EECS 219C: Computer-Aided Verification
Introduction & Overview

Sanjit A. Seshia
EECS, UC Berkeley
Computer-Aided Verification (informally)

Does the system do what it is supposed to do?
The End User’s Perspective

Does the system do what it is supposed to do?
The Engineer’s Perspective

Does the implemented system meet its specifications?
The Mathematician’s Perspective

Prove or disprove (verify) that the mathematical model of the system satisfies a mathematical specification.

\[ x(t) = f(x(t), u(t)) \]
Formal Methods

Rigorous mathematical / algorithmic techniques for specification, design, verification and maintenance of computational systems.

The essence: It’s about PROOF

• Specify proof obligations
• Prove that system meets those obligations
• Synthesize provably-correct system
What we’ll do today

• Introductions: to Sanjit and others
• Brief Intro. to the main course topics
  – Motivation
  – Temporal Logic, Model Checking, SAT, and Satisfiability Modulo Theories (SMT)
  – History, Opportunities, Challenges
• Course Logistics
My Research


Theory
Computational Logic, Algorithms, Learning Theory, Optimization

Practice
CAD for VLSI/Bio, Computer Security, Embedded Systems, Education

Example: Learning+Verification for Auto-Grading Lab-based Courses
Class Introductions

Please introduce yourselves
-- state name and research interests/areas
(Programming Systems, Computer Security, CAD, Embedded Systems, Synthetic Biology, Control Theory, etc.)
Computer-Aided Verification

• Automatically verifying the correctness of systems

Verifier

System
Environment
Property

B

Yes (system correct)
/  no (here’s a bug)

• Questions for today:
  – Is it relevant?
  – Is it feasible?
  – What will we study?
Ariane disaster, 1996
$500 million software failure

FDIV error, 1994
$500 million

Estimated worst-case worm cost:
> $50 billion

Toyoda Recalls 1.9 Million Prius Hybrids
Over Software Flaw

By Jeremy Hsu
Posted 12 Feb 2014 | 21:55 GMT

Bugs cost Time, Money, Lives, …

S. A. Seshia
An Example from Embedded/Cyber-Physical Systems

Medical devices run on software too… software defects can have life-threatening consequences.

[Journal of Pacing and Clinical Electrophysiology, 2004]

“the patient collapsed while walking towards the cashier after refueling his car […] A week later the patient complained to his physician about an increasing feeling of unwell-being since the fall.”

“In 1 of every 12,000 settings, the software can cause an error in the programming resulting in the possibility of producing paced rates up to 185 beats/min.”
“It’s an Area with a Pessimistic View!”
No, not really.

• The theory underlying algorithmic verification is beautiful
• It’s about the notion of PROOF
• It’s interdisciplinary
• The implementations are often non-trivial
  – Scaling up needs careful hacking
• It’s fun to work on!
Is Verification Feasible?

• “Easiest” non-trivial verification problem is NP-hard (SAT)

• But the outlook for practice is less gloomy than for theory…
  – More hardware resources
  – Better algorithms
My Experience with SAT Solving

Speed-up of 2012 solver over other solvers

Solver


Speed-up (log scale)
Experience with SPIN Model Checker

[G. Holzmann]

Some algorithmic improvements in the last two decades

Memory (Megabytes)

10000

1000

100

10

1


A sample verification problem from 1980
tpc – a logic model of a telephone switch
Topics in this Course

• Computational Engines
  – Boolean satisfiability (SAT)
  – Satisfiability modulo theories (SMT)
  – Model checking
  – Syntax-guided synthesis (SyGuS)

• Advanced Topics (“Research Frontiers”)
  – Quantitative/Probabilistic verification
  – Deduction + Inductive Learning
  – Synthesis from multi-modal specifications
  – Human-Computer Interaction & Verification
  – New application domains
  – … (more later in this lecture)
Topics of this Course (another view)

Application Domains

Circuits, Software, Networks, Hybrid Systems, Biological Systems, etc.

Verification Strategies

Automata-theoretic, Symbolic, Abstraction, Learning, etc.

Computational Engines

SAT, BDDs, SMT
Boolean Satisfiability (SAT)

Is there an assignment to the $p_i$ variables s.t. $\phi$ evaluates to 1?
Two Applications of SAT

• Equivalence checking of circuits
  – Given an initial (unoptimized) Boolean circuit and its optimized version, are the two circuits equivalent?
  – Standard industry CAD problem
• Malware detection (security)
  – Given a known malicious program and a potentially malicious program, are these “equivalent”?
• Many other applications:
  – Cryptanalysis, test generation, model checking, logic synthesis, ….
Satisfiability Modulo Theories (SMT)

Is there an assignment to the $x, y, z, w$ variables s.t. $\phi$ evaluates to 1?
Applications of SMT

• Pretty much everywhere SAT is used
  – The original problem usually has richer types than just Booleans!

• To date: especially effective in
  – software model checking
  – test generation
  – software synthesis
  – finding security vulnerabilities
  – high-level (RTL and above) hardware verification
Model Checking

• Broad Defn:
  A collection of algorithmic methods based on state space exploration used to verify if a system satisfies a formal specification.

