EECS 219C: Formal Methods Introduction & Overview

Sanjit A. Seshia

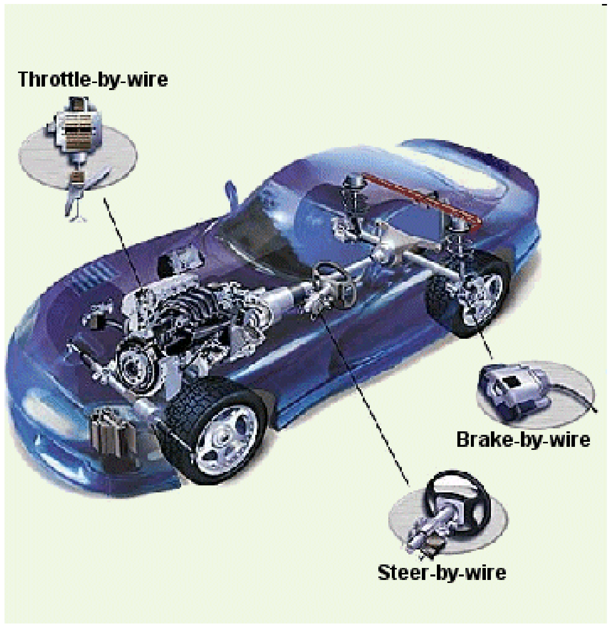
EECS, UC Berkeley

Buying a Car

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

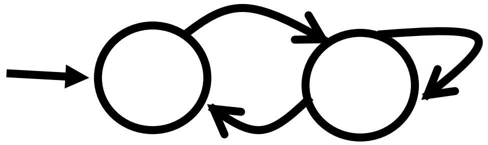
The Engineer’s Perspective

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**



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

Formal Verification (informally)

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

Formal Methods

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

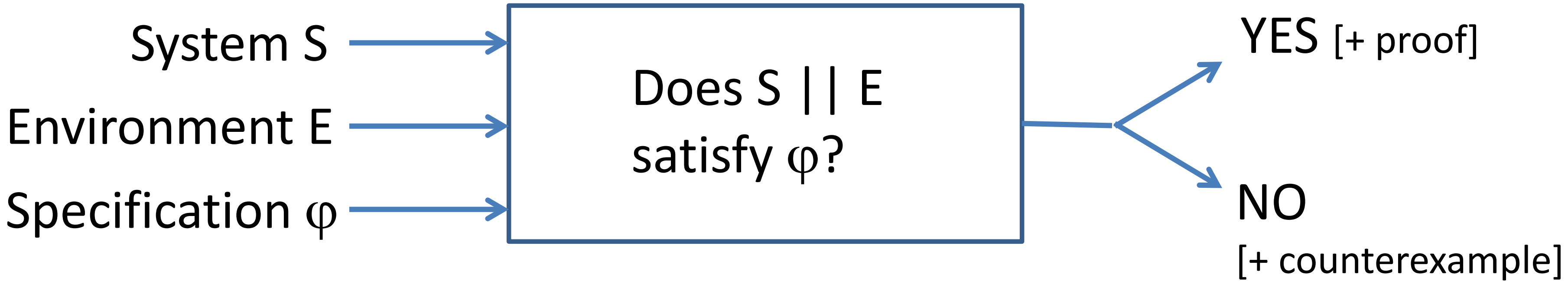
The essence: It's about PROOF

- Specify proof obligations
- Prove that system meets those obligations
- Synthesize provably-correct system

The Formal Methods Lens

- Formal Methods ≈ Computational Proof methods
  - Specification/Modeling ≈ Statement of Conjecture/Theorem
  - Verification ≈ Proving/Disproving the Conjecture
  - Synthesis ≈ Generating (parts of) Conjecture/Proof
  - ➢ Tools/techniques: SAT / SMT solvers, model checkers, theorem provers, simulation-based falsification, …

Verification:



Three Key Areas of Formal Methods

- Specification
  - WHAT must the system (program) do?
  - includes Modeling
- Verification
  - WHY does the system do it? (or not)
- Synthesis
  - HOW does the system do it?

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

“Formal Methods: Specification, Verification, Synthesis”

+

Theory

Practice

Computational Logic,
Algorithms,
Learning Theory,
Optimization

CAD for Circuits/Bio,
Computer Security,
Embedded/Cyber-Physical
Systems, Education

Current Foci: Verified Intelligent (AI) Systems /
Secure Systems

Class Introductions

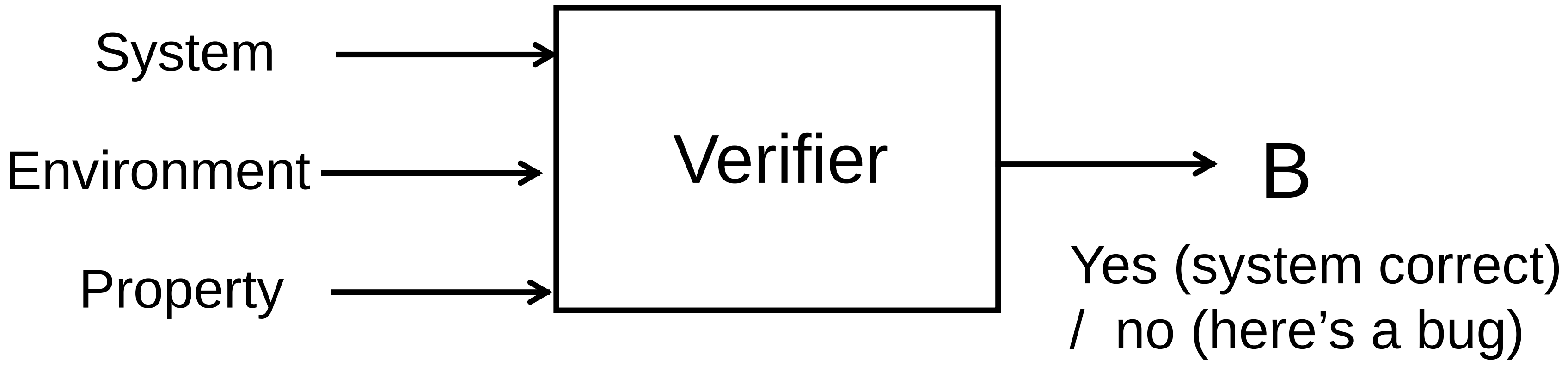
Please introduce yourselves

-- state name and research interests/areas

(Programming Systems, Computer Security, Arch/CAD, Embedded Systems, BioSystems, Control Theory, Robotics, etc.)

