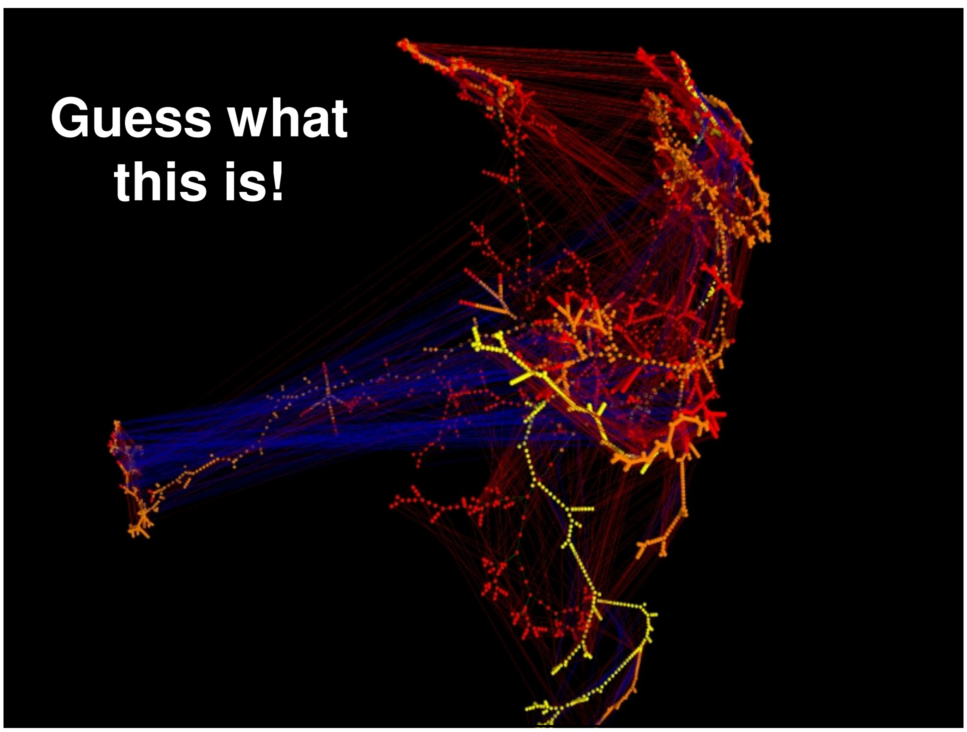
EECS 219C:
Computer-Aided Verification

Introduction & Overview

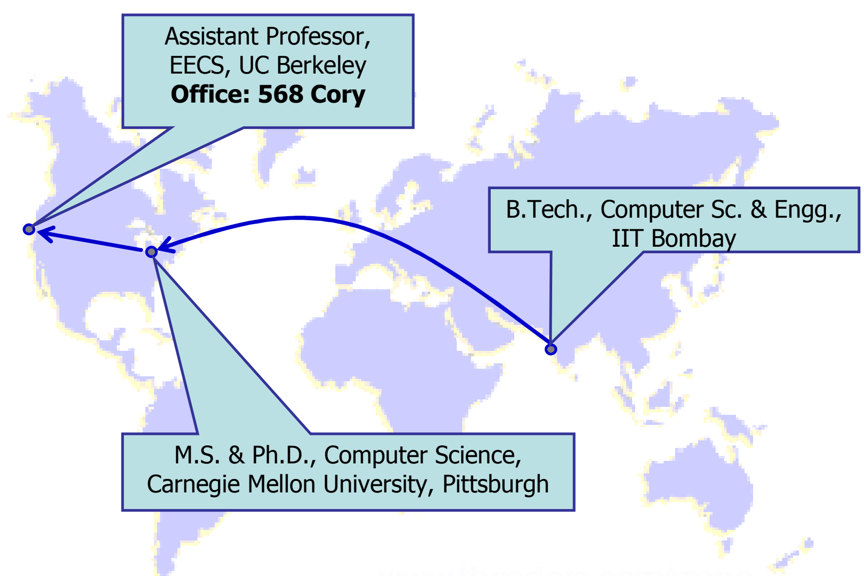
Sanjit A. Seshia
EECS, UC Berkeley



What we'll do today

- Introductions: to Sanjit and others
- Intro. to Model Checking
  - 25 years since the first papers
  - History, Opportunities, Challenges
- Course Logistics & Survey

About Me



My Research

+

Theory

Computational Logic,
Algorithms

Practice

CAD for VLSI,
Computer Security,
Program Analysis,
Dependability

Example: Fast automatic theorem proving used to build a better virus/worm detector

S. A. Seshia 5

Class Introductions

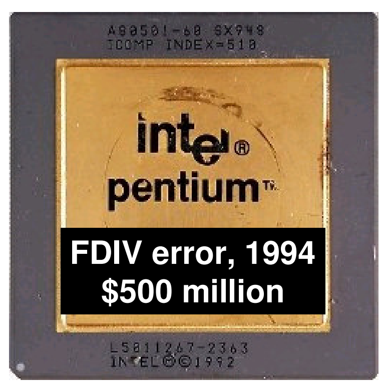
Please introduce yourselves

S. A. Seshia 6

Computer-Aided Verification

- Automatically verifying the correctness of computer systems
- Is it relevant?
- Is it feasible?
- What will we study?

