Welcome to CS 598

“Software Verification”
better called
“Applications of software verification
techniques, logic, machine learning
and program synthesis techniques to
exciting new domains”

Fall 2015

Madhusudan Parthasarathy (Madhu)
madhu@cs.uiuc.edu
What is this course about?

• The course, on purpose, will have a dynamic agenda, evolving depending on the course participants, the discussions, and the papers we read and the ideas we discover.

• Broad theme:
  There are a huge number of techniques that have emerged in recent years in software verification, program synthesis, logics, and machine learning.

  Can we use these techniques in new pastures, combining techniques to solve known problems or even solving problems yet unformulated?
Techniques

• Software verification [CS477]:

Floyd-Hoare style deductive verification: reducing program verification to logical reasoning

Verification conditions, weakest-pre/strongest post, contract-based programming, Inductive invariants, ghost code, relating verification to formal semantics.

Automation: Using SMT solvers, inductive invariant synthesis, abstraction (abstract-interpretation, predicate abstraction) abstraction followed by model-checking (SLAM)

Natural proofs: sound but incomplete automatica ways of reasoning about expressive properties of software.
Techniques

• Logic [CS498 Logic series]
  Logic as a universal language for modeling, for semantics, for reasoning, for specification, for verification.

  Classical logic: FOL, sound and complete proof systems, decidability/undecidability boundaries

  Decidable logics (e.g., Presburger arithmetic)

  SMT solvers: fast logic solvers for simple logics

  *Natural proofs: sound but incomplete procedures for reasoning with undecidable logics*
Techniques

• Machine Learning [CS 446: Dan Roth this sem]

  – Unconventional algorithms:
    Designing algorithms when the specification itself isn’t clear. Typically where humans do better.

  – Algorithms that automatically improve on (possibly ill-defined) tasks through experience.

  – Improves with experience at some task T
    - with respect to some performance measure P
    - based on experience E

Techniques include: supervised classification algorithms including learning Boolean functions (CNF/DNF/decision trees), learning threshold functions, SVMs, online learning, Bayesian learning, structured learning, expectation-minimization algorithms, clustering…
Techniques

• Program Synthesis [no course]
  – Can we synthesize a piece of code, a piece of a system, currently written by humans?

Applications:
  Improving programmer productivity
  Enabling end-users, who cannot program, program!

– Success stories:
  • Gulwani et al: Synthesizing Excel programs that transform Excel documents. Incorporated into MS Excel.
  • Smaller successes: see JSNice—making JS code nice

• Large national efforts:
  - NSF Expeditions project ExCAPE
  - DARPA MUSE program on helping programmers complete program drafts using BIG repositories of existing code.
Err = 0.0;
for(t = 0; t<T; t+=dT){
    if(stage==STRAIGHT){
        if(t > ??) stage= INTURN;
    }
    if(stage==INTURN){
        car.ang = car.ang - ??;
        if(t > ??) stage= OUTTURN;
    }
    if(stage==OUTTURN){
        car.ang = car.ang + ??;
        if(t > ??) break;
    }
    simulate_car(car);
    Err += check_collision(car);
}
Err += check_destination(car);
Techniques

- Program Synthesis [no course]
  - How do we synthesize small pieces of code using search?
    
    Techniques:
    - Brute force explicit search using symmetry reductions
    - Using constraint solvers to search symbolically for solutions
    - Randomized search using MCMC methods
    - Machine learning based approaches

How do we synthesize small pieces of code using code repos?
- Machine learning
  (see work by Vechev et al on JSNice, Nice2Predict)
New pastures

Program Synthesis Applications

- Program specification mining
- Sketch completion, code completion,
- Program transformations, helping programmers make transformations,
- Synthesizing distributed protocols,
- Synthesizing networks layers,
- Helping programmers understand code, de-obfuscate code
- Helping end-users in IoT settings to manage their devices/
programming your home robots.
New pastures

Applications in security

Applying verification, synthesis, logic, and machine-learning techniques to problems in security

- Detecting and preventing DOS attacks
- Helping programmers write in a secure platform/paradigm.
- Mobile app security:
  Certifying apps to be obviously secure using natural proofs
- Hyperproperty verification
  How do we verify information flow?
  Relational verification

- What are the pressing problems in security of cloud, mobile, and traditional spaces that we can solve using our techniques?
New pastures

Verification of large important software

Can we actually build some important software from scratch with important properties verified?
  [ExpressOS project]

Can we verify important properties of mobile apps?
  (say that they meet their manifest security properties?)

Can we build a secure cloud platform?
  [secure against other programs running in the cloud;
   secure against cloud operators]

Can we verify important distributed protocols / platforms?
  [ Verifying distributed key value-stores using `program-models` ]
## Applications of machine learning

Can we synthesize annotations using machine learning for verifying some important domain in software verification? [e.g., memory safety, device drivers, ICE learning of invariants, linux kernel memory safety]

Can we use machine learning to synthesize code?

Can we use machine learning to synthesize specifications for code? [Daikon, synthesizing pre/post conditions, synthesizing stateful interfaces for libraries, synthesizing specs for the Android library]

Can we use machine learning using code repositories to do code completion?

Can we combine synthesis using code repos and search?
Aims of this course

Learn some core techniques:
  Verification
  Logic
  Synthesis
  Machine-learning

Read, present, and discuss papers that explore new pastures where either the core techniques have already been used or where we believe they could be applicable.

Survey the related work and landscape of problems and techniques used.

Hash out the specifics as we move along.
Course mechanics

Some homework:
- Covering core techniques

Read and present papers
- starting very soon
- start choosing first paper next week

Paper presentations:
- all must read the papers before coming to class
- presentation followed by questions and discussion

Project:
- Work in teams or individually
- Team presents papers related to project and discusses with the whole class
- Write a paper; try to publish if it looks publishable.
Input from you!

- Write your name and id
- PhD? MCS? MS?
- Advisor?
- Current project(s)
  Describe them in detail.
- What “new pastures” interest you? What problems interest you?
  Write in detail.

Based on your input, over the next week, we will finalize on the topics and initial set of papers.
Projects will be decided a little later.

First few weeks will be lectures by me on core techniques.