EECS 219C:
Computer-Aided Verification

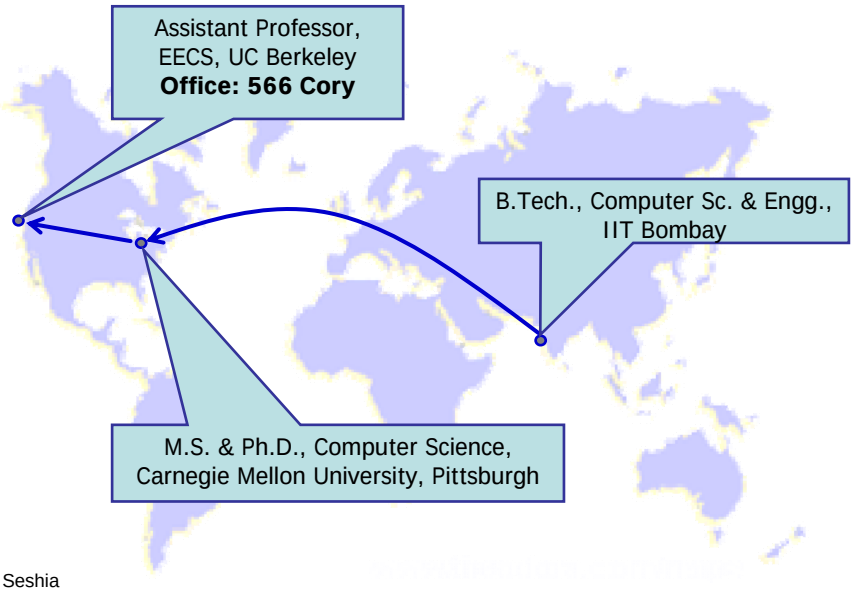
Introduction & Overview

Sanjit A. Seshia
EECS, UC Berkeley

What we'll do today

- Introductions: to Sanjit and others
- Brief Intro. to Model Checking, SAT, and Satisfiability Modulo Theories (SMT)
  - History, Opportunities, Challenges
- Course Logistics

About Me



My Research

"Algorithms for Dependable Computing"

+

Theory

Computational Logic,
Algorithms,
Learning Theory,
Optimization

Practice

CAD for VLSI,
Computer Security,
Embedded Systems,
Program Analysis

Example: Game-theoretic online learning used to estimate worst-case execution time of a program

Class Introductions

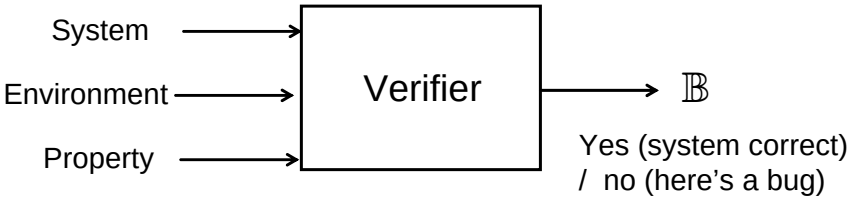
Please introduce yourselves

-- state name and research area(s)

(CAD, Embedded Systems, Control Theory, Programming Systems, etc.)

Computer-Aided Verification

- Automatically verifying the correctness of systems



- Questions for today:
  - Is it relevant?
  - Is it feasible?
  - What will we study?


```
<msblast.exe> (the primary executable of the exploit)
I just want to say LOVE YOU SAN!!
billy gates why do you make this possible ? Stop
making money and fix your software!!
windowsupdate.com
start %s
tftp -i %s GET %s
%d.%d.%d.%d
```

Estimated worst-case worm cost: > $50 billion

Bugs cost Time and Money

- Cost of buggy software estimated to range $22 Billion - $ 60 B / year [NIST, 2002]
- Verification takes up 70% of hardware design cycle
- Post-silicon validation & debugging accounts for ~ 1/3rd of design cost

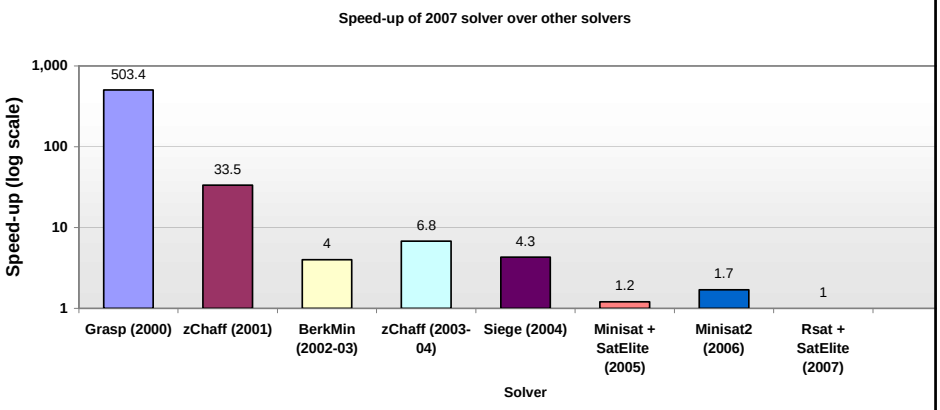
"It's an Area with a Pessimistic View!" No, not really.