S. A. Seshia 7




```
<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

S. A. Seshia 8

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

S. A. Seshia 9

"Such a Pessimistic View of Life!" No, not really.

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

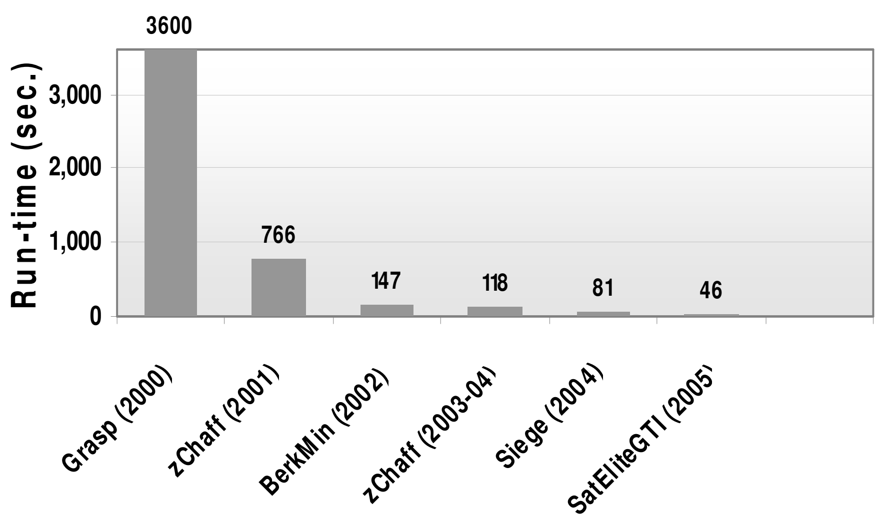
S. A. Seshia 10

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

S. A. Seshia 11

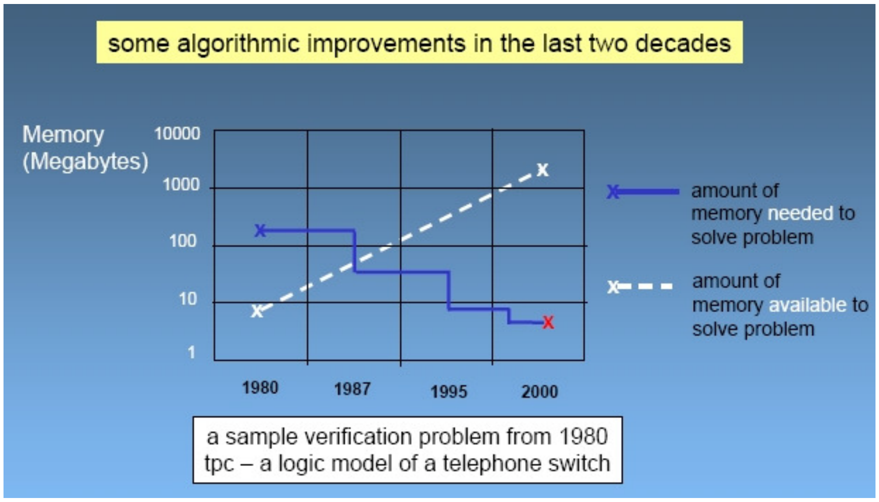
My Experience with SAT Solvers



12

Experience with SPIN Model Checker

[G. Holzmann]



S. A. Seshia 13

What we will study:

Model Checking & Computational Logic

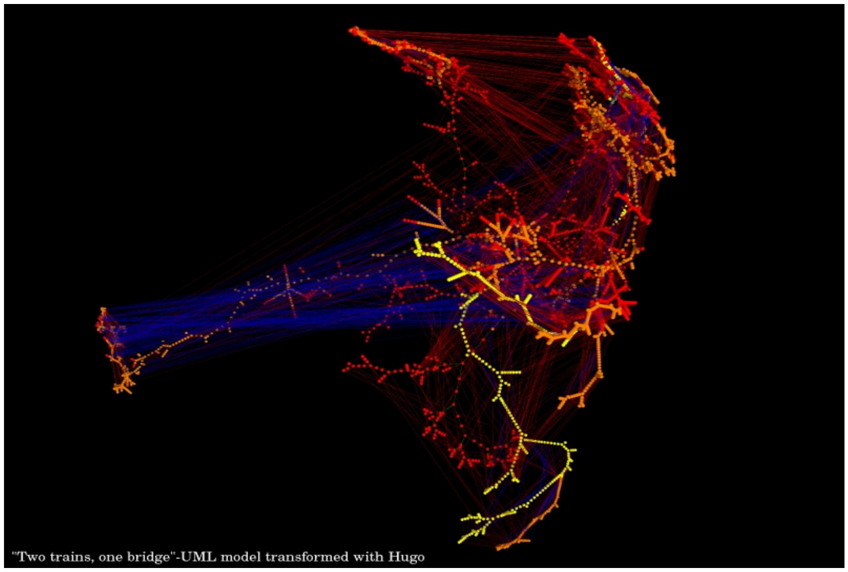
Computational Logic

- **Mathematical logic for reasoning about computation**
  (& computer science for logic)
- Covers many areas, including model checking, and other topics:
  - Constraint Solving
  - Functional Programming & Lambda Calculus
  - Type Theory
  - Logical Aspects of Computational Complexity
- Sample journal:
  ACM Transactions on Computational Logic

Model Checking

A collection of **algorithmic methods** based on **state space exploration** used for **computer-aided verification**.

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
  – …

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 enters industrial use
  - IBM RuleBase, Synopsys Magellan, 0-In FV, Jasper JasperGold
- 1999-2004: Software model checking comes of age
  - Ball & Rajamani start SLAM project at MSR
  - Decision procedures (SMT solvers) get much faster
  - Many projects to date: Blast, CMC, Bandera, MOPS, …
  - SLAM becomes a Microsoft product "Static Driver Verifier"

Research Frontiers in Model Checking

- Last year was the 25th anniversary of the original papers on finite-state model checking
- So there was a party! The 25MC symposium.
- Experts gave their opinion on what the grand challenges are…
- … And I interpreted them ☺
  - These reflect opportunities for impact

Challenge #1: Coverage in Verification

- Suppose the model checker reports that the system is correct.
- Can we really believe it? Why or why not?

Challenge #1: Coverage in Verification

- Suppose the model checker reports that the system is correct.
- Can we really believe it? Why or why not?

Two Issues:

- Verification is only as good as the set of properties you verify
- Model checkers are being used as debuggers. When have we found all bugs? When do we stop model checking?

WE NEED COVERAGE METRICS

Challenge #2: Verification → Reliability

- Verification can only be applied to (small) components of an overall design
- How does that relate to overall system reliability?
  - The real problem is to design reliable systems
  - Can we get a "mean time between failures" number from outputs of formal verification?

S. A. Seshia 25

Challenge #3: Verification → 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 to fix it?

S. A. Seshia 26

Challenge #4: Scalability

- Problems underlying verification are intrinsically hard
  - SAT, QBF, etc.
- How do we scale up?
  - Leverage increasing parallelism in hardware
  - Design "adaptive" algorithms that circumvent worst-case complexity