Formal Verification

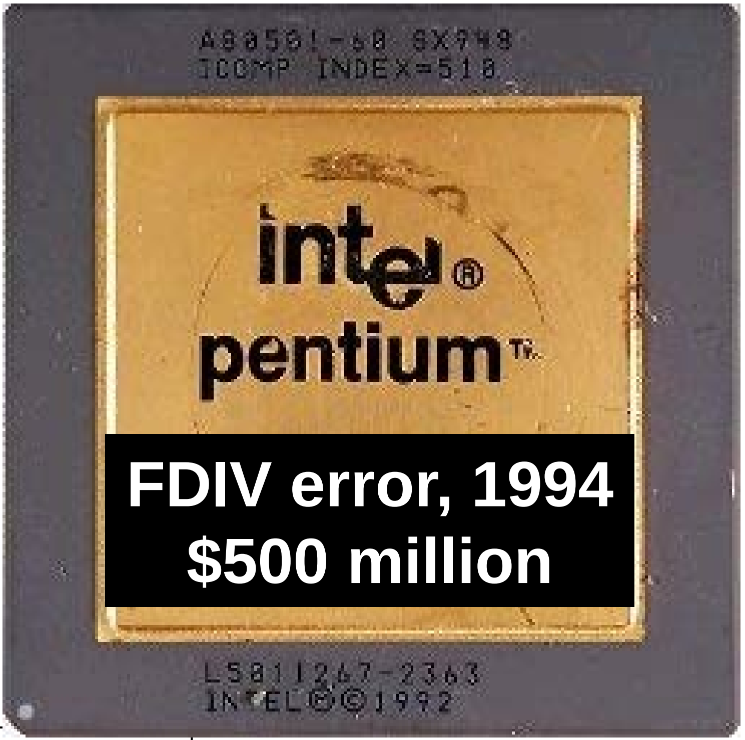
- Automatically verifying the correctness of systems



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

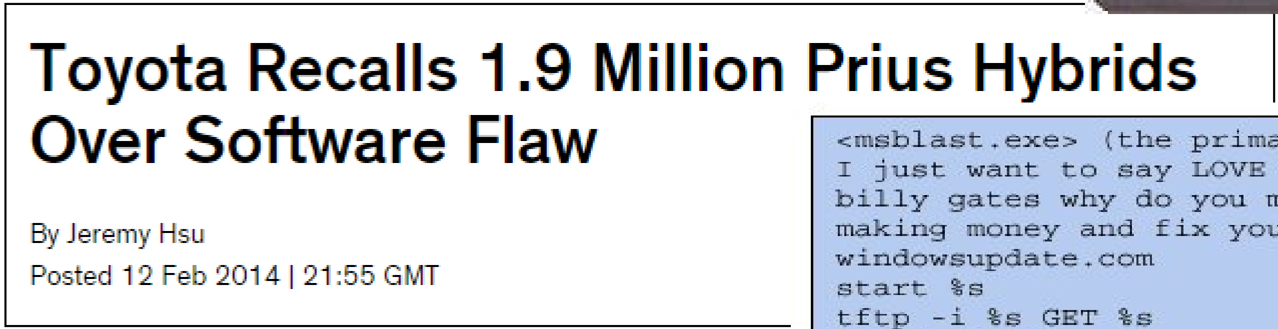
Ariane disaster, 1996

$500 million software failure



FDIV error, 1994

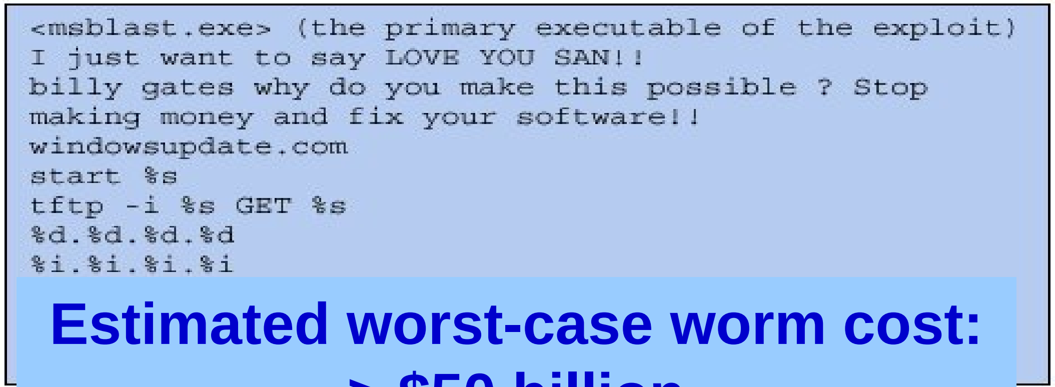
$500 million



Toyota Recalls 1.9 Million Prius Hybrids Over Software Flaw

By Jeremy Hsu

Posted 12 Feb 2014 | 21:55 GMT



```
<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
%i.%i.%i.%i
```

