Software Verification: Basics of Formal Methods

Meeting Times: Two 120-minute lectures + X hour as needed
Prerequisites: CS 1, CS 10 or equivalent, CS 30

Course Description

This course introduces formal verification as a practical engineering discipline for building software you can actually trust. Rather than treating formal methods as an academic curiosity, we study them the way working engineers at AWS, Microsoft, Airbus, and NASA do: as a toolkit for answering the question *"is this system actually correct?"* before it fails in production.

Students progress through three tiers of verification technology: automated model checkers (TLA+, Alloy), auto-active verifiers (Dafny), and interactive theorem provers (Coq, Isabelle). Each tool is grounded in real-world case studies, from the Therac-25 radiation therapy accidents to AWS's verification of DynamoDB, so students learn not just *how* to use each tool, but *when* to reach for it.

Learning Outcomes

By the end of this course, students will be able to:

1. Explain why formal methods matter by reference to specific historical software failures and industry adoption stories.
2. Specify system behavior in TLA+ and verify safety and liveness properties using the TLC model checker.
3. Express structural and relational properties in Alloy and interpret counterexamples.
4. Write pre/postconditions and loop invariants in Dafny and debug verification failures.
5. Construct interactive proofs in Coq and Isabelle for small but nontrivial theorems.
6. Select an appropriate formal methods tool for a given problem domain and justify the trade-off between automation and expressiveness.

Tools & Software

Students will install and use, in order of appearance:

- **TLA+ Toolbox** with the TLC model checker
- **Alloy Analyzer**
- **Dafny** (via VS Code extension)
- **Coq** with CoqIDE or Proof General
- **Isabelle/HOL**

A setup guide and lab environment will be provided before Week 1.

Weekly Schedule

Week 1: What Are Formal Methods, and Why Do They Matter?

Historical failures and their lessons: Therac-25, Ariane 5 Flight 501, Knight Capital's 45-minute bankruptcy, Mars Pathfinder priority inversion. The spectrum of verification: testing, static analysis, model checking, auto-active verification, interactive proof. Where each technique sits on the automation-vs-expressiveness curve.

Lab: Course tool setup and a guided walkthrough of a small buggy system that testing misses but model checking catches.

Week 2: TLA+ and Model Checking

Specifying systems as state machines. TLA+ syntax: variables, actions, next-state relations. Safety vs. liveness. Using TLC to exhaustively explore state spaces.

Case study: AWS's use of TLA+ on DynamoDB and S3.

Lab: Specify and verify a simple distributed protocol.

Week 3: Alloy and Relational Logic

Structural modeling: signatures, relations, and constraints. Scope-bounded analysis and why it differs philosophically from unbounded model checking. Reading counterexamples as evidence.

Case study: Alloy analysis of security protocols and filesystem designs.

Lab: Model an access-control policy and find its holes.

Week 4: Dafny I: Specifications and Automated Proof

Transitioning from "check my model" to "prove my code." Preconditions, postconditions, and frame conditions. How Dafny's SMT solver (Z3) automates what used to require manual proof.

Case study: Microsoft's IronFleet.

Lab: Verify simple array and list routines.

Week 5: Dafny II: Loop Invariants and Debugging Proofs

The hardest idea in the course: loop invariants. Strategies for discovering invariants. Termination and decreases clauses. Why a verifier fails silently, and how to read what it's telling you. Recursion and induction in Dafny.

Lab: Verify binary search and an in-place array reversal; these are the exercises where students typically get stuck, by design.

Week 6: Coq and Interactive Theorem Proving

Why some things can't be automated. Dependent types, tactics, and the Curry-Howard correspondence, pragmatically. Building proofs step by step.

Case study: CompCert's verified C compiler.

Lab: Guided proofs from *Software Foundations* on lists and natural numbers.

Week 7: Isabelle/HOL and the Big Picture

Isabelle's approach contrasted with Coq's.

Case study: seL4, the verified microkernel, and what 200,000 lines of Isabelle proof bought its users.

Course capstone: choosing the right tool for a problem. Discussion: where formal methods are heading in industry.

Lab: A guided Isabelle proof, followed by a capstone exercise where students must pick a verification technology for a given specification and justify the choice.

Assessment

Component	Weight
Weekly labs (7)	40%
Problem sets (4)	25%
Midterm project (TLA+ or Alloy model of a chosen system)	15%
Final project (Dafny verification or Coq/Isabelle proof)	20%

Course Policies

Late work: Labs may be submitted up to 48 hours late with a 20% penalty. You all get two free late days with the understanding that life happens.

Collaboration: Discussion of concepts and debugging strategies is encouraged. All submitted specifications, proofs, and code must be your own. If you intend to work together as a team, you each need to submit the assignment, and flag who did what.

Policies

Academic Integrity

Students are expected to follow Dartmouth's and Guarini's Academic Honor Principles/Policies.

Religious Observances

Dartmouth has a deep commitment to support students' religious observances and diverse faith practices. Some students may wish to take part in religious observances that occur during this academic term. If you have a religious observance that conflicts with your participation in the course, please meet with me as soon as possible, before the end of the second week of the term at the latest, to discuss appropriate course adjustments.

Student Accessibility and Accommodations

Students requesting disability-related accommodations and services for this course are required to register with Student Accessibility Services (SAS; Apply for Services webpage;

student.accessibility.services@dartmouth.edu; 1-603-646-9900) and to request that an accommodation email be sent to me in advance of the need for an accommodation. Then, students should schedule a follow-up meeting with me to determine relevant details such as what role SAS or its Testing Center may play in accommodation implementation. This process works best for everyone when completed as early in the quarter as possible. If students have questions about whether they are eligible for accommodations or have concerns about the implementation of their accommodations, they should contact the SAS office. All inquiries and discussions will remain confidential.

This syllabus is a guide; specific dates and readings may shift to match class progress.