  - Leverage automated abstraction

"A complex hybrid cocktail of AI techniques will come to bear on model checking" – K. McMillan

Challenge #5: Infinite-State Systems

- Model checking has been very effective for systems with Boolean state
  - Finite-state systems, pushdown systems
- The next frontier: *Real-time and Hybrid Systems*
- Idea: Can we leverage all the work that's been done for Boolean state?

Challenge #6: The Invisible Specification

- We typically assume that a formal specification is given.
- This doesn't usually happen!
- We need techniques for:
  - Making writing and re-using specifications easier for designers/programmers
  - Automatically inferring specifications (from executions or otherwise)

S. A. Seshia 29

Challenge #7: InterOperability

- There are a ton of verification tools available today
  - Each finds bugs in its "niche"
  - Each has its specification & modeling language
- How can we make them operate together, so that one can find bugs/finish where another runs out of gas?

S. A. Seshia 30

Challenge #8: Structuring Code for Verification

- Code (C, Verilog, …) is often written in a way that makes verification difficult
  - E.g., not modular, structure in the code that can help automated abstraction isn't obvious
- How do we write code so that formal verification is easier?

There are many 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 depth

- Equivalence checking of digital circuits [219B – Kuehlmann, 290A – Brayton]
- Software Testing [294 – Sen]
- Topics in Program Verification (e.g., Hoare logic) [263 – Necula]
- Simulation of circuits [219A – Brayton]

Course Logistics

- Check out the webpage:
  www.eecs.berkeley.edu/~sseshia/219c
- Detailed schedule is up

Course Outline

- 3 parts
- Part I: Computational Engines for Model Checking
  - Basics: SAT, BDDs, etc.
- Part II: Foundations of Model Checking – Systems with Boolean State
  - "Classic" model checking (finite-state, also pushdown)
  - More theoretical in content, applies broadly to many areas in EE and CS
- Part III: Research Frontiers
  - The challenging problems that remain to be addressed – this is where the payoff is

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 of all are on reserve at Engg Liby
- Handouts for other material

Grading

- 3 Homeworks (30%)
  – On the first half of the course
- 1 Presentation of advanced topic (15%)
  – Based on papers to be posted on the webpage
- **Project (50%)**
  – Do original research, theoretical or applied
  – Sample topics will be announced by month-end; good to pick something close to your research area
  – Project proposal due mid-Feb.
  – Culminates in final presentation + written paper
  – *Over half of last year's projects turned into / contributed to conference papers!*

S. A. Seshia

[What's missing on this slide? ☺]

37

Misc.

- Office hours: M 2 - 3 pm, W 1 - 2 pm
- Pre-requisites: check webpage; come talk to me if unsure about taking the course
  – Undergraduates need special permission to take this class
- Student background survey

S. A. Seshia

38