CS:5810

Formal Methods in Software Engineering

Course Overview

Cesare Tinelli

Fall 2023

Instructional Staff

Prof. Cesare Tinelli, instructor

Mitziu Echeverria, TA

Course Info and Material

- All information, including the syllabus, available on **website** at:

  `http://www.cs.uiowa.edu/~tinelli/classes/5810/Fall23`

- Textbook: *Program Proofs* by Rustan Leino, MIT Press, 2023
- Class notes and additional reading material to be posted on the website
- Recorded lectures on **UICapture**
- Announcements and discussions on **Piazza**
- Submissions and grades on **ICON**
- Check the course website and the Piazza website regularly!

Course Design Goals

1. Learn about formal methods (FMs) in software engineering
2. Understand how FMs help produce high-quality software
3. Learn about formal modeling and specification languages
4. Write and understand formal requirement specifications
5. Learn about main approaches in formal software verification
6. Know which formal methods to use and when
7. Use automated and interactive tools to verify models and code

Course Topics

Software Specification

- High-level design
- Code-level design

Main Software Validation Techniques

Model Checking: often automatic, abstract

Deductive Verification: typically semi-automatic, precise (source code level)

Abstract Interpretation: automatic, correct, incomplete, terminating

Course Organization

- Course organized by level of specification
- Emphasis on tool-based specification and validation methods
- A number of graded and ungraded exercises, in class and at home
- Hands-on homework where you specify, design, and verify
- 3 introductory homework assignments (individually)
- 3 mini projects (in teams)
- 1 midterm, 1 final exam
- More details in the syllabus on the website

Part I: High-level Design

Language: Alloy

- Lightweight modeling language for software design
- Amenable to a fully automated analysis
- Aimed at expressing complex structural and behavioral constraints for a software system
- Intuitive modeling tool based on relational logic
- Automatic analyzer based on automated reasoning 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 I: High-level Design

Language: Alloy

- Lightweight modeling language for software design
- Amenable to a fully automated analysis
- Aimed at expressing complex structural and behavioral constraints for a software system
- Intuitive modeling tool based on relational logic
- Automatic analyzer based on automated reasoning 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: Code-level Specification

Language: Dafny

- Programming language with built-in specification constructs
- Automated static (i.e., compile-time) verification of specs
- Compilation to many other programming languages
- Sophisticated verification engines based on theorem proving techniques
- Auxiliary spec constructs to *help* verification engines complete their proofs

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

Part II: Code-level Specification

Language: Dafny

- Programming language with built-in specification constructs
- Automated static (i.e., compile-time) verification of specs
- Compilation to many other programming languages
- Sophisticated verification engines based on theorem proving techniques
- Auxiliary spec constructs to *help* verification engines complete their proofs

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