

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?**

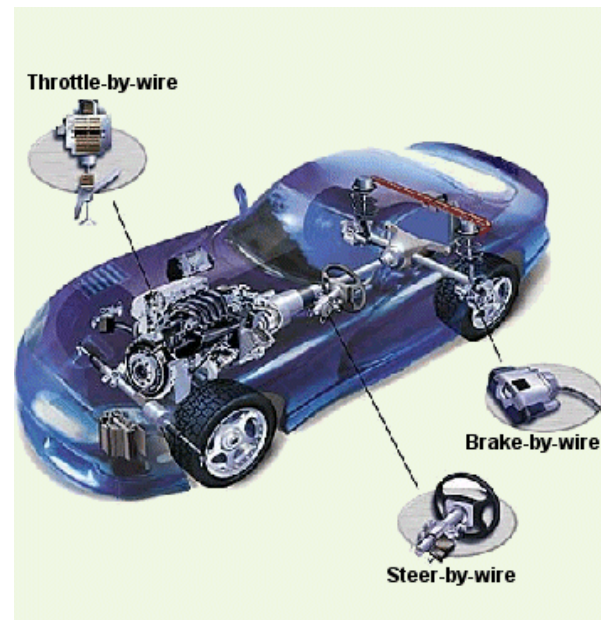
# The End User's Perspective

Does the **system** do what it is supposed to do?



