Curso organizado por níveis de especificação
- Ênfase em escrita e validação de especificações formais através de ferramentas
- Vários exercícios de fixação
- Listas de exercício práticas em que vocês definem um sistema, o especificam e verificam esta especificação
- Projetos sobre os temas vistos

Tópicos do curso

- **Software Specification**
  - High-level design
  - Code-level design
- **Constraint Solving for Formal Verification**
  - Brief overview of SAT, SMT, and encodings
- **Software Validation Techniques**
  - Model Finding/Checking:
    often automatic, abstract
  - Deductive Verification:
    typically semi-automatic, precise

Part I: High-level Design

Language: Alloy

- Lightweight modeling language for software design
- Amenable to a fully automatic analysis
- Aimed at expressing complex structural constraints and behavior in a software system
- Intuitive structural modeling tool based on first-order logic
- Automatic analyzer based on SAT solving technology

Learning Outcomes

- Design and model software systems in the Alloy language
- Check models and their properties with the Alloy analyzer
- Understand what can and cannot be expressed in Alloy

Part II: Constraint Solving for Formal Verfification

Methods: overview of SAT, SMT and encodings

- Brief overview of these problems
- How state-of-the-art tools solve them
- How Alloy verification conditions are encoded into these problems

Learning Outcomes

- Understanding of what is behind verification tools
- Notions of how to encode problems into SAT and SMT

Part III: Code-level Specification

Language: Dafny

- Programming language with specification constructs
- Specifications embedded in source code as formal contracts
- Tool support with sophisticated verification engines
- Automatic analysis based on SMT solving technology

Learning Outcomes:

- Write formal specifications and contracts in Dafny
- Verify functional properties of Dafny programs with automated tools
- Understand what can and cannot be expressed in Dafny

Formal Methods

Formal Methods

Rigorous techniques and tools for the development and analysis of computational (hardware/software) systems

- Applied at various stages of the development cycle
- Also used in reverse engineering to model and analyze existing systems
- Based on mathematics and symbolic logic (formal)

Main Artifacts in Formal Methods

1. System requirements
2. System implementation

Formal methods rely on

a. some formal specification of (1)
b. some formal execution model of (2)

Use tools to verify mechanically that implementation satisfies (a) according to (b)

Why Use Formal Methods

- Mathematical modeling and analysis contribute to the overall quality of the final product
- Increase confidence in the correctness/robustness/security of a system
- Find more flaws and earlier (i.e., during specification and design vs. testing and maintenance)

Formal Methods: The Vision

- Complement other analysis and design methods
- Help find bugs in code and specification
- Reduce development, and testing, cost
- Ensure certain properties of the formal system model
- Should be highly automated

A Warning

- The notion of "formality" is often misunderstood (formal vs. rigorous)
- The effectiveness of formal methods is still debated
- There are persistent myths about their practicality and cost
- Formal methods are not yet widespread in industry
- They are mostly used in the development of safety, business, or mission critical software, where the cost of faults is high
- Nevertheless, as automation rises, they are increasingly used in industry

The Main Point of Formal Methods is Not

- To show “correctness” of entire systems
  - What is correctness? Go for specific properties!
- To replace testing entirely
  - Formal methods do not go below byte code level
  - Some properties are not formalizable
- To replace good design practices

There is no silver bullet!

No correct system w/o clear requirements & good design

Overall Benefits of Using Formal Methods

- Forces developers to think systematically about issues
- Improves the quality of specifications, even without formal verification
- Leads to better design
- Provides a precise reference to check requirements against
- Provides documentation within a team of developers
- Gives direction to latter development phases
- Provides a basis for reuse via specification matching
- Can replace (infinitely) many test cases
    - Formal representations can facilitates automatic test case generation

Specifications: What the system should do

- Simple properties
  - Safety properties: something bad will never happen
  - Liveness properties: something good will happen eventually
  - Non-functional properties: runtime, memory, usability, ...
- "Complete" behaviour specification
  - Equivalence check
  - Refinement
  - ...