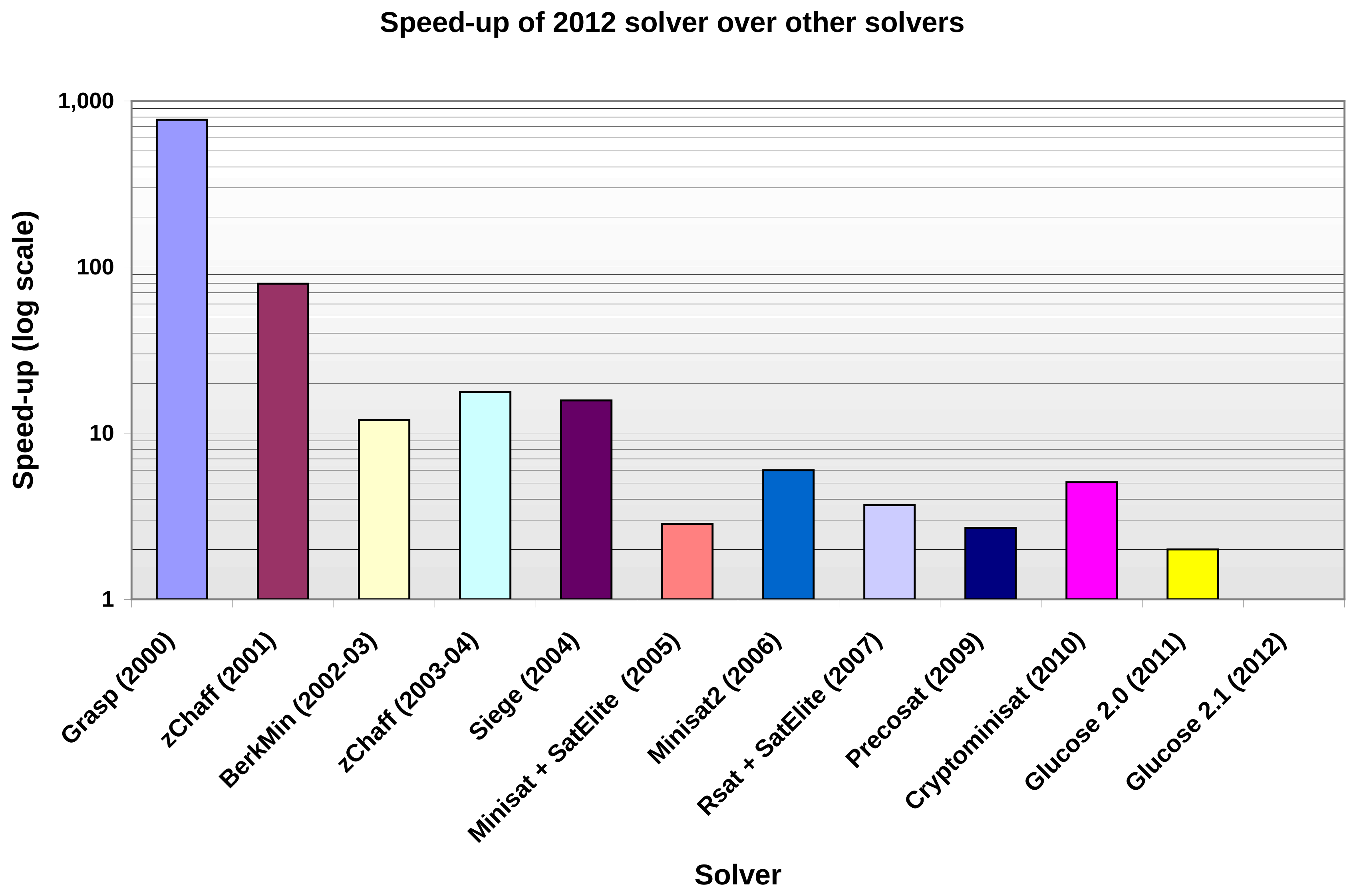
Estimated worst-case worm cost:
> $50 billion

Bugs cost Time, Money, Lives, …

Is Verification Feasible?