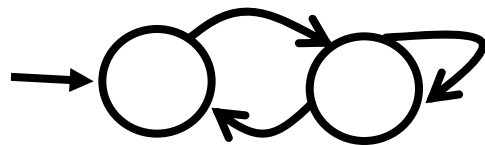
# The Engineer's Perspective

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**

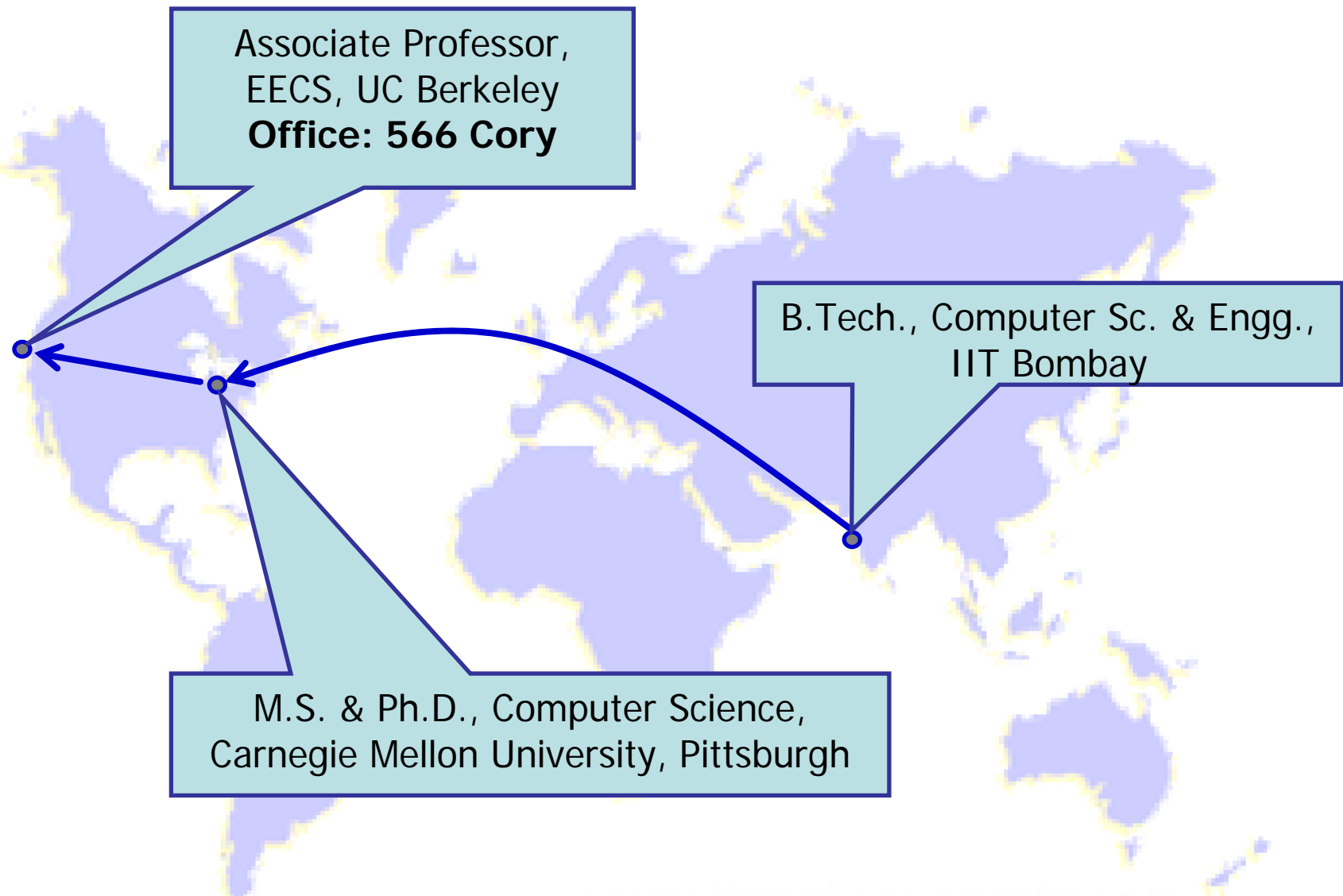


$$\dot{x}(t) = f(x(t), u(t))$$

# What we'll do today

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

# About Me



Associate Professor,  
EECS, UC Berkeley  
**Office: 566 Cory**

B.Tech., Computer Sc. & Engg.,  
IIT Bombay

M.S. & Ph.D., Computer Science,  
Carnegie Mellon University, Pittsburgh

# My Research

“Formal Methods: Algorithms & Modeling for Dependable Computing”