- The theory underlying algorithmic verification is beautiful
- It's interdisciplinary
- The implementations are often non-trivial
  - Scaling up needs careful hacking
- It's fun to work on!
- Analogy: coding theory is also about dealing with errors in data transmisson, storage, etc., but it's really interesting theory!

Is Verification Feasible?

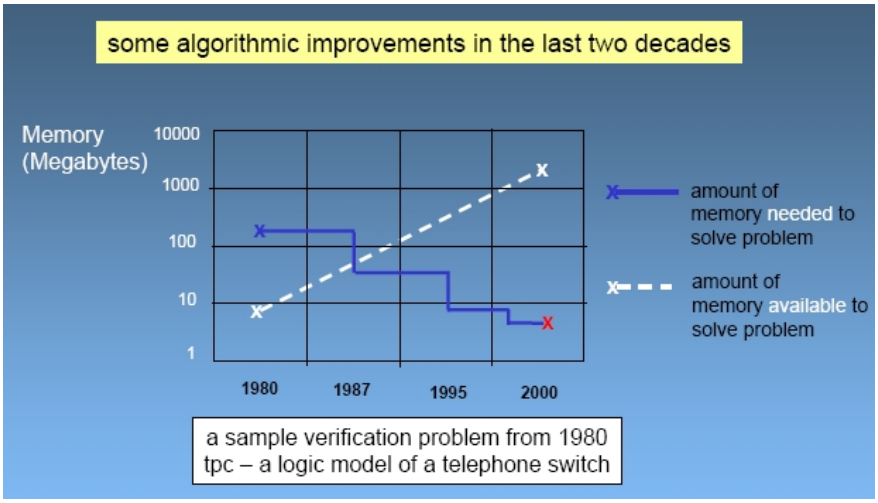
- Easiest non-trivial verification problem is NP-hard (SAT)
- But the outlook for practice is less gloomy than for theory…
  - More hardware resources
  - Better algorithms

My Experience with SAT Solving



Experience with SPIN Model Checker

[G. Holzmann]



Topics in this Course

- Fundamental Algorithmic Techniques
  - Boolean satisfiability (SAT)
  - Satisfiability modulo theories (SMT)
  - Model checking
- Advanced Topics ("Research Frontiers")
  - Quantitative verification
  - Probabilistic verification
  - Verification + Learning (dealing with environment uncertainty)
  - Synthesis from specifications
  - … (more later in this lecture)

Topics of this Course (another view)

Application Domains

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

Verification Strategies

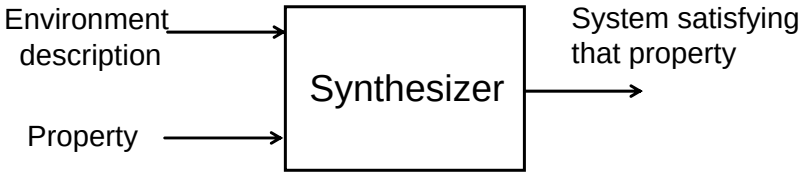
Automata-theoretic, Symbolic, Abstraction, Interpolation, etc.

Computational Engines

SAT, BDDs, SMT

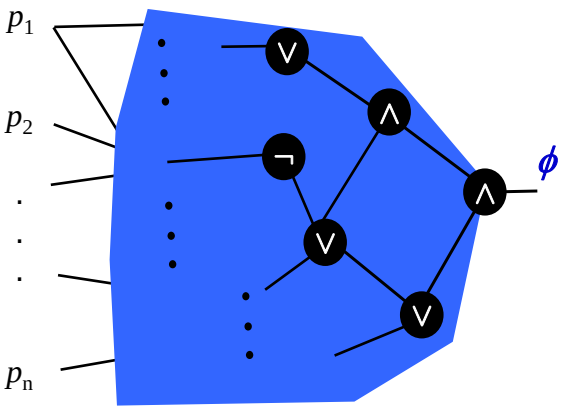
A Theme of this Course

- Investigating the Interplay between Verification and Synthesis



- Verification methods invoke synthesizers
  - Synthesizing an 'inductive invariant' or lemma
- Synthesis tools rely on verifiers

