Welcome to CS 598MP

Software Verification

Fall 2010

Madhusudan Parthasarathy (Madhu)
madhu@cs.uiuc.edu

What is this course about?

- Advanced research-level course on software analysis
  - Mainstream software verification
    (as found in recent papers in
    POPL/PLDI/CAV/TACAS/ICALP/PPOPP/SAS etc.)
  - No rewriting techniques; no semantics
    (covered in other courses)

  - First 1/3 course: Core techniques
  - Second 1/3 course: Presentations on techniques by everyone
  - Third 1/3 course: Projects, and project presentations
  - No final exam; problem sets only on core techniques

  - Ideally, each person is expected to learn all core techniques, learn one particular technique in detail, and use it in a concrete project, giving a theoretical or practical contribution.
  - Project will hopefully be at the level of a conference publication

Landscape of program verification

Abstract Interpretation

Static analysis/ data-flow analsyis

Counter-example guided abstraction + model-checking

Explicit model checking

Floyd/Hoare style verification

Types -- engineered for each property

Shallow specs; more automataed

Complex specs; less automated

Testing

Symbolic testing using SAT and SMT solvers

Deductive verification using SMT solvers (unroll loops)

Dimensions

- Complex control
  - Handling recursion, concurrency, message-passing
- Handling data
  - Reasoning with integers, strings, objects, etc.
- Handling heaps
  - Unbounded data-structures like lists, trees, etc.
- Adapting to concurrency
  - Interleaving explosion, compositional reasoning, partial-order reductions

Techniques: Logic

- Logic!!
  - Program analysis of all kinds requires reasoning

    (E.g. x>y & x'=x+1 => x'>y;

    adding x larger to the end of a sorted list is still sorted if x is larger than all elements in the list)
  - Advent of SMT solvers:
    - Constraint solvers for particular theories
    - Engineering abstraction of logical reasoning that any program analysis tool can use
    - Completely automated
    - Boolean logic: SAT
    - Other theories: linear arithmetic, arrays, heaps, etc.

Techniques: Logic

- Use of logic
  - Hoare-style verification: Verification conditions
  - Abstraction: finding the abstract transitions
  - Symbolic execution: solving path constraints to generate input
  - Unsound deductive verification: Again verification conditions
  - SMT solvers enable all these technologies!

  - Interpolation in symbolic exploration
  - Circular compositional reasoning

Techniques: Model checking

- Model-checking
  - Reachability checking of data-abstracted models
  - Explicit search
  - Binary decision diagram (BDD) based search
  - SAT based search for bounded model checking
  - Unbounded SAT-based search (interpolation, etc.)
  - Searching programs with recursive procedures
  - Concurrent model-checking
  - Compositional model-checking

  - Enables key algorithms for model-checking using abstraction

Techniques: Model checking

- Model-checking
  - Reachability checking of data-abstracted models
  - Explicit search
  - Binary decision diagram (BDD) based search
  - SAT based search for bounded model checking
  - Unbounded SAT-based search (interpolation, etc.)
  - Searching programs with recursive procedures
  - Concurrent model-checking
  - Compositional model-checking

  - Enables key algorithms for model-checking using abstraction

Some successful tools

- Testing by Symbolic executions
  - PEX (http://research.microsoft.com/en-us/projects/pex/)
    Whitebox testing
    (internal to Microsoft; available in Visual Studio for .NET)
    PEX-for-fun website
  - SAGE
    Checks for security vulnerabilities in Windows code
    stems from DART/CUTE : ``concolic testing''
  - VeriSol (NEC) for Verilog
  - …

Some successful tools

- Explicit model-checking
  - Verisoft (http://cm.bell-labs.com/who/god/verisoft/)
    - Fully automatic tool;systematic state-space exploration; 1996; Bell-labs
  - SPIN (http://spinroot.com/spin/whatispin.html)
    - Checks software models
  - CHESS
    - Concurrent programs with bounded preemptions
- Partially symbolic approaches
  - Java PathFinder (NASA): (http://javapathfinder.sourceforge.net/)

Some successful tools

- Symbolic model-checking for finite-state systems/hardware
  - SMV : Symbolic Model Verifier

    NuSMV: http://nusmv.irst.itc.it/

    BDD-based model-checking

    SAT-based bounded model-checking too
  - Software, handling recursion/concurrency
    - BEBOP (within SLAM): http://research.microsoft.com/en-us/projects/slam/
    - MOPED: http://www.fmi.uni-stuttgart.de/szs/tools/moped/
    - GETAFIX (here!): http://www.cs.uiuc.edu/~madhu/getafix/
    - TVLA: http://www.math.tau.ac.il/~tvla/

Some successful tools

- Abstraction based tools
  - ASTREE – abstract-interpretation (http://www.astree.ens.fr/)

    For flight control software
  - SLAM /SDV – Microsoft (http://www.microsoft.com/whdc/devtools/tools/sdv.mspx)

    For device drivers
  - FSoft – NEC

    (http://www.nec-labs.com/research/system/systems_SAV-website/index.php)
  - TVLA (http://www.math.tau.ac.il/~tvla/)

    Abstractions for heaps using shape analysis
  - Yogi – MSR

    http://research.microsoft.com/en-us/projects/yogi/

    Combines static verification with testing

Some successful tools

- Deductive Floyd-Hoare style verification
  - ESC-Java
  - Boogie (MSR) (http://boogie.codeplex.com/)
    Also VCC (http://research.microsoft.com/en-us/projects/vcc/)
    (use Z3 SMT solver)
  - STORM (http://stormchecker.codeplex.com/)
    - Unsound analysis for finding bugs (uses Z3)
  - FUSION (from NEC)

SMT solvers

- A plethora of satisfiability-modulo-theory solvers
  - Simplify, Yices, Z3, CVC, UCLID
  - SAT solvers: zChaff, MiniSAT,…

  - Core technology in several engines
  - Eg. Z3 is used in SDV,PREfix, PEX, SAGE, Yogi, Spec#, VCC, HAVOC, SpecExplorer, FORMULA, F7, M3, VS3, …

Project

- Pick a problem
  - E.g. security properties using information flow for web apps, verification of concurrent lock-free data-structures, etc.
- Pick a technique
  - E.g. constraint-based testing, predicate abstraction,…
- Adapt technique to solve the problem
  - Theory+Implementation: A practical prototype implementing a solution using the technique
  - Theory: a fundamental problem that needs to be addressed theoretically (but practically motivated to solve the problem)
  - Idea should be of potentially publishable :

    i.e. publishable *if* it works well in practice

    (or leads to an interesting theory)

Logistics

- Course website + newsgroup
- Recommend attending and presenting work in FM seminar
- HWs (about 4-5 sets)
- Grades will depend on presentation and project
- Projects will be “reviewed” by other people in the group.

- Questions?