Formal Specification

The expression in some formal language and at some level of abstraction of a collection of properties that some system should satisfy [van Lamsweerde]

- formal language:
  - syntax can be mechanically processed and checked
  - semantics is defined unambiguously by mathematical means
- abstraction:
  - above the level of source code
  - several levels possible

Formal Specification

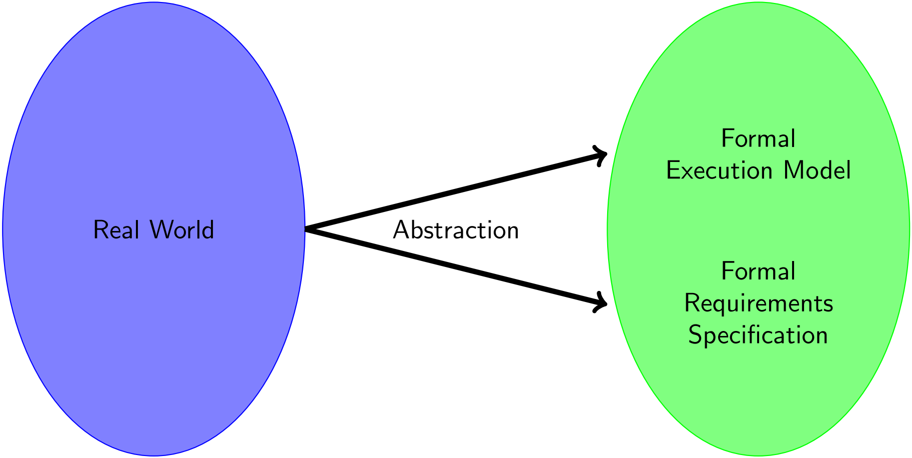
The expression in some formal language and at some level of abstraction of a collection of properties that some system should satisfy [van Lamsweerde]

- properties:
  - expressed in some formal logic
  - have a well-defined semantics
- satisfaction:
  - ideally (but not always) decided mechanically
  - based on automated deduction and/or model checking techniques

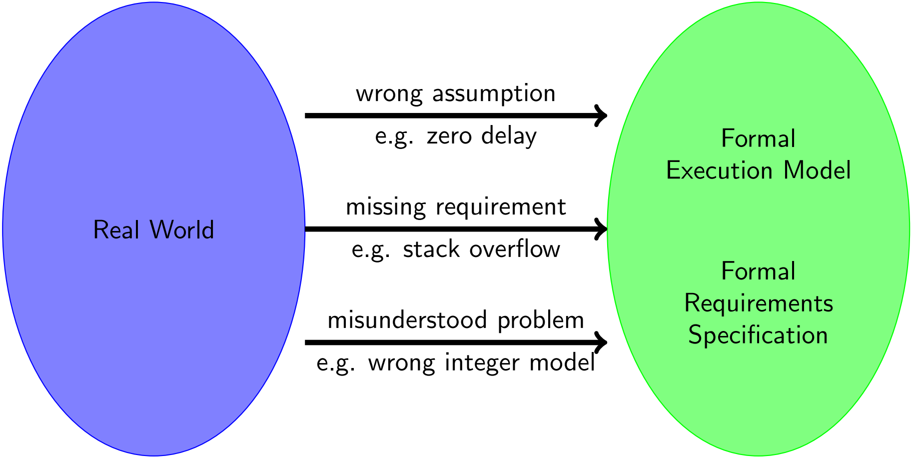
A Fundamental Fact

Formalisation of system requirements is hard

Difficulties in Creating Formal Models



Difficulties in Creating Formal Models



Another Fundamental Fact

Proving properties of systems can be hard

Level of System Description / Expressiveness of Specification

High level (modeling/programming language level)

- Complex datatypes and control structures, general programs
  - General properties
- Easier to program
  - High precision, tight modeling
- Automatic proofs (in general) impossible!

⋮

Low level (machine level)

- Finitely many states
  - Finitely many cases
- Tedious to program, worse to maintain
  - Approximation, low precision
- Automatic proofs are (in principle) possible

Current and Future Trends

Slowly but surely formal methods are increasingly used in industry.

- Designing for formal verification
- Combining semi-automatic methods with SAT/SMT solvers, theorem provers
- Combining static analysis of programs with automatic methods and with theorem provers
- Combining testing and formal verification
- Integration of formal methods into the development process

Summary

- Software is becoming pervasive and very complex
- Current development techniques are inadequate
- Formal methods ...
  - are not a panacea, but will be increasingly necessary
  - are (more and more) used in practice
  - can shorten development time
  - can push the limits of feasible complexity
  - can increase product quality
  - can improve system security
- We will learn to use different formal methods for different development stages