

Environment Modeling: A Usability Challenge for Verifying Cyber-Physical Systems

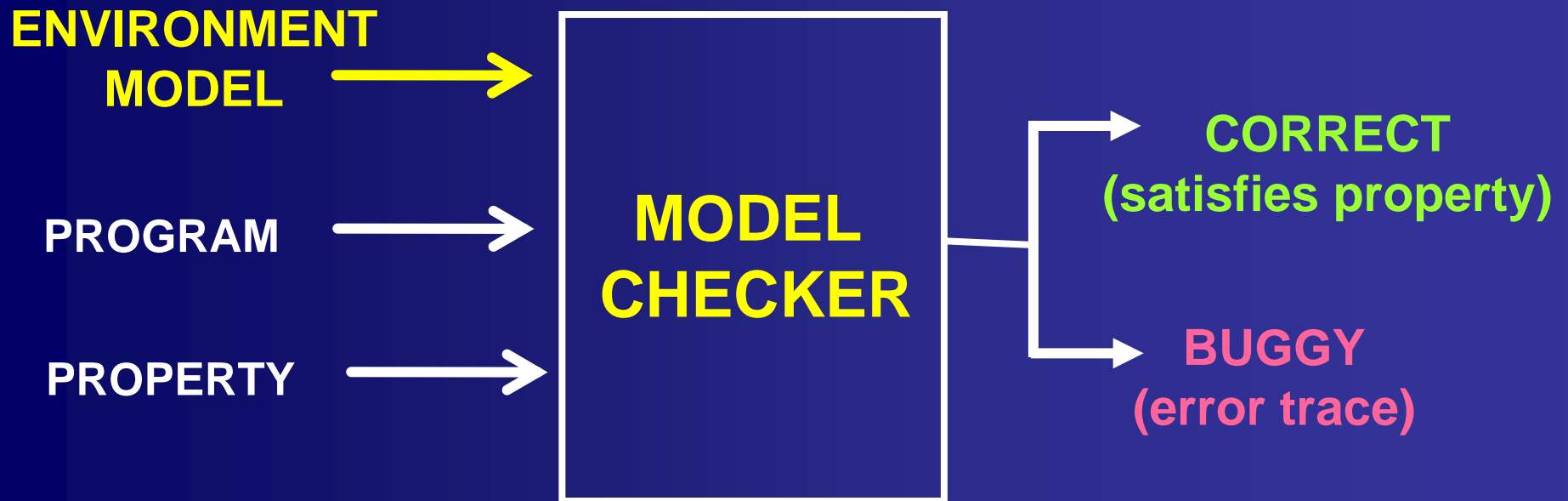
Sanjit A. Seshia

**EECS Department
UC Berkeley**

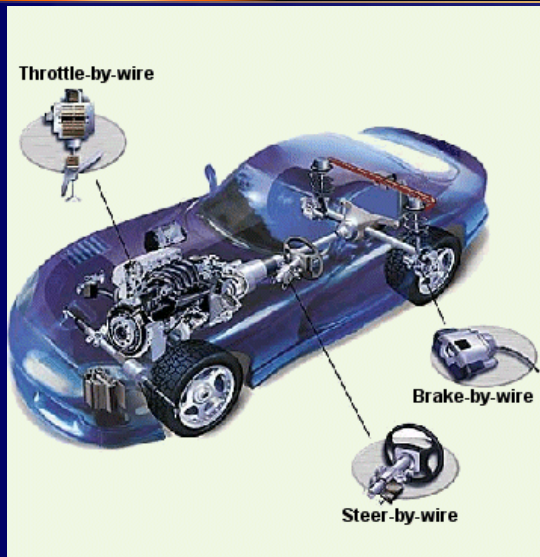
Workshop on Usable Verification

November 2010

Typical Verification Picture: Is it complete?!



Quantitative Analysis / Verification

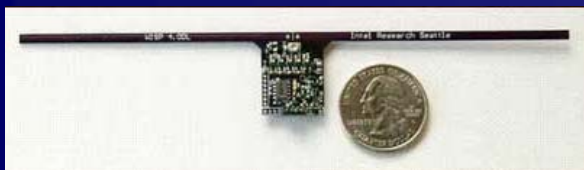


Does the brake-by-wire software always actuate the brakes within 1 ms?

Safety-critical embedded systems

Can this new app drain my iPhone battery in an hour?