**Theory**

Computational Logic,  
Algorithms,  
Learning Theory,  
Optimization

+



**Practice**

CAD for VLSI,  
Computer Security,  
Embedded Systems,  
Software Engineering

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



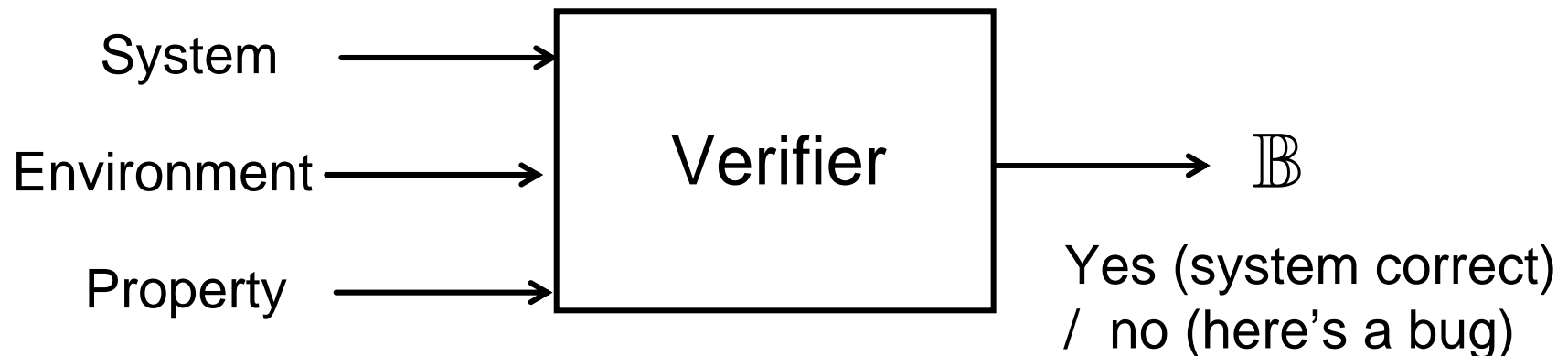
# Class Introductions

Please introduce yourselves

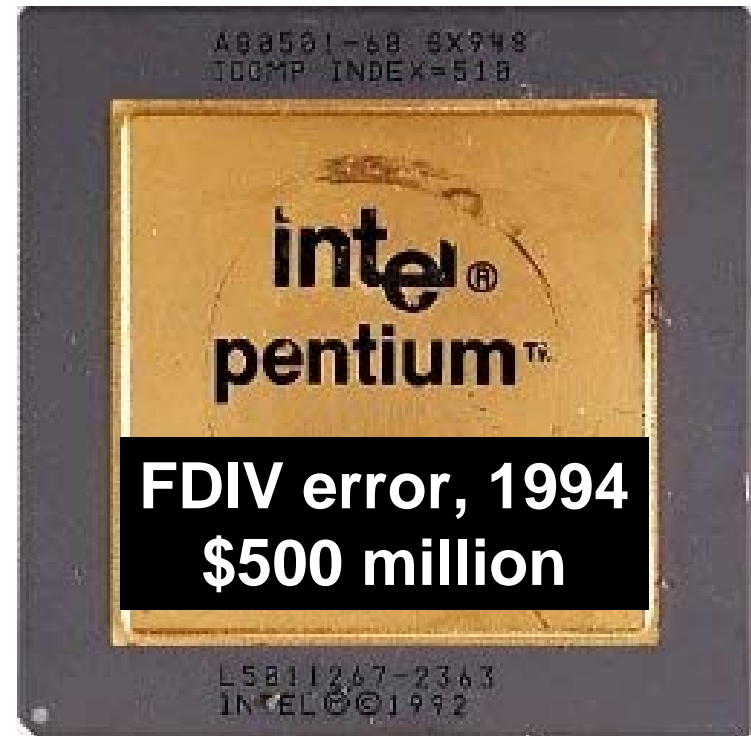
-- state name and research interests/areas  
(Programming Systems, Computer Security,  
CAD, Embedded Systems, Synthetic  
Biology, Control Theory, etc.)

# Computer-Aided Verification

- Automatically verifying the correctness of systems



- 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**

# 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]



[different device]

“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.**”

# 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  $\sim 1/3^{\text{rd}}$  of design cost

# “It’s an Area with a Pessimistic View!”

## No, not really.

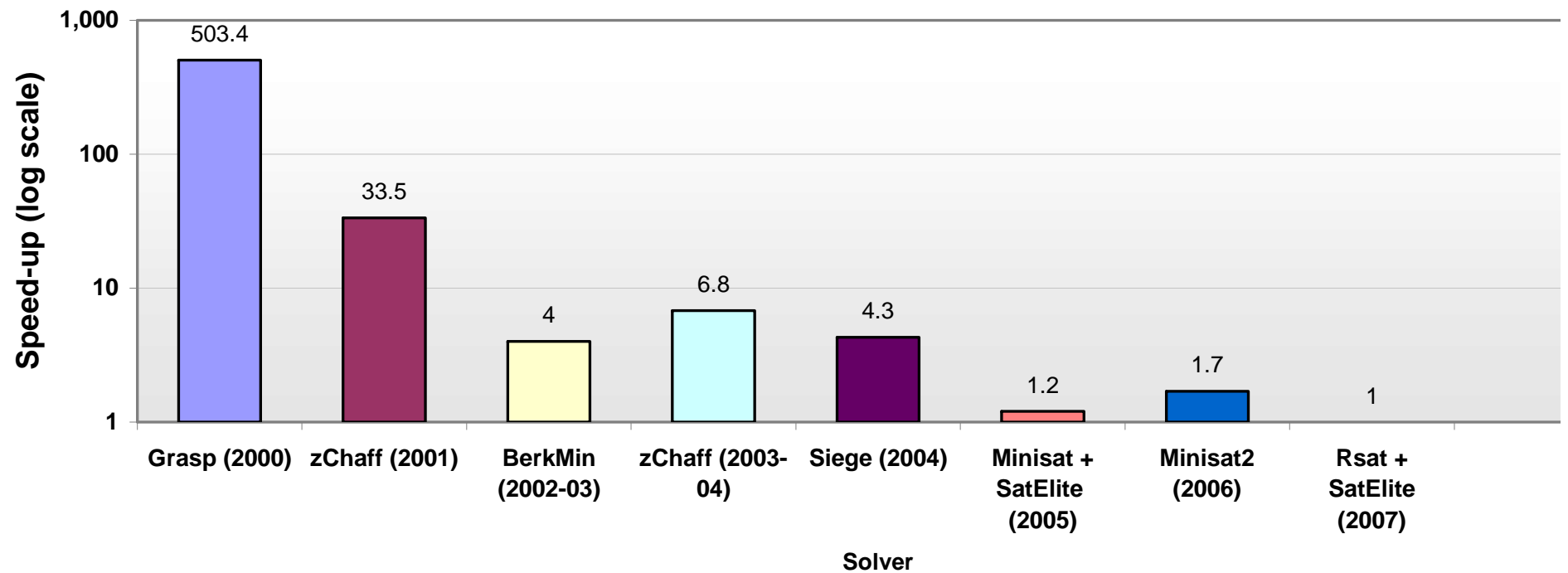
- The theory underlying algorithmic verification is beautiful
- It’s interdisciplinary
- The implementations are often non-trivial
  - Scaling up needs careful hacking
- It’s fun to work on!
- Analogy: coding theory is also about dealing with errors in data transmission, storage, etc., but it’s really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