• Original Defn: (Clarke)
  A technique to check if a finite-state system is a model of (satisfies) a temporal logic property.
Visualizing Model Checking

"Two trains, one bridge"-UML model transformed with Hugo

S. A. Seshia

[Moritz Hammer, Uni. Muenchen]
Model Checking, (Over)Simplified

• Model checking “is” graph traversal?
• What makes it interesting:
  – The graph can be HUGE (possibly infinite)
  – Nodes can represent many states (possibly infinitely many)
  – How do we generate this graph from a system description (like source code)?
  – Behaviors/Properties can be complicated (e.g. temporal logic)
  – …
A Brief History of Model Checking

• 1977: Pnueli introduces use of (linear) temporal logic for specifying program properties over time [1996 Turing Award]

• 1981: Model checking introduced by Clarke & Emerson and Quielle & Sifakis
  – Based on explicitly traversing the graph
  – capacity limited by “state explosion”

• 1986: Vardi & Wolper introduce “automata-theoretic” framework for model checking
  – Late 80s: Kurshan develops automata-theoretic verifier

• Early - mid 80s: Gerard Holzmann starts work on the SPIN model checker
A Brief History of Model Checking

• 1986: Bryant publishes paper on BDDs
• 1987: McMillan comes up with idea for “Symbolic Model Checking” (using BDDs) – SMV system
  – First step towards tackling state explosion
• 1987-1999: Flurry of activity on finite-state model checking with BDDs, lots of progress using: abstraction, compositional reasoning, …
  – More techniques to tackle state explosion
• 1990-95: Timed Automata introduced by Alur & Dill, model checking algorithms introduced; generalized to Hybrid Automata by Alur, Henzinger and others
A Brief History of Model Checking

• 1999: Clarke et al. introduce “Bounded Model Checking” using SAT
  – SAT solvers start getting much faster
  – BMC found very useful for debugging hardware systems

• 1999: Model checking hardware systems (at Boolean level) enters industrial use
  – IBM RuleBase, Synopsys Magellan, 0-In FV, Jasper JasperGold

• 1999-2004: Model checking + theorem proving: software and high-level hardware comes of age
  – SLAM project at MSR, SAL at SRI, UCLID at CMU
  – Decision procedures (SMT solvers) get much faster
  – Software verifiers: Blast, CMC, Bandera, MOPS, …
  – SLAM becomes a Microsoft product “Static Driver Verifier”
A Brief History of Model Checking

• 2005-date: Model Checking is part of the standard industrial flow. Some new techniques and applications arise:
  – Combination with simulation (hardware) and static analysis/testing (software) [Many univ/industry groups]
  – Checking for termination in software [Microsoft]
  – Program synthesis [Berkeley, Microsoft, MIT, Penn, …]
  – Lots of progress in verification of concurrent software [Microsoft CHESS project]
• Clarke, Emerson, Sifakis get ACM Turing Award; SAT solving advances are recognized

WHAT’S NEXT?!
Research Frontiers in Formal Verification

• Three Themes:
  – New Demands on Computational Engines
  – New Applications
  – The “Human Aspect”
    • Steps that require significant human input
    • Systems with humans in the loop

→ suggested project topics next week
Formal Methods for Education

Goal: To enable personalized learning for lab-based courses in science and engineering → CPSGrader, deployed on edX and on campus
Formal Methods for Robotics

**Goal:** To synthesize motion plans automatically for a group of robots with complex dynamics for *Linear Temporal Logic (LTL)* specification

**Tool:** ComPlan

[VIDEO](https://www.youtube.com/watch?v=pSjGwhH29Zs)
Formal Methods for Networking

Networks Tomorrow

• Online services $\rightarrow$ latency, cost sensitive
• Merchant Silicon $\rightarrow$ Build your own router
• Rise of Data centers $\rightarrow$ Custom networks
• Software defined Networks (SDNs) $\rightarrow$ custom design “routing program”
• P4 (next generation SDN) $\rightarrow$ redefine hardware forwarding at runtime

Opportunity to custom design networks to optimize goal.
Potential simplifications but complex interactions, hard to get right
Digital Hardware Design as Inspiration?

Electronic Design Automation
(McKeown SIGCOMM 2012)

Network Design Automation
(NDA)?
Course Logistics

• Check out the webpage:
  www.eecs.berkeley.edu/~sseshia/219c

• Tentative class schedule will be up soon
  – 2007 Turing Award lecture (optional viewing)
  – IMP: Think about project topics!
Course Outline

• 2 parts
• Part I: Model Checking, Boolean reasoning (SAT, BDDs), SMT
  – Basics, how to use these techniques, and how to extend them further
• Part II: Advanced Topics
  – The challenging problems that remain to be addressed
Reference Books

• See list on the website
• Copies will be on reserve at Engg Liby
• e-Handouts for most material
Grading

• Scribing lectures (20%)
  – 2 lectures per person: Scribe one lecture, edit another lecture
  – Sign-up sheet next week

• Homework (10%)
  – First part of the course

• Paper discussions / class participation (10%)
  – Last month of the course

• Project (60%)
  – Do original research, theoretical or applied
  – Sample topics will be announced by end of next week
  – Project proposal due mid Feb.
  – Culminates in final presentation + written paper
  – ~50% of past projects led to conference papers!
Misc.

• Office hours: W 1:30 – 2:30, and by appointment
• Pre-requisites: check webpage; come talk to me if unsure about taking the course
  – Undergraduates need special permission to take this class