Formal software development methods

Madhusudan Parthasarathy
(madhu)

Lecture #1: Introduction

Motivation

Software validity is one of the main open problems in computer science.

- Bugs have been there forever.
  You can't wish them away
  (like pointer-arithmetic or goto-s)
- Software verification is almost entirely fuelled by ideas from academia.
  - *Theory* makes so much sense
  - Heuristics need theoretical basis to work.
    Blind heuristics simply don't work.

Some bugs

- Ariane V Crash ('96)
  - 64bit -> 16 bit conversion
- Pentium FDIV bug ('97)
  - lookup table had mistakes
- Mars Orbiter
  - feet-per-second -> Newtons-per-second
- Therac-25
  - Radiation therapy machine overdoses patients

Blue screen of death!

Presentation of a Windows 98 beta by Bill Gates at COMDEX on April 20, 1998.

Some bugs…

- In 2006, Windows crashed while Gates was giving a demo of Microsoft Media Center at CES ---

  “blue screen of death”

- Windows 2000 is the chosen OS for the new

  Type 45 Destroyers of the British Navy.

- Plan to fit it to Vanguard class boats that carry the UK's Trident thermo-nuclear intercontinental ballistic missiles.
- **“***Windows for Warships safe for Royal Navy***” ---** says MoD

Microsoft Powerpoint EULA Point 11

- **11**. EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, **IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER** (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) **ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT**, THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS EULA, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, BREACH OF CONTRACT OR BREACH OF WARRANTY OF MICROSOFT OR ANY SUPPLIER, AND EVEN IF MICROSOFT OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The GPL

- **11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. **THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU**. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- **12. IN NO EVENT** UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING **WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM** AS PERMITTED ABOVE, **BE LIABLE TO YOU FOR DAMAGES**, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES **ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM** (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Goal

- Make software reliable
- Software industry---Automobile industry
- What is the notion of certification for

  software ? Compare to airline industry.

  In the future, will software be *required*

  to meet certain requirements (like being

  certified by a model-checker) before being

  used for certain applications?

Goal

- What will JAVA 2010 look like?
  - More features for program annotation
  - Built-in tools that can validate/model-check properties and give feedback to programmer
  - Programming styles that can help automate verification

Review

- ’70s -- proving programs correct
  - Hoare, Djikstra
  - Program invariants
  - Belief: programmers will eventually write programs and prove them correct with help of theorem provers.
  - Failure
    - No way to find bugs; heavily manual; unaccepted in industry (and this class!)
    - Fails for large systems

Review

SPIN (Holzmann, Bell Labs, ’90s)

- Explicit-state model checker
- Heuristics to control state-space explosion
  - Partial order reduction
  - Hashing and approximate search
  - Specification: LTL / automata

Review

Focus on hardware verification

SMV (McMillan, Clarke, CMU, '80s)

- Symbolic model checker using binary decision diagrams (BDDs)
- Could handle large state spaces
  - Heuristics to handle search spaces well
  - Specification: CTL (and later LTL)
  - by far the most useful technique in the hardware domain

Review

- Advent of SAT tools (2000)
  - zChaff (Princeton)
  - can handle formulas with 100000 vars,
    and millions of clauses!

- The idea of bounded model checking
  - Is there a path of length k that reaches
    an unsafe state?
  - NuSMV and contemporary model checkers use SAT heavily.

Review

- The SLAM tool from Microsoft Research
  Ball and Rajamani, 2000
- Static Driver Verifier (SDV)– distributed tool from Microsoft Research– big breakthough!
- Model-checker that validates device drivers against formal specifications.
- Key ideas
  - Predicate abstraction to boolean programs
  - Algorithms to check pushdown automata
  - State-space exploration of bool pgms using BDDs.
  - Tremendous success; Static Driver Verifier

Review

- The Metal project from Stanford.

  Dawson Engler

- Static analysis to find patterns of bad programming practice in systems code.

- Very successful in terms of errors found
  - 100s of bugs (incl security) found in Linux/BSD
  - Errors in various protocols, drivers.
  - Coverity (2004)

Review

- Light-weight static analysis
  - Pointer chasing
  - Data-dependency analysis
  - Usually explicit-state analysis on CFG
  - Example: Codesurfer (Tom Reps)

Techniques you will learn

- Abstraction into models
  - Abstract Interpretation (Cousot&Cousot ’70s)
  - Predicate abstraction
  - Automatic theorem provers
  - Inference of invariants
  - Shape analysis for handling data structures
  - Separation logic
  - Abstraction refinement
    - Counter-example guided
    - SAT proof guided

Techniques you will learn

- Specification of properties
  - Temporal logics (LTL, CTL), assert, pre-post conditions (FOL)
  - Automata-based techniques to verify LTL

    (Gödel-award winning work!)

- Algorithms to search models
  - Static analysis: Dataflow problems using finite models
  - Finite state models (large!)

    Boolean Decision Diagram (BDDs)
  - Bounded model checking

    SAT techniques
  - Finite-data programs with recursion
  - Algorithms to explore models with recursion

Techniques (you will learn)

- Software verification (infinite domains)
  - Floyd's theory of program verification

    -- reducing whole program verification to normal mathematical theorems
  - Hoare's pre-post condition verification
  - Automatic theorem provers pave the way for some automated verification of software (use of tool Boogie or variant).

  - Automatic software verification using predicate abstraction

Tools you will learn to use

- SMV, NuSMV (originally from CMU)
  - Symbolic Model Verifier ; LTL; CTL; BDDs; SAT solving
- Boogie (or JACK)
  - Invariant based verification using theorem proving
- SAT solver (zCHAFF or minsat)

Gritty details

- Undergrad requirement will be simpler and more straightforward
- Graduate/PhD students will have to do a project near the end of the course
- No midterm exam ; there will be a final exam
- Homework will be given every 2 weeks
  - Homework will consist of pen-paper questions as well as implementation of program models and use of verification tools
- Grading
  - Undergrads: HW: 40% Finals: 60%
  - Grads/PhD: HW 30% ; Project 30% ; Finals: 60%
  - More info on webpage coming soon…..