Speed-up of 2007 solver over other solvers

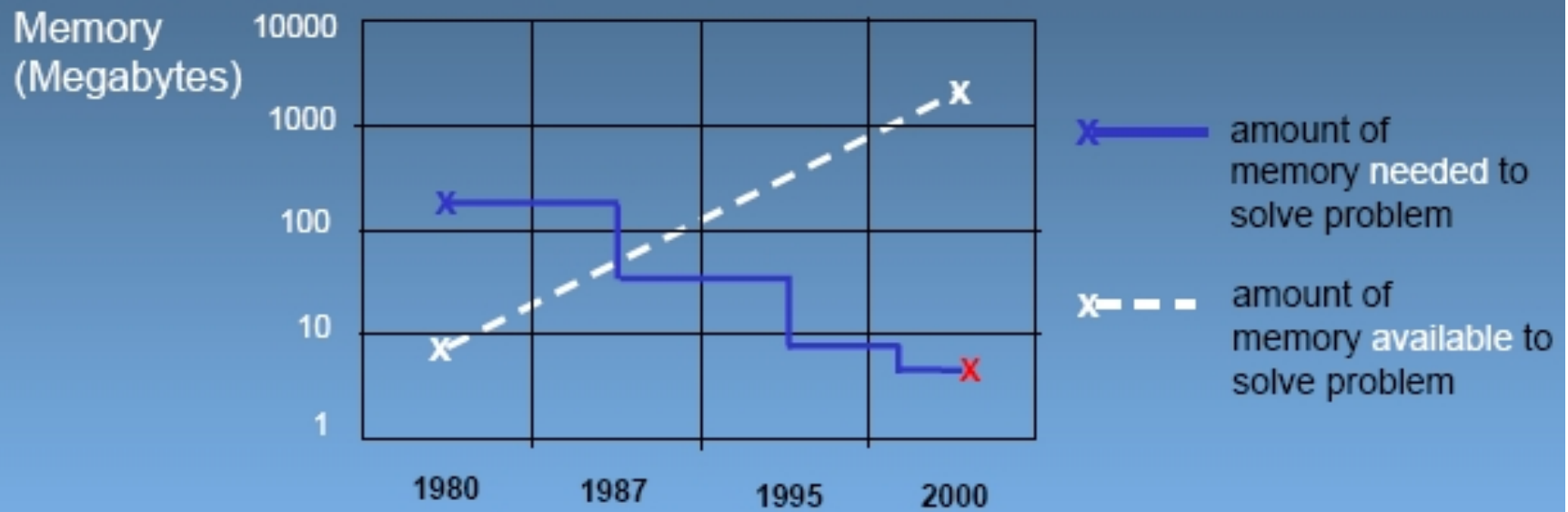




# Experience with SPIN Model Checker

[G. Holzmann]

some algorithmic improvements in the last two decades



a sample verification problem from 1980  
tpc – a logic model of a telephone switch

# Topics in this Course

- Fundamental Algorithmic Techniques
  - Boolean satisfiability (SAT)
  - Satisfiability modulo theories (SMT)
  - Model checking
- Advanced Topics (“Research Frontiers”)
  - Quantitative verification
  - Deduction + Inductive Learning
  - Synthesis from specifications
  - Human-Computer Interaction & Verification
  - ... (more later in this lecture)

