Software Verification and Testing

Azadeh Farzan

CS410 - Fall 2020

Motivation

- Software Validation: one of the toughest open problems in Computer Science.
- Verification has always been derived by academia
  - very rich theoretical basis
    - logics, algorithms, calculi, ...
  - a lot of room for pragmatism
    - theoretically-motivated heuristics

A List of Known Software Bugs

- Northeast blackout
  - data race error
- Ariane V Crash (1996)
  - 64 bit to 16 bit conversion
- Pentium FDIV bug (1997)
  - lookup table had mistakes
- Mars Orbiter
  - feet-per-second vs. Newtons-per-second

- Therac-25
  - radiation therapy over-radiated patients
- Windows crashed during Gate's presentation in 2006
  - windows is used to control highly sensitive army carriers (including those that carry thermo-nuclear intercontinental ballistic missiles).

Newer Bugs

- The Heartbleed bug
  - Random generator error
- Loads of airline outages
  - British Airways (the most recent)
- Loads of news about security breeches
  - Spectre: most famous

What kind of certification do we get for software these days?

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

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.

My favourite part of "The Good Omens"

... along with the standard computer warranty agreement which said that if the machine 1) didn't work, 2) didn't do what the expensive advertisements said, 3) electrocuted the immediate neighborhood, 4) and in fact failed entirely to be inside the expensive box when you opened it.

... this was expressly, absolutely, implicitly and in no event the fault or responsibility of the manufacturer, that the purchaser should consider himself lucky to be allowed to give his money to the manufacturer, and that any attempt to treat what had just been paid for as the purchaser's own property would result in the attentions of serious men with menacing briefcases and very thin watches.

Our Holy Grail

- Make software (more) reliable
  - Software is a product!
    - needs industry standards.
      - A notion of certification for Software
- Meanwhile … make it more reliable
  - partial validation, intelligent testing, …
  - Next generation languages with better validation support.

What is Verification Anyway?

- Proving (in a formal way) that program satisfies a specification written in a logical language.
  - Formal models for programs.
  - Logics for specifications.
  - Algorithms for checking the model against the specification.

Extended Example:

Greatest Common Divisor

in Dafny

Remind Yourself of CSC236 program correctness material!

Program Correctness

- Partial Correctness
  - Pre/Postconditions: formal specification
    - Every terminating execution of the program satisfies the specification.
- Total Correctness
  - partial correctness + proof of program termination

Practical Relevance

What is the point?

Watch this talk!

And, this one if your interest was piqued .

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 formul 0000 variables and millions of cla
- Boosted t Bounded Model Checking (BMC)
  - d other more contemporary model ers

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 derivers against formal spec.
  - Key ideas: predicate abstraction, algorithms for pushdown automata, BDDs for boolean programs.

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
  - Rosette: a program synthesis tool

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~~
  Dataflow Analyses

End of Intro.

Your part

- Read the assigned reading
- Consult all the resources listed
- Do the work
  - It is cliche but: you will get as much as you put into the course.

Text Books, Aids, ...

- No official Text
  - A list of helpful references are posted on the course webpage
- Four TAs
  - They will do most tutorials for you and partially help you use the tools and help you with problem solving.

Prerequisites

- 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 proofs or 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 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
partially by design
and, partially 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!

As in the real word, you will be only compared to your peers.