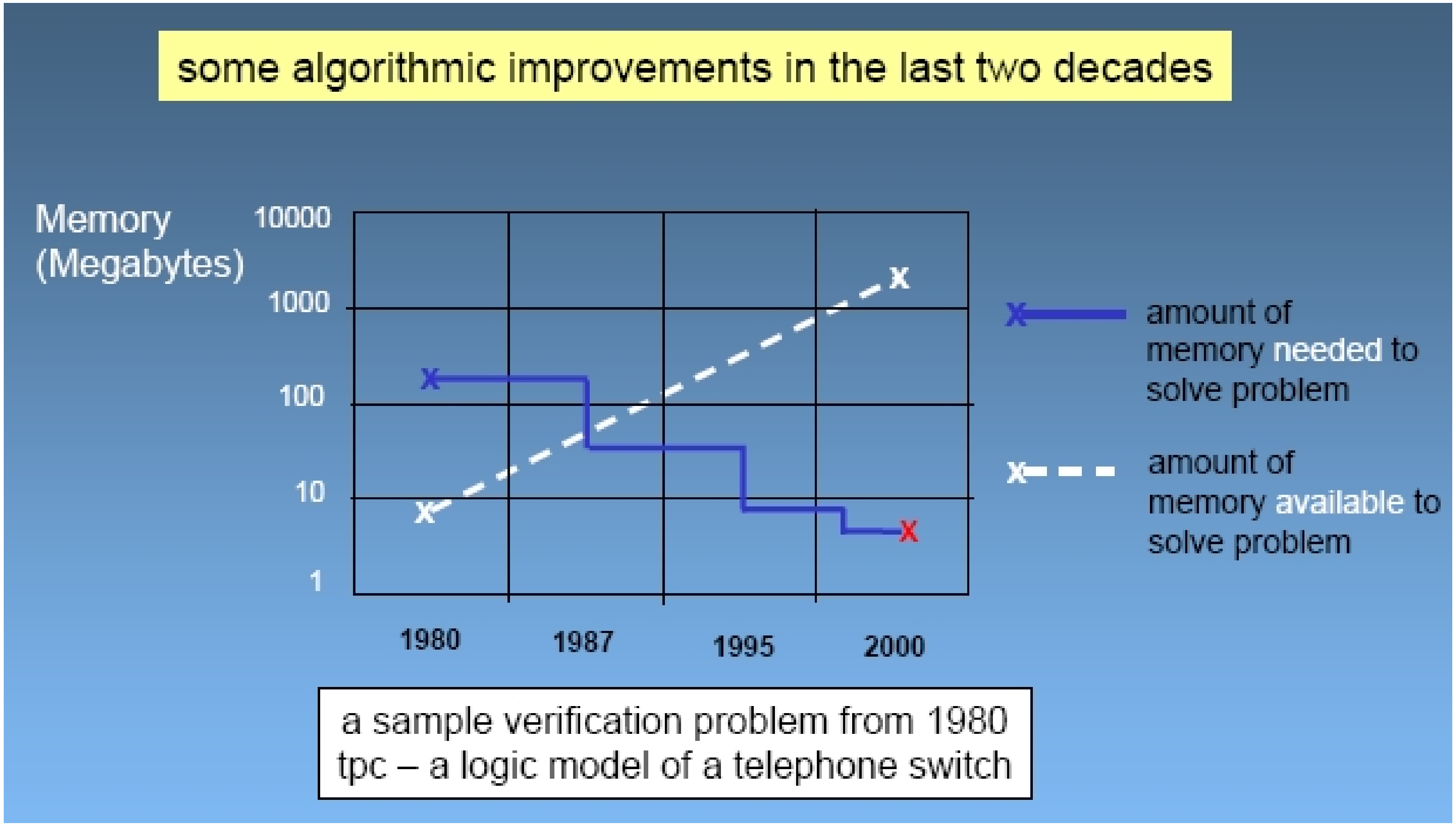
- "Easiest" non-trivial verification problem is NP-hard (SAT)
- But the outlook for practice is less gloomy than for theory…
  - More hardware resources
  - Better algorithms

My Experience with SAT Solving
(over ~a decade)



Experience with SPIN Model Checker

[G. Holzmann]



Topics in this Course

- Computational Engines
  - Boolean satisfiability (SAT)
  - Satisfiability modulo theories (SMT)
  - Model checking
  - Syntax-guided synthesis (SyGuS)
- Advanced Topics (“Research Frontiers”)
  - Deduction + Inductive Learning
  - Safe/Verified Artificial Intelligence (AI)
  - Human-Robot/Computer Interaction & Formal Methods
  - New application domains
  - … (more later in this lecture)

Topics of this Course (another view)

Application Domains

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

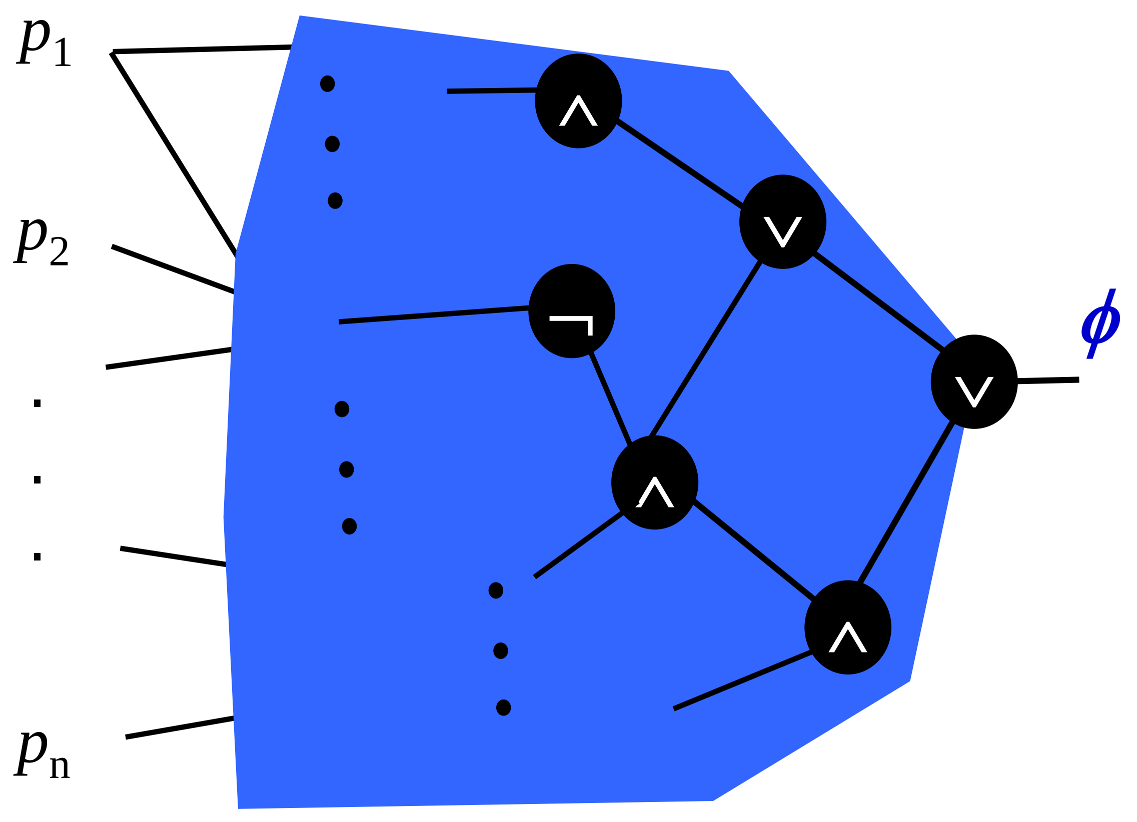
Verification/Synthesis Strategies

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

Computational Engines

SAT, BDDs, SMT

Boolean Satisfiability (SAT)