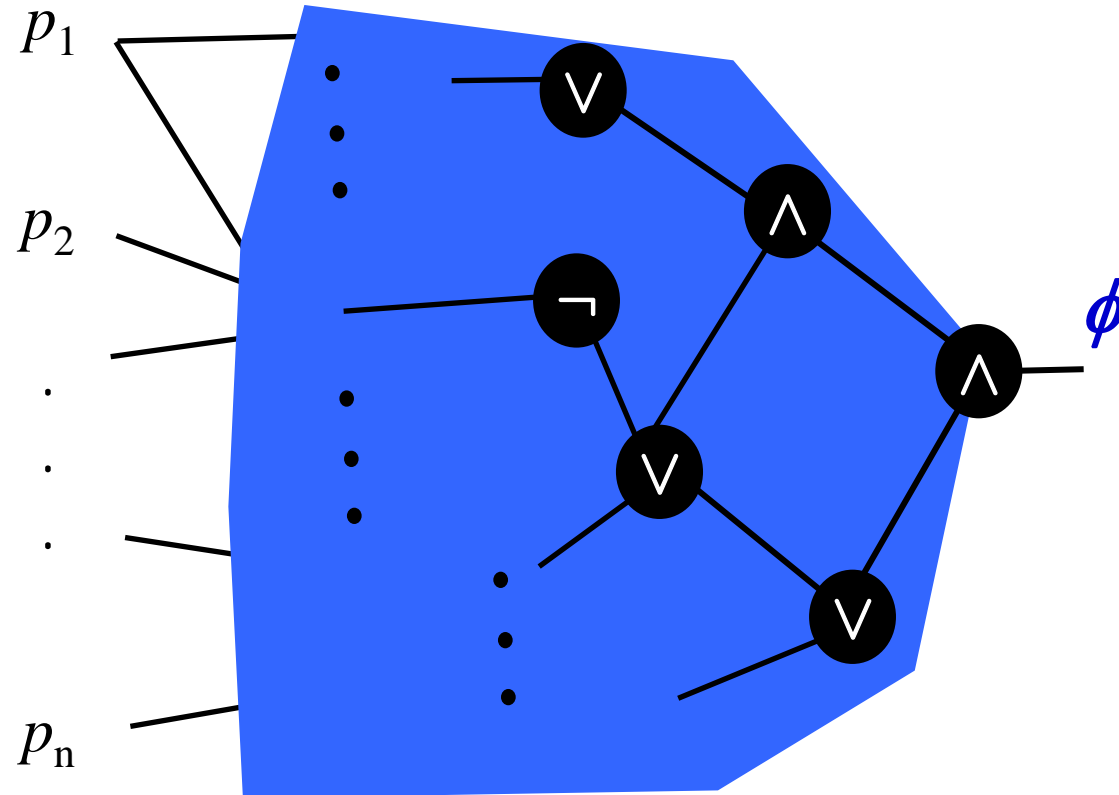
# Topics of this Course (another view)

Application Domains  
**Circuits, Software, Hybrid Systems,  
Biological Systems, etc.**

Verification Strategies  
**Automata-theoretic, Symbolic,  
Abstraction, Learning, etc.**

