Formal Methods in Software Engineering

§1 Annotation

Ever dreamed of writing perfect, bug-free code? In this course, you'll dive into the depths of formal methods: explore propositional and first-order logic, master SAT and SMT solvers, and discover the fundamentals of verification — from transition systems and Kripke models to specification and temporal logics (LTL, CTL, ATL). We'll examine SAT- and BDD-based verification approaches, look at bounded model checking (BMC) and property-directed reachability (IC3/PDR), and show how to systematically ensure reliability and correctness (safety and liveness properties) in complex systems. All theoretical concepts are reinforced through hands-on examples using academic tools such as NuSMV, Alloy/Forge, and Dafny, so you can see exactly how research ideas become practical solutions. This course is ideal for anyone seeking to understand the core principles of verification and apply them in real projects, aiming for the highest standards of software quality and reliability.

§2 Learning Outcomes

By the end of the semester, students should be able to:

- Demonstrate fluency in propositional and first-order logic for use in formal specification.
- Encode verification problems in SAT/SMT formulations and effectively utilize modern solvers (e.g., Cadical, Z3).
- Model reactive systems using transition systems and Kripke structures, and specify correctness properties in temporal logics (LTL, CTL, ATL).
- Employ a range of model checking techniques — including SAT-based, BDD-based, bounded model checking, and IC3/PDR — to verify safety and liveness properties.
- Use software tools (NuSMV, Alloy/Forge, Dafny) to perform model checking and automated reasoning on simplified but realistic software systems.
- Critically assess industrial and academic literature on formal verification, synthesizing insights into a final project or case study presentation.

§3 Prerequisites

Students are expected to have prior exposure to:

- Discrete mathematics (propositional logic, set theory)
- Basic proof techniques (natural deduction)
- Automata theory and formal languages

- At least one programming language

Experience with software engineering or systems design is helpful but not required.

§4 Course Format

- **Lectures**: Present theoretical foundations and methods.
- **Seminars**: Discuss research papers, advanced topics, and industrial cases.
- **Assignments**: Reinforce core concepts via problem sets and tool-based labs.
- **Project**: Undertake a substantial verification or modeling task.
- **Exam**: Assess understanding of theoretical and applied aspects of formal methods.

§5 Course Structure

5.1. Overview

- **Introduction to Formal Methods**: Motivation, applications, and core concepts.

5.2. Propositional Logic

- **Content:** Syntax, semantics, normal forms (CNF), tautologies, satisfiability.
- **Applications:** Encoding simple constraints, forming the basis for SAT solving.
- **In-class/Lab:** Translating small puzzles or system properties into propositional logic.

5.3. SAT

- **Content:** SAT-solving fundamentals, DPLL backtracking, conflict-driven clause learning (CDCL).
- **Tools and Demos:** MiniSAT, short solver experiments.
- **Assignments:** Students encode a small problem (e.g., Sudoku or scheduling) and run a solver to find solutions or detect unsatisfiability.

5.4. First-Order Logic

- **Content:** Syntax, semantics, quantifiers, free vs. bound variables, theories in FOL.
- **Deduction in FOL:** Natural deduction, sequent calculus (at a high level).
- **Relevance:** Understanding how real software specifications require more expressive logic than propositional alone.

5.5. SMT

- **Content:** Extending SAT to Satisfiability Modulo Theories (linear arithmetic, arrays, bitvectors).
- **Standard Formats:** SMT-LIB language for specifying problems.
- **Tools and Frameworks:** Z3 usage.
- **Nelson-Oppen Framework:** Combining theories for more complex verifications.

5.6. Model Checking

- **Transition Systems and Kripke Structures:** Modeling program states, transitions, labeling atomic propositions.
- **Temporal Logics:** Safety, liveness, fairness; LTL, CTL, ATL for specifying system properties.
- **NuSMV Tutorial:** Creating models, writing properties, interpreting counterexamples.
- **Alloy / Forge:** Relational modeling, generating instances or counterexamples.
- **Dafny:** A language+tool that integrates specification, verification, and proof-like checks.

5.7. Advanced Model Checking Techniques

- **SAT-based Model Checking:** Bounded model checking (BMC) with unrolling.
- **BDD-based Model Checking:** Using binary decision diagrams for state-space representation.
- **k-Induction and Inductive Invariants:** Proving properties beyond a fixed bound.
- **IC3 / PDR:** Incremental construction of inductive proofs, property-directed reachability.
- **Comparisons and Trade-Offs:** When each technique excels or struggles.

§6 Projects and Assignments

Students will complete:

1. **Lab Exercises:** Applying each method or tool in small-scale examples — e.g., encoding a puzzle in SAT, verifying a simple concurrency protocol in NuSMV, exploring an Alloy model.
2. **Major Course Project:** A deeper verification or specification task selected from instructor-provided ideas (e.g., verifying a distributed cache algorithm in NuSMV or Alloy, experimenting with Dafny for array safety proofs).

3. **Literature Reviews and Presentations:** Groups research an industrial or academic use of formal methods (e.g., hardware verification at Intel, flight software checks at NASA). They present both the methodology and lessons learned.

§7 Grading and Evaluation

Homework (20%)	Review (20%)	Project (30%)	Exam (20%)	Participation (10%)

7.1. Homework Assignments (20%)

Assignments focusing on each logic or tool introduced.

7.2. Literature Review & Presentation (20%)

Students analyze a real case study, synthesizing insights from papers or technical reports.

7.3. Term Project (30%)

A substantial modeling/verification effort that integrates multiple techniques from the course. Includes a written report and final presentation.

7.4. Final Exam (20%)

Tests both theoretical understanding (logic, model checking principles) and tool-based reasoning (e.g., how to encode properties in NuSMV).

7.5. Participation (10%)

Evaluates discussion contributions, attendance, engagement in peer reviews, and collaboration in labs.

§8 Course Policies

- Standard university policies on academic integrity, attendance, and accommodations apply.
- Students are encouraged to regularly collaborate and discuss concepts, but all submitted work must be their own unless explicitly stated otherwise.
- Late submissions will be penalized unless prior arrangements are made with the instructor.

§9 Resources

Lecture notes, slides, and additional readings will be uploaded in the course GitHub repo: https://github.com/Lipen/formal-methods-course.

§10 Contacts

...