Overview:
Brief History
Verification in the Past

- In 70s
- Proving programs correct
  - Floyd, Hoare, Dijkstra, ...
  - Philosophy: programmers write programs and prove them correct with a prover.
- Failed but is resurging
  - All or nothing approach: no way to find bugs.
  - Heavily manual ... non-appealing!
Success Stories

**SPIN** (Holzmann)

- Explicit-state model checker
- Heuristics to control state-space explosion
  - partial order reduction
  - hashing and approximate search
- Specification: LTL/automata
Success Stories

- **SMV** (Started by McMillan), later NuSMV

- Symbolic model checker using binary decision diagrams (BDD)

- handles large state spaces

  - heuristics to handle search spaces well

  - specification: CTL (and later LTL)

- by far the most useful for hardware
Success Stories

- Big advances in SAT solvers
  - zChaff (Princeton)
    - can handle formulas with 100,000 variables and millions of clauses!

- Boosted the idea of Bounded Model Checking (BMC)
  - AWS Security Guy refers to these!
Success Stories

- The **SLAM** tool from Microsoft Research (Ball and Rajamani)

- **Static Driver Verifier**: big breakthrough

  - model checker that validates device drivers against formal spec.

  - Key ideas: predicate abstraction, algorithms for pushdown automata, BDDs for boolean programs.
Success Stories

- **The Metal project** from Stanford (Dawson Engler)
  - Static analysis to find patterns of bad programming in system code
  - very successful in error finding: hundreds of bugs found in Linux/BSI.
Success Stories

- Light-weight static analysis
  - pointer chasing, data dependency analysis, ...
- Codesurfer by GrammaTech
- Testing tools: PEX by Microsoft, DART, etc.
Correct by Construction

- **Program Synthesis**: produce a program that satisfies a specification

- **Specifications**: logical or examples

- **Algorithms** for performing the synthesis

- **Formal models**: to define state space of viable candidates.
Program Synthesis

- End user programming: for those who know zero programming
  - Example: Excel’s Flashfill

- Menial Programming Tasks: saving precious programmer time
  - The reverse Von Neumann

- Removing Human Error: removing human error
Learning Objectives
Ultimate Goal:

Change the way you think and reason about programs by producing a paradigm shift in your thinking.
Learning Objectives

- How to reason about programs
- Hoare Logic and **Invariants**
- Become familiar with **formal models**
- CFGs, state transition systems, symbolic representations, ...
- Specification of properties
- Temporal logics (LTL, CTL), assertions, pre-post conditions
Algorithms/techniques for reasoning

- Invariants, Fixpoints, Model Checking

You will *teach yourself* tools such as:

- Dafny (a theorem prover)
- A SAT and an SMT solver
- NuSMV: a model checker
A rough outline to the course
Course Progression by Topic

- Program Correctness
- Recursive Programs
- Iterative Programs
- Hoare Logic
- Decision Procedures
- Symbolic Methods
- Temporal Logics
- LTL
- CTL
- Model Checking
- Program Synthesis
End of Intro.

Questions?
Text Books, Aids, ...

- No official Text

- A list of helpful references are posted on the course webpage

- Others may follow throughout the term: books, papers, theses, ...

- Four TAs

- They will do most tutorials for you and partially help you use the tools and help you with problem solving.
Prerequisites

* Basic knowledge of Automata and Languages, Theory of Computation, Propositional (boolean) logic, First Order Logic, set theory, algorithms, data structures, and programming
Now, a word of advice ...
Don’t take this course if ...

- You don’t like logic
- You don’t like theory
- Your knowledge of logic/theory is shaky
- You want to an easy course to satisfy a breath/depth requirement
- You think this is a systems course
Don’t take this course if …

- You think you know what is useful and what is not!
- You are not self-sufficient at learning new things quickly on your own
- You are bad at working in a team
- It will basically be assumed that you can dig yourself out of a hole with the help of your peers!
The course is adversarial by design and, out of necessity.
What does adversarial mean?

This is an **elective 4th** year course.

It will not be as cleanly streamlined as your 1st/2nd year courses.

There are lectures, but you are meant to **learn a lot on your own**.

**Problem solving** requires undefinable background.

You are meant to learn to use new tools on your own with **shady** online documentation.
Just do your best and do not worry about grades!