Computational Engines  
**SAT, BDDs, SMT**

# Boolean Satisfiability (SAT)

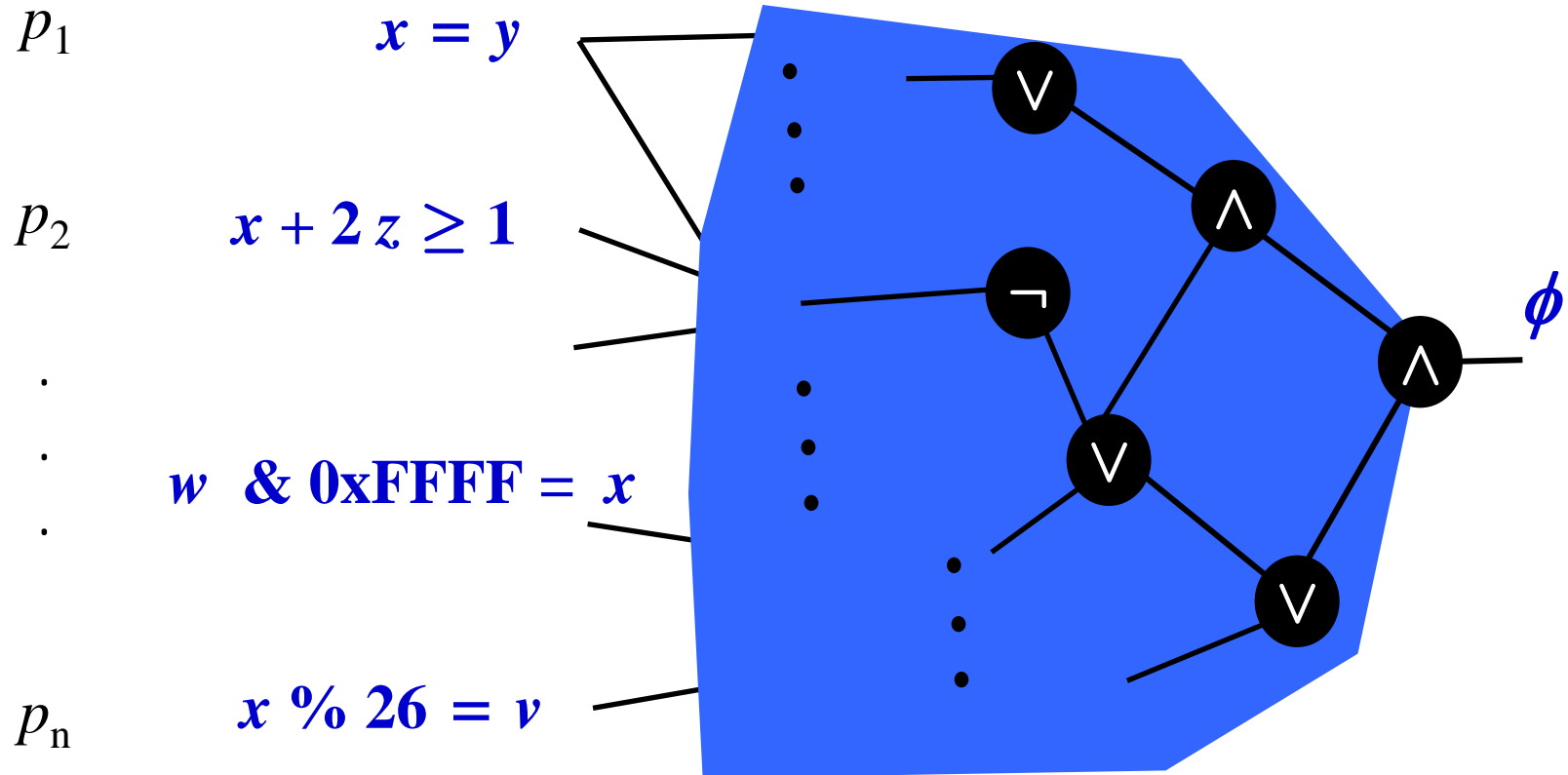


**Is there an assignment to the  $p_i$  variables  
s.t.  $\phi$  evaluates to 1?**

# 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, 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
  - finding security vulnerabilities
  - high-level (RTL and above) hardware verification
- A course goal: find new applications!

# Model Checking

- Broad Defn:

