

Automated Verification

COURSE
ECE 750

T29

Building Reliable Software in the Age of AI

Instructor **Arie Gurfinkel** Electrical and Computer Engineering Website ece.uwaterloo.ca/~agurfink/ece750t29

Why verification matters now

AI accelerates software development, increasing the need to establish that software behaves as intended. It also helps discover and exploit weaknesses in existing systems. OpenAI’s reports of [agents escaping sandboxes](#) and [compromising external systems](#) illustrate how software flaws and gaps in isolation can undermine intended protections. Formal reasoning provides ways to specify these protections, find violations, and establish guarantees under explicit assumptions.

AI also makes verification more accessible by helping construct specifications, models, and proofs. This course develops the judgment to guide that process and assess its results, together with the automated reasoning foundations behind the tools. We move from practical verification tasks to logical encodings, solver algorithms, memory and ownership, and invariant discovery. Later topics connect proof assistants with AI-assisted software and verifier development. Examples connect these methods to software libraries, systems code, and compiler transformations, with projects offering further applications in students’ research areas.

What you will learn

- 01 Formulate useful correctness claims and assess their assumptions, scope, and adequacy.
- 02 Use verification tools and guide AI agents in selecting models, specifications, and proof strategies.
- 03 Explain how verification conditions, memory representations, and solver algorithms influence results and performance.
- 04 Interpret counterexamples, inferred invariants, and formal proofs; distinguish bounded evidence from inductive guarantees.
- 05 Carry out and communicate a focused investigation of a verification problem or technique.

Learning format and preparation

Lectures develop the core concepts through worked examples, tool demonstrations, and discussion. Research papers and tutorials accompany the lectures, typically one primary reading per meeting. Projects give students room to explore advanced topics in depth.

Students should be comfortable with programming and mathematical reasoning. Examples use C and Rust; students unfamiliar with Rust are expected to learn it independently. No prior experience with verification tools or Lean is required.

Assessment

20%

Two short assignments

Practical verification and interpretation; 10% each.

15%

Technical presentation

A focused topic or investigation supporting the project.

55%

Project and final report

Substantive investigation, evidence, and critical analysis.

10%

Reading and participation

Preparation and substantive engagement with course material.

Course at a glance

The schedule below describes the intended progression by teaching week, excluding the reading break. Topic order, readings, examples, and software tools may be adjusted while retaining the learning objectives and core coverage. Presentations may be interleaved with the later topics.

WEEK	THEME	TOPICS AND QUESTIONS
01	What does it mean to verify?	Correctness claims, specifications, assumptions, vacuity, and bounded checking through practical examples.
02	Deductive verification	Contracts and inductive invariants for Rust libraries and data structures; AI-assisted proof development and feedback.
03	SMT and equality reasoning	Solver interfaces, models, cooperation between decision procedures, and congruence closure.
04	Verification conditions	Reducing program correctness to logic; program transformations, query encodings, and interpreting results.
05	Memory and ownership	Memory safety in systems code: aliasing, array-based models, ownership, and prophecy for mutable state.
06	Automatic invariant discovery	Abstract interpretation for low-level code, such as kernel extensions; approximation, inferred facts, and false alarms.
07	Horn clauses and induction	Constrained Horn clauses for unbounded program behavior and resource-management protocols; from bounded checks to induction.
08	Property-directed reasoning	IC3/PDR and symbolic invariant inference; generalization, summaries, and the effect of encoding choices.
09	Proof assistants with AI agents	Reading formal models and theorem statements; software modeling and reviewing agent-assisted proofs in Lean.
10	From Rust to functional proofs	Functional translation, embeddings, and reasoning about mutation; proof decomposition and automation.
11	Checking verification tools	Equivalence and product programs for compiler transformations; translation validation and AI-assisted verifier development.
12	Student investigations	Project presentations, technical discussion, and synthesis of the course's methods and limitations.

Applications and tools

Representative applications include data-structure correctness, memory and resource safety in systems code, and correctness of program transformations. Each illustrates how the specification, program model, and reasoning method shape the guarantee.

Tools may include CBMC, Kani, or SeaHorn for bounded checking; Verus and Z3/Spacer for SMT-based verification; and Lean-based workflows such as Aeneas. The selection may evolve with availability and suitability for the exercises.

Projects and AI assistance

Projects may apply a method, reproduce or compare results, examine a verification claim, or extend a technique. Possible directions include distributed protocols and concurrent software. The final report develops the investigation beyond the presentation; original research is welcome but not required.

AI assistance is part of the course. Students remain responsible for understanding, explaining, and validating submitted work, including its specifications and assumptions. Assignment instructions will state collaboration and AI attribution requirements.