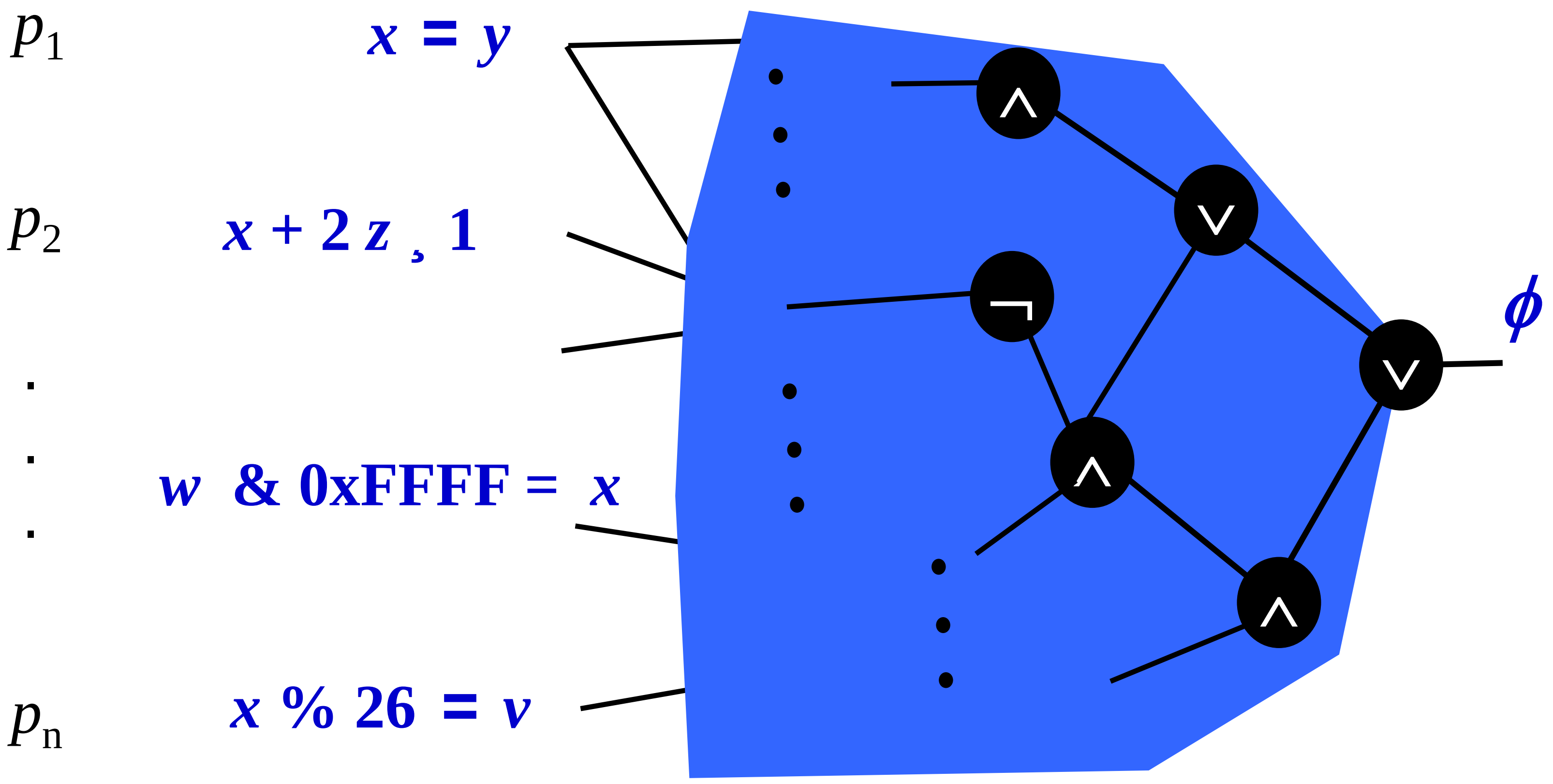


Is there an assignment to the p_i variables s.t. ϕ evaluates to 1?

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, logic 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
  - software synthesis
  - finding security vulnerabilities
  - high-level (RTL and above) hardware verification