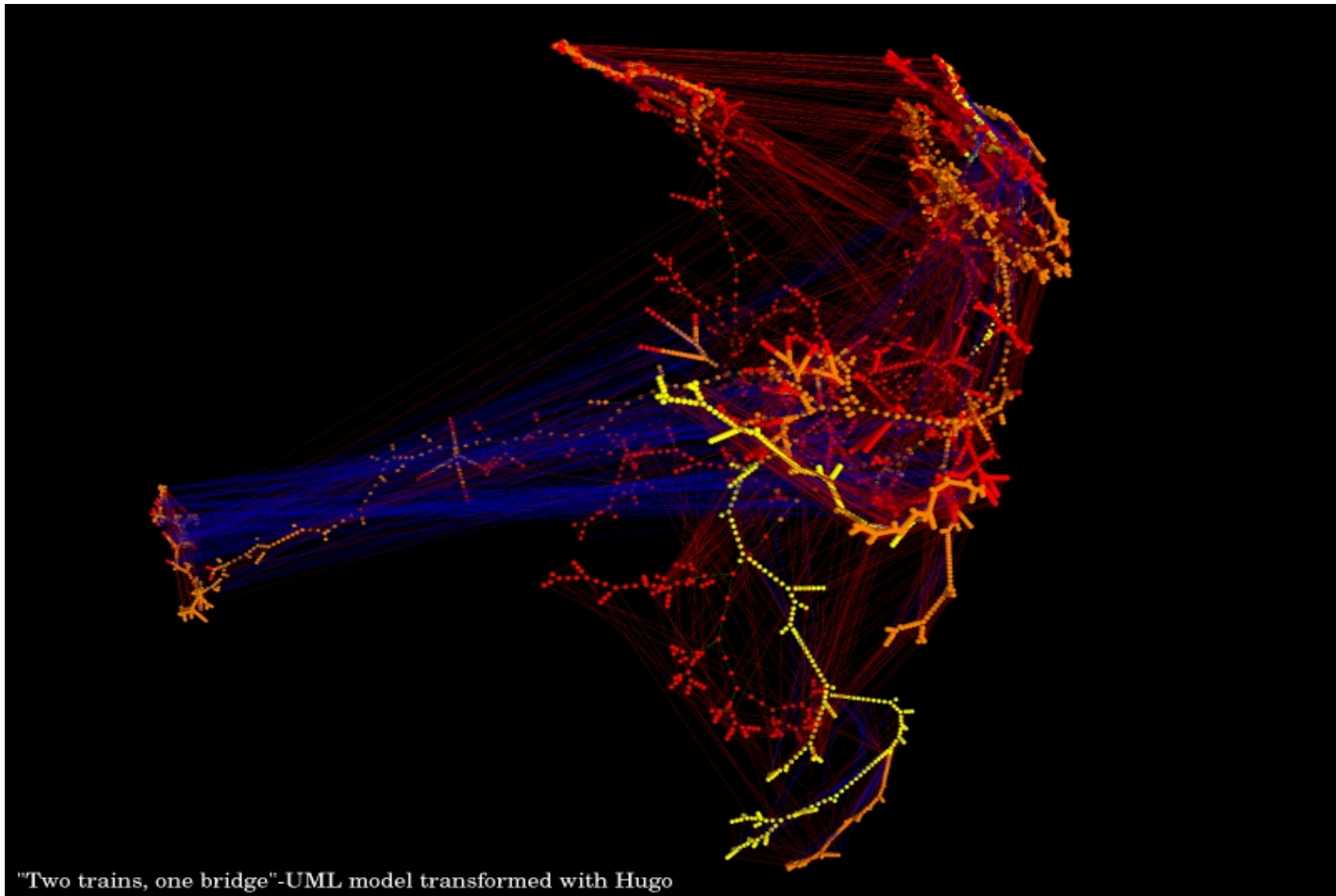
A collection of **algorithmic methods** based on **state space exploration** used to verify if a **system satisfies a formal specification.**

- Original Defn:

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



# Visualizing Model Checking



# 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 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 CHES 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

- 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 in separate slide set

# Related Classes

- Program Synthesis [294 – Bodik]
- Classic Program Verification (e.g., Hoare logic, abstract interpretation) [263 – Necula]
- Embedded Systems [249 – Sangiovanni-Vincentelli]
- Numerical Simulation [219A – Roychowdhury]
- Computer Security [261 – Wagner]

**JOINT PROJECTS ARE ENCOURAGED**



# Course Logistics

- Check out the webpage:  
[www.eecs.berkeley.edu/~sseshia/219c](http://www.eecs.berkeley.edu/~sseshia/219c)
- Tentative class schedule is up
  - 2007 Turing Award lecture screening this Friday
  - Next class will be Sep 7
  - IMP: Think about project topics in the interim

# 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
  - Special focus on synthesis from specifications

# Reference Books

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

# Grading

- 2-3 Homeworks (30%)
  - On the first half of the course
- Paper discussions / class participation (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
  - *~50% of past projects led to conference papers!*

# Misc.

- Office hours: MF 11 – 12, 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