Boolean Satisfiability (SAT)

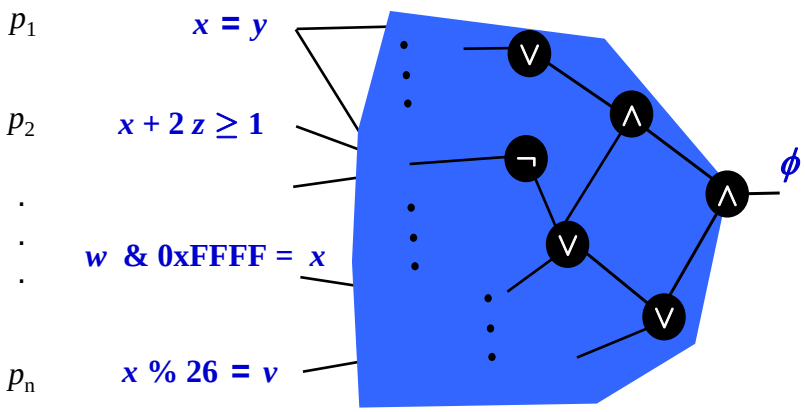


Is there an assignment to the p_i variables s.t. ϕ evaluates to 1?

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, synthesis, ….

Satisfiability Modulo Theories (SMT)



Is there an assignment to the x,y,z,w variables s.t. ϕ 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
  - finding security vulnerabilities
  - high-level (RTL and above) hardware verification
- A course goal: find new applications!

Model Checking

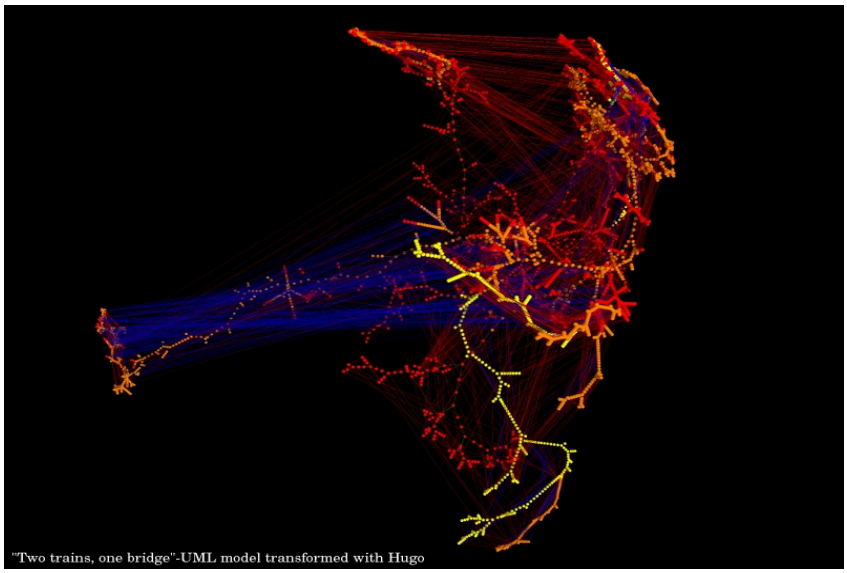
- Broad Defn:

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

- Original Defn:

  A technique to check if a finite-state system is a model of (satisfies) a temporal logic property.

Visualizing Model Checking



[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 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 software and high-level hardware designs 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

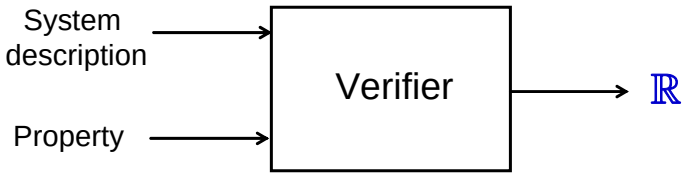
- 2005-date: Model Checking is part of the standard industrial flow. Some new techniques and applications arise:
  - Combination with simulation (hardware) and static analysis/testing (software) [Many univ/industry groups]
  - Checking for termination in software [Microsoft]
  - Evaluating fault-tolerance of circuits & software to device faults (soft errors) [Berkeley, UIUC]
  - Lots of progress in verification of concurrent software [Microsoft CHESS project]
- Clarke, Emerson, Sifakis get 2008 ACM Turing Award; Designers of Grasp/Chaff solvers get 2009 CAV Award

WHAT'S NEXT?!

