Formal Foundations of Software Engineering

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Pavel Parízek

FACULTY OF MATHEMATICS AND PHYSICS
Charles University

Goals of the course

- Show methods and tools for specification and modeling of
  - Requirements
  - Architecture
  - System behavior

- Show methods, languages and tools for
  - More formal design, specification and prototyping of software systems

Department of Distributed and Dependable Systems

Structure

- Lectures
  - Basic concepts (“theory”)
  - Languages (syntax, usage)
  - Tool demo & examples
- Labs
  - Small practical tasks
  - Playing with tools

Department of Distributed and Dependable Systems

Why you should attend

- Get some knowledge about formal methods
  - Commonly used languages
  - Benefits & limitations
- Usage of formal methods can actually help you in software development practice
- Ability to read and understand models created by someone else (human, IDE, generative AI)

Contents

- General introduction to formal methods
- Algebraic specification techniques (CASL)
- Rewriting systems (Maude, OBJ3)
- Model-oriented languages (Z, VDM, Alloy)
- UML (modeling) & OCL (specification)
- Petri nets (modeling concurrent systems)
- Temporal & dynamic logics (TLA+)
- *Domain-specific languages (DSLs)*

Grading

- Homeworks
  - Topics: Maude, VDM or Alloy, UML/OCL, Petri nets
  - Each awarded with 0-25 points
    - Base: 0-15 points for the solution (model, documentation)
    - Bonus: 0-10 points for discussion in person (mini-defense)
  - You need to submit at least two for “zápočet”

- Final exam
  - Basic principles, theory, comparing approaches
  - Awarded with 0-25 points

- Scale
  - 80-125: excellent
  - 65-79: very good
  - 50-64: good (pass)
  - 49 and less: failure

Contact

- Web: http://d3s.mff.cuni.cz/teaching/ntin043
- Email: parizek@d3s.mff.cuni.cz
- Room 309

Department of Distributed and Dependable Systems

Related courses

- System Behavior Models and Verification (NSWI101)
  - http://d3s.mff.cuni.cz/teaching/nswi101
- Program Analysis and Code Verification (NSWI132)
  - http://d3s.mff.cuni.cz/teaching/nswi132

General introduction to formal methods

Informal modeling and specification

- Approach 1: Creating some diagram by hand
  - Relations between components in the system
- Approach 2: Drawing finite state automaton
  - Nodes: possible states of the system
  - Transitions: actions that update state
- Limitations
  - Unclear semantics (ambiguous)
  - Validation by tools not possible

Department of Distributed and Dependable Systems

What are formal methods ?

- Mathematical techniques
- Supported by tools

- Languages
  - Specification notation
    - Formal syntax & semantics
  - Reasoning mechanism

- Enable rigorous software development

Department of Distributed and Dependable Systems

Formal description of software systems

- Interface perspective
  - Specifying requirements and desired properties
- Implementation perspective
  - Modeling internal behavior

- Characteristics
  - Expression in some formal language
  - Typically at certain level of abstraction
  - Precise, consistent, and unambiguous

What are formal methods good for

- Precisely capturing user's requirements
- Modeling behavior of critical subsystems
- Validation (testing, analysis, verification)
- Generating code from specification/models
  - Iterative refinement (transformations)
  - Model-driven engineering (MDE)

Usage pattern

1. Manually write a formal specification (model)

2. Semi-automatically validate & fix all problems

3. Iteratively transform (refine) into real code
   - Allow provably correct refinement steps
   - Implementation correct-by-construction

Benefits

- General: improved quality of software systems

- Enable system validation at very early stage
- Detecting many issues (but some remain!!)
  - ambiguity, inconsistency, plain bugs, missing pieces
- Better resilience against non-standard states

- Required for mission/safety-critical systems

Limitations

- Insufficient scalability to realistic systems
- High overall costs (man-power, time)

Practice: critical systems

- Application domains
  - transportation, military, healthcare, tele-com
- Small or middle-sized
  - 10-1000 KLOC
- Very high cost of errors

Case study: subway line in Paris

- Development process
  1. Abstract models and specifications in B
  2. Iterative refinement to concrete models
  3. Transformation to source code in ADA

- Quantitative metrics
  - Formal specification: 100 KLOC in B
  - Source code: 87 KLOC in ADA
  - Validation: proved 28K claims and found many bugs
    - No error found after the deployment !!

Case study: helicopter AH-6 (Boeing)

- Goal: robustness against cyber attacks
  - Examples: rogue software in auxiliary devices, compromised USB stick

- Main characteristics of the internal software system
  - Safety enforced through architecture (isolated modules)
  - Restricted communication over architectural boundaries
  - Access control (privileges, capabilities, runtime checks)
  - Information flow behavior (controlled and proven safe)

- Observation
  - Proper architecture and configuration are important for high assurance

- Additional information
  - G. Klein, J. Andronick, M. Fernandez, I. Kuz, T. Murray, and G. Heiser. **Formally Verified Software in the Real World**. Communications of the ACM, vol. 61, no. 10, Oct 2018
  - https://cacm.acm.org/magazines/2018/10/231372-formally-verified-software-in-the-real-world

Department of Distributed and Dependable Systems

MDE & formal methods

- MDE: model-driven engineering
  - Automated code generation
- Model-based testing
- Domains: embedded systems
  - automotive, industry manufacturing robots

Disclaimer

- Formal methods do not guarantee correctness
  - "*a formally verified program is only as good as its specification*“
- It is very easy to create a bad specification
  - Problems: incompleteness, inconsistency, typos
- Remedy: search for bugs & validate everything

Department of Distributed and Dependable Systems

Ten Commandments of Formal Methods

1. Choose an appropriate notation
2. Formalize, but do not over formalize
3. Estimate costs
4. Have a formal methods guru on call
5. Not abandon traditional development methods
6. Document sufficiently
7. Not compromise quality standards
8. Not be dogmatic
9. Test, test, and test again
10. Reuse

Management of models and specifications

- Why: formal models and specifications are normal software artifacts
- What to do: versioning, peer reviews, ...
- Problems and challenges
  - Keeping consistency of models in multiple files
  - Inconsistency between specification and code

Department of Distributed and Dependable Systems

Design by Contract

- Granularity: procedures, objects

- Preconditions
- Postconditions
- Invariants

- Methodology
  - Define contracts by hand
  - Use tool for verification

Department of Distributed and Dependable Systems