Model Checking

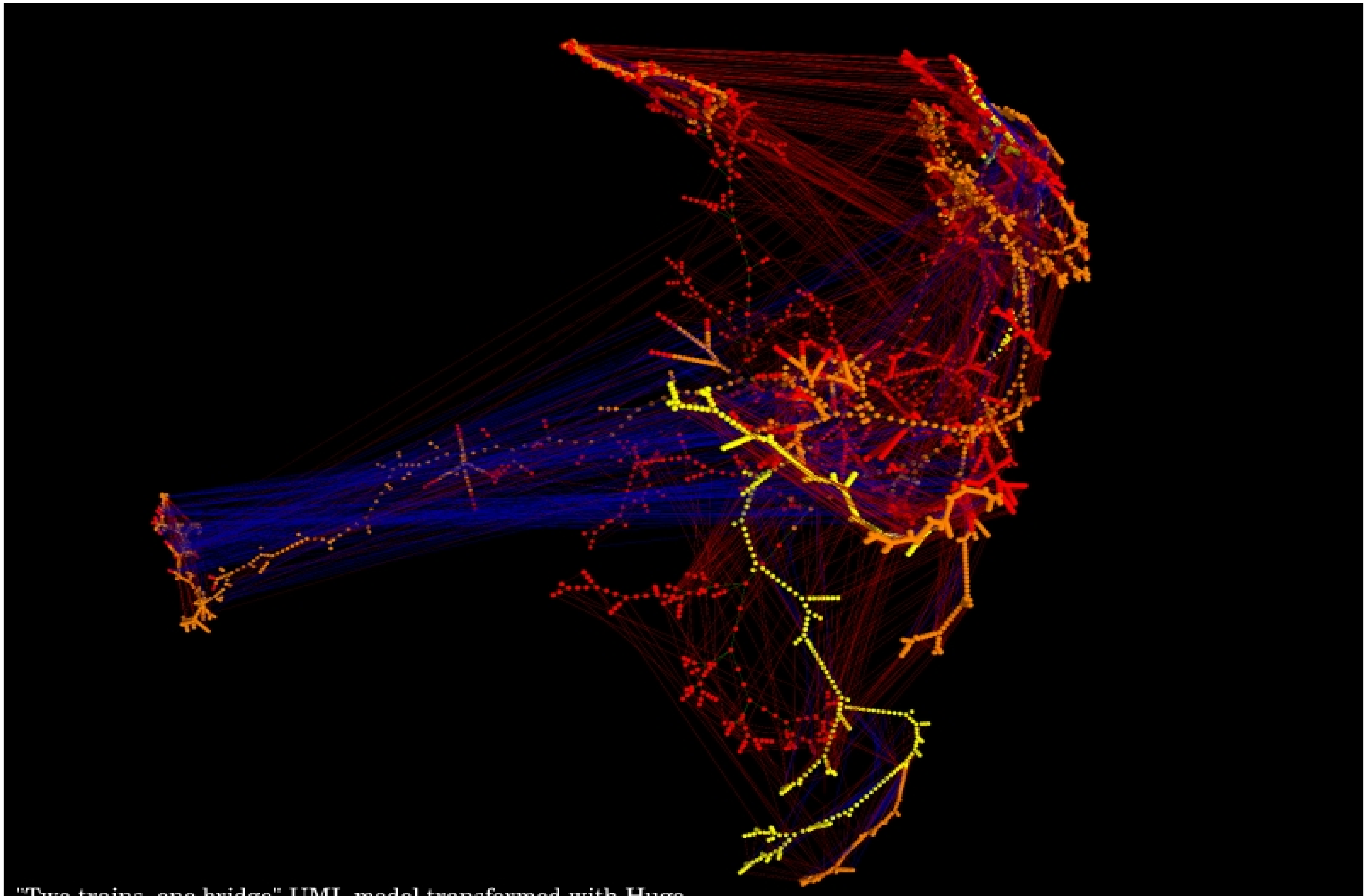
- Broad Defn:

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

[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; Turing Award for Lamport (in part for Specification/Verification work)

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 by month-end

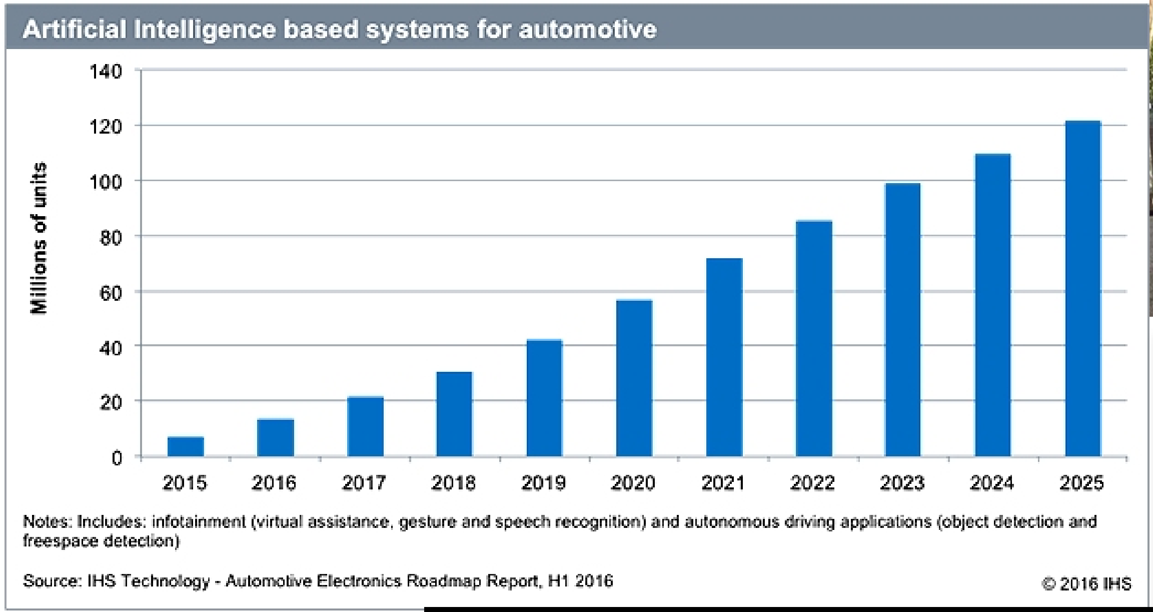
Formal Methods meets Machine Learning

- Machine Learning → Formal Methods
  - Greater efficiency, ease of use/applicability
  - Formal Inductive Synthesis
- Formal Methods → Machine Learning
  - Stronger assurances of safety/correctness for learning systems

Further details:

1. S. A. Seshia, “Combining Induction, Deduction, and Structure for Verification and Synthesis”, Proceedings of the IEEE, November 2015.
2. S. A. Seshia, D. Sadigh, and S. S. Sastry, “Towards Verified Artificial Intelligence”, July 2016, http://arxiv.org/abs/1606.08514