Research Frontiers in Formal Verification

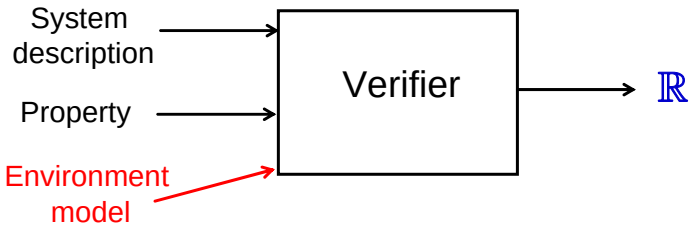
- We will go over several "challenge areas" for formal verification
- These will form the foci for projects and paper presentations

Challenge #1: Quantitative/Probabilistic Verification



Examples:
- How long does this program take, in the worst-case?
- What is the mean time to failure of this system?
- How many bits of secret data does this program leak?

Challenge #1: Quantitative/Probabilistic Verification



Example:

• How long does this program take, in the worst-case?

Depends on the processor, OS, other programs in the system, network, I/O devices, etc…

→ How can we accurately/automatically model these?

Challenge #2: Verifying Large-Scale Networked Systems

- Hardware: "Many-core" designs
  - Potentially have 100s of cores connected through a large on-chip network
- Software:
  - Data-center applications
- Embedded:
  - large-scale sensor networks, electric grid network, …

➢ **How do we verify properties that only fail at large-scale?**

- ➢ **verification typically only works for small number of nodes**
- ➢ **Need new techniques for automatic abstraction**

Challenge #3: Verification → Diagnosis & Repair

- Suppose a model checker reports an error trace.
- Work doesn't stop there! We need to perform
  - Diagnosis: Where is the error?
  - Repair: How best to fix it?
- Diagnosis is also the main component of post-silicon validation

Challenge #4: Verification → Synthesis

- Can we adapt verification techniques to synthesis a design from specifications?
  - E.g. synthesis from temporal logic
- Major challenges:
  - Computational cost
  - Difficulty of writing formal specifications
- Ideas: Combine formal techniques with machine learning (see ICSE'10 and CPS'10 papers on my webpage)

Challenge #5: Infinite-State Systems, Rich Data

- Model checking has been very effective for systems with Boolean state and control-intensive structure
  - Finite-state systems, pushdown systems
- We still need more work for: *Real-time and Hybrid systems, Analog/Mixed-signal circuits, Rich data (e.g. strings)*
- Ideas: New SMT solvers for strings and non-linear arithmetic, combining numerical simulation with traditional formal verification methods, new abstraction methods

Challenge #6: Verification of Interactive Systems

- Many safety-critical systems interact with humans
  - Avionics (pilots), medical devices (doctors, patients), …
- How do we verify that these interactive systems work correctly?
  - E.g. mode confusion in avionics: pilot thinks the system is in Mode A whereas it is in a different Mode B
- Idea: Combine formal verification with systematic testing by humans
  - E.g. Verifying an Electronic Voting Machine (recent CCS'09 paper available off my webpage)

There are other challenges as well.

We will look at some of these in the second half of the course, and I encourage your projects to address these.

Topics in Verification that we **won't** study in much depth

- Equivalence checking and synthesis of digital circuits [219B – Kuehlmann]
- Software Testing [265 – Sen]
- Classic Program Verification (e.g., Hoare logic, abstract interpretation) [263 – Necula]
- Numerical Simulation [219A – Roychowdhury]

Course Logistics

- Check out the webpage:
  www.eecs.berkeley.edu/~sseshia/219c
- Detailed schedule will be up this week

Course Outline

- 2 parts
- Part I: Model Checking, Boolean reasoning (SAT, BDDs), SMT
  - How to use these techniques, and how to extend/develop them further
- Part II: Advanced Topics
  - The challenging problems that remain to be addressed
  - Special focus on synthesis from specifications

Reference Books

- Two recommended books:
  - "Model Checking" by Clarke, Grumberg, Peled
    - Good reference book if you intend to work in model checking or related area
  - "Logic in Computer Science" by Huth & Ryan
    - Useful especially if you lack some background
- Other reference books listed on website
- Copies will be on reserve at Engg Liby
- e-Handouts for other material

Grading

- 2-3 Homeworks (30%)
  - On the first half of the course
- Paper presentations (20%)
  - Second half of the course
- **Project (50%)**
  - Do original research, theoretical or applied
  - Sample topics will be announced by end of this week
  - Project proposal due mid Feb.
  - Culminates in final presentation + written paper
  - *30-50% of past projects led to conference papers!*

Misc.

- Office hours: M 1 – 2:30 pm, 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