The University of Iowa Fall 2014

Formal Methods in Software Engineering

Course Overview

STAFF

- Instructor: Prof. Cesare Tinelli
- TA: Ben Berman

COURSE INFO AND MATERIAL

- All information, including the syllabus, available at:

  `http://www.cs.uiowa.edu/~tinelli/181`

- No required textbook
- Class notes and additional reading material to be posted on the website
- Suggested reference for background in logic:

  M. Huth and M. Ryan. ***Logic in Computer Science****.*
  Cambridge University Press, 2004 (2nd edition)

- Check the website regularly!

Course Design Goals

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

Course Topics

Software Specification

- High-level semantic design
- System design and behavioral properties
- Code-level properties

Software Validation

- Model Finding/Checking:
  often automatic, abstract
- Deductive Verification:
  typically semi-automatic, precise (source code level)
- Abstact Interpretation:
  automatic, correct, not complete, terminates

Course Organization

- The course is organized by level of specification
- Emphasis on tool-based specification and validation methods
- A few ungraded exercises
- Hands-on homeworks where you specify, design, and verify
- For each main topic
    - A team introductory homework
    - A team mini-project
- 1 take-home midterm, 1 take-home final exam
- More details on the syllabus and the website

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: Model-based Development

Language: Lustre

- Executable specification language for synchronous reactive systems
- Designed for efficient compilation and formal verification
- Used in safety-critical applications industry
- Automatic analysis with tools based on model-checking techniques

Learning Outcomes:

- Write system and property specifications in Lustre
- Perform simulations and verifications of Lustre models
- Understand what can and cannot be expressed in Lustre

PART III: CODE-LEVEL SPECIFICATION

Languages: Dafny

- Object-based PL with specification constructs
- Specifications embedded in source code as formal contracts
- Mostly an academic tool but with a very sophisticated verification engine
- Automatic analysis based on theorem proving techniques

Learning Outcomes:

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