There is a growing need for effective methods of designing correct computer hardware and software, particularly for safety critical systems such as automotive embedded systems software or for systems that are costly to recall such as microprocessors (e.g., the $500M loss by Intel on the Pentium division error.) To cope with these problems, Formal Methods have been developed based on the use of mathematical logic precision tools for specifying and reasoning about program correctness. Hardware, software, and design automation companies use formal methods to make their products more reliable and less costly to develop.

This course will survey the basic concepts of formal methods. The emphasis will be on using and applying mathematical logic plus finite state systems theory to program verification and debugging.

**TOPICS:**

1) Preliminaries: Discrete math, logic, automata, transition systems.

2) Verification of Sequential and Nondeterministic Programs:
   - Flowchart programs: invariants, well-founded sets.
   - Assertional reasoning: partial and total correctness, compositionality.
   - Predicate transformers: weakest precondition calculus: wp, wlp.

3) Verification of Concurrent and Reactive Programs:
   - Linear temporal logic: G (always), F (sometime), X (nexttime), etc.
   - Branching temporal logic: logics CTL, CTL*
   - Model checking:
     - by Tarski-Knaster theorem for branching time;
     - By language containment for linear time.
   - State explosion: The problem and some solutions.

4) As time permits advanced topics from:
   - Linear time vs. Branching time temporal logics.
   - Tableaux for testing satisfiability and automata construction.
   - Binary Decision diagrams for symbolic model checking.
   - Abstraction techniques: homomorphisms, bisimulations, symmetry, etc.

**TEXTS:**

Recommended: Huth and Ryan, *Logic for Computer Science.*

**GRADING POLICY:**

Participation: 50%
Research/Expository Project and 15 page written Report: 50%