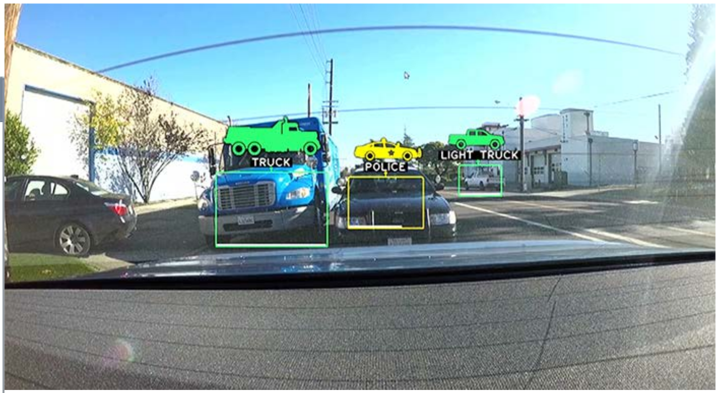
Growing Use of Machine Learning/AI in Cyber-Physical Systems

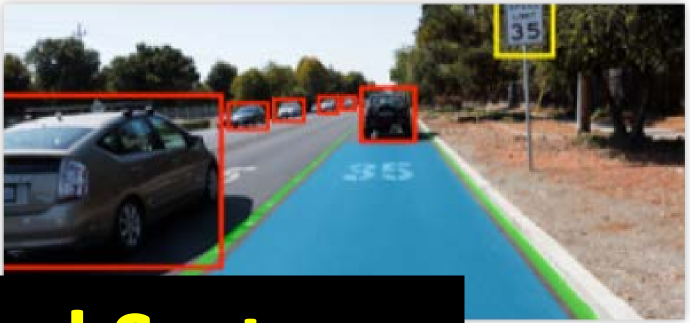


Notes: Includes: infotainment (virtual assistance, gesture and speech recognition) and autonomous driving applications (object detection and freespace detection)

Source: IHS Technology - Automotive Electronics Roadmap Report, H1 2016

© 2016 IHS

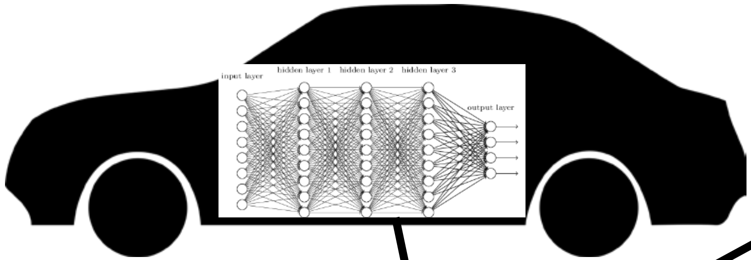




Many Safety-Critical Systems

Challenges for Verified AI

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514.



System **S** →

Environment **E** →

Specification **φ** →

Does S || E satisfy φ?

→ YES [+ proof]

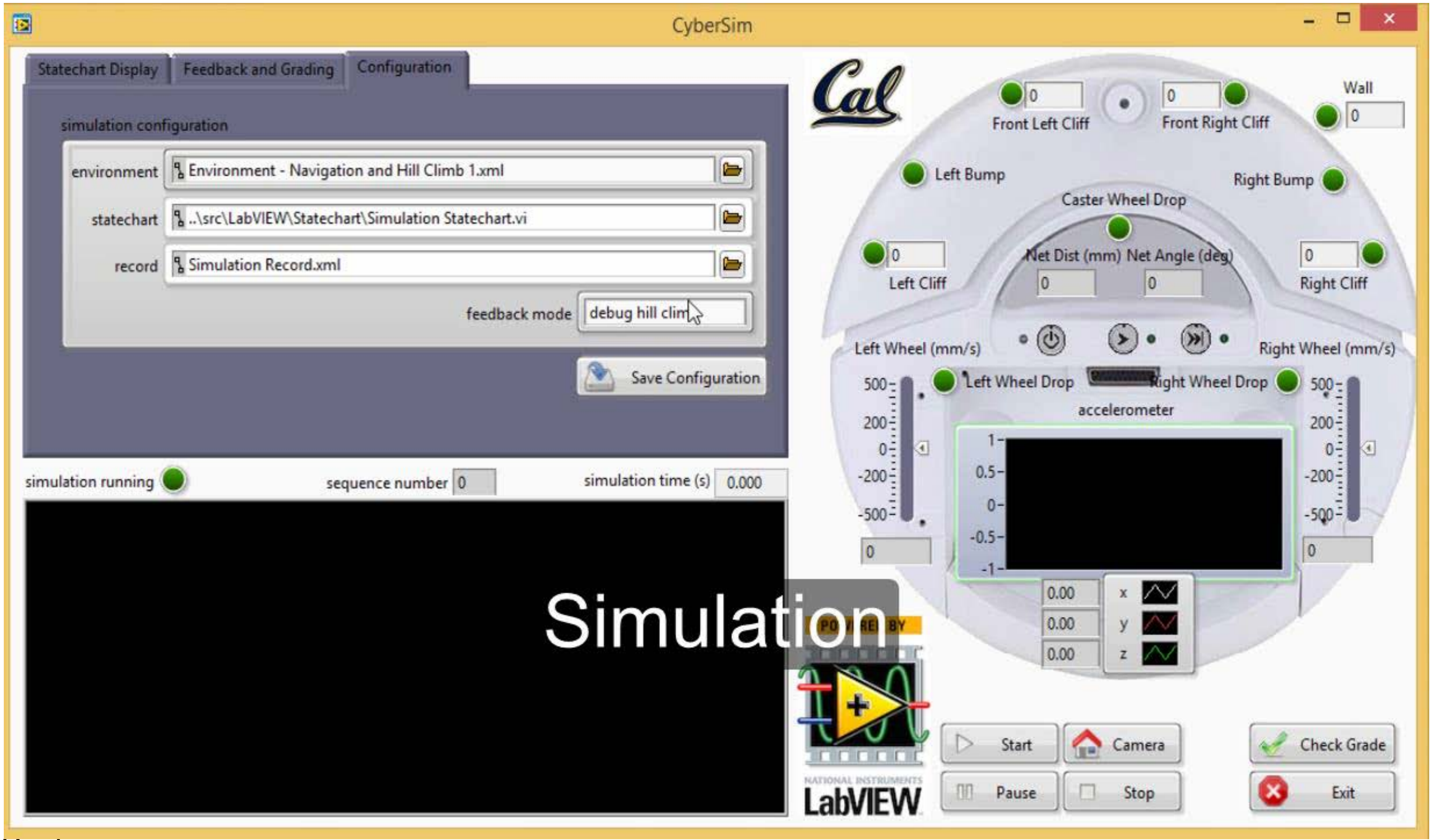
→ NO [+ counterexample]

Design Correct-by-Construction instead?

Counterexamples, etc. from Rich Signal Spaces?

Formal Methods for Education

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



CyberSim

simulation configuration

environment: Environment - Navigation and Hill Climb 1.xml

statechart: ..\src\LabVIEW\Statechart\Simulation Statechart.vi

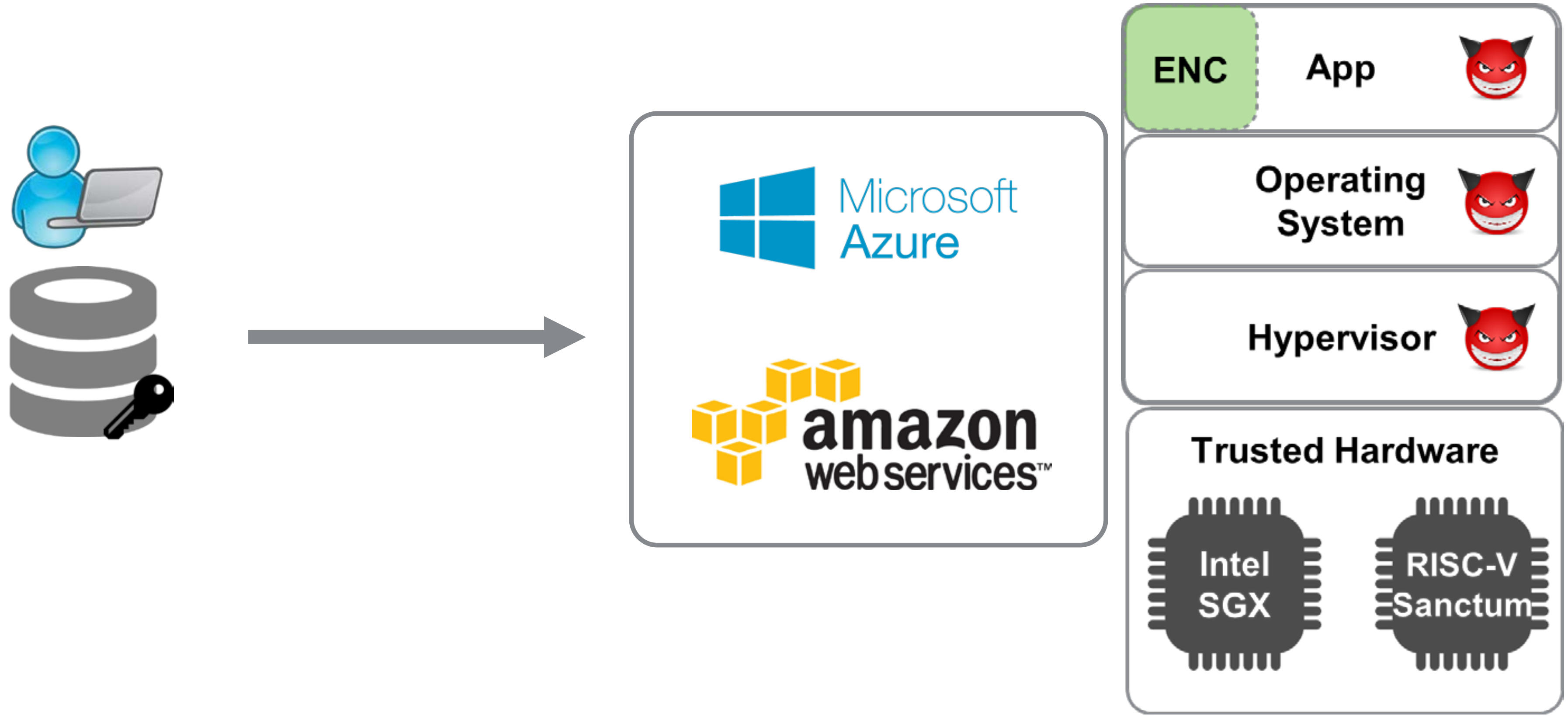
record: Simulation Record.xml

feedback mode: debug hill clim

sequence number: 0

simulation time (s): 0.000

Formal Methods for Secure Systems



- Does my secret data remain secret?
- Does the program execute as it is supposed to?
- Is the right program executed?

Course Logistics

- Check out the webpage:

  www.eecs.berkeley.edu/~sseshia/219c

- Tentative class schedule will be up soon
  - 2007, 2013 Turing Award lectures (optional viewing)
  - IMP: Think about project topics!

Course Outline

- 2 parts
- Part I: Model Checking, Theorem Proving, 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

- No textbook; course notes from previous years
- See list on the website
- e-Handouts for most material
  → on bCourses

Grading

- Homework (25%)
  - First part of the course
- Scribing lectures (15%)
  - 2 lectures per person: Scribe one lecture, edit another lecture
  - Sign-up sheet next week
- Paper discussions / class participation (10%)
  - Last month of the course
- **Project (50%)**
  - 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: MW 2:30 – 3:30, 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