Formal Foundations of Software Engineering

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems



Martin Nečaský Pavel Parízek



CHARLES UNIVERSITY IN PRAGUE faculty of mathematics and physics

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



Structure

Lectures

- Basic concepts ("theory")
- Languages (syntax, usage)
- Tool demo & examples

Labs

- Small practical tasks
- Playing with tools



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



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, Alloy, UML/OCL, Petri nets
 - Each awarded with 0-20 points
 - You need to submit at least 2 for "zápočet"
- Final exam
 - Basic principles, theory, comparing approaches
 - Awarded with 0-30 points
- Scale
 - 71-110: excellent
 - 55-70: very good
 - 41-54: good (pass)
 - 40 and less: failure

Contact

Web: <u>http://d3s.mff.cuni.cz/teaching/ntin043</u>

- Pavel Parízek
 - parizek@d3s.mff.cuni.cz
 - room 202
- Martin Nečaský
 - necasky@ksi.mff.cuni.cz
 - room 209



Related courses

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



General introduction to formal methods



What are formal methods ?

- Mathematical techniques
- Supported by tools

- Languages
 - Specification notation
 - Formal syntax & semantics
 - Reasoning mechanism
- Enable rigorous software development

ependabl

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)

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



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



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 !!

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



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



Design by Contract

- Granularity: procedures, objects
- Preconditions
- Postconditions
- Invariants
- Methodology
 - Define contracts by hand
 - Use tool for verification

