EECS498

Formal Verification of Systems Software

Instructor
Manos Kapritsos
https://manos.prof

Class meeting times (in person)
2x 80 minutes (Lecture)
1x 50 minutes (Lab)

Course description

For the past 60 years, we have been relying on testing for building robust, bug-free software. And yet, despite our best efforts, bugs slip past our test cases all the time, resulting in a number of incidents that range from benign to catastrophic. In this class, you will learn an entirely different way to build robust software: by writing code that is formally proven to be **free of bugs**. We will approach this problem from a systems perspective, focusing on how to design complex (often distributed) systems, while still being able to reason about their correctness. At the same time, the class will give you hands on experience with Dafny, one of the most promising tools for practical formal verification of systems software.

During this course, you will learn how to formally *specify* a system's behavior, how to *prove* that the high-level design of the system meets that specification and finally how to show that the system's low-level implementation retains those properties. The course **does not assume any prior knowledge in formal verification**. We will start from the basics of the Dafny language and build from there. In the end, you should be able to design and prove correct a complex system. The experience of doing so will make you a more careful and effective programmer, even when you don't write formally verified code.

Learning objectives

By the end of the course, you should:

- Understand the fundamentals of formal verification (specification, proof, invariants, etc.)
- Be familiar with Dafny at both the protocol and implementation level
- Be able to design a new system as a state machine in Dafny
- Understand the concept of an inductive invariant and how to find one
- Be able to prove the correctness of a protocol-level state machine
- Understand the concept of refinement
- Be able to perform a refinement-based proof on a complex system
- Be familiar with more advanced topics such as asynchronous APIs and proof automation

Tentative list of assignments

- Exercise set #1 (done individually)
  - Dafny basics: datatypes, assertions, predicates, etc.
  - Recursion
  - Loop invariants
  - Specification
- Exercise set #2 (done individually)
  - State machines
  - Inductive invariants
- Exercise set #3 (done individually)
  - Distributed systems
- Project 1 (groups of 2)
  - Design and prove correct a Distributed Lock service
- Exercise set #4 (done individually)
  - Refinement
- Exercise set #5 (done individually)
  - Asynchronous API
  - Application correspondence
- Project 2 (groups of 2)
  - Design a Sharded Hash Table (SHT) protocol and prove it is correct using refinement

Grading policy (subject to change)

- Problem sets: 30%
  - Set #1: 8%, Set #2: 8%, Set #3: 4%, Set #4: 4%, Set #5: 6%
- Projects: 30%
  - Project 1: 15%
  - Project 2: 15%
- Exams: 40%
  - Midterm exam: 20%
  - Final exam: 20%

Tentative lecture schedule

Lecture 1: Introduction

Lecture 2: Dafny mechanics

Lecture 3: Collections, quantifiers

Lecture 4: Datatypes, Imperativeland

Lecture 5: Recursion, loop invariants, specification

Lecture 6: State machines and behaviors

Lecture 7: State machines and proving properties

Lecture 8: JNF, Inductive invariants

Lecture 9: Inductive invariants, Crawler 2

Lecture 10: Leader election demo
Lecture 11: Distributed systems
Lecture 12: Distributed systems (cont.), 2PC
Lecture 13: Refinement
Lecture 14: Asynchronous specs
Lecture 15: Application correspondence, triggers
Lecture 16: Modules and automation
Lecture 17: Cross-host concurrency
Lecture 18: Multi-level refinement
Lecture 19: Incremental Inference of Inductive Invariants
Lecture 20: Case study: IronFleet
Lecture 20: The future of systems verification
Lecture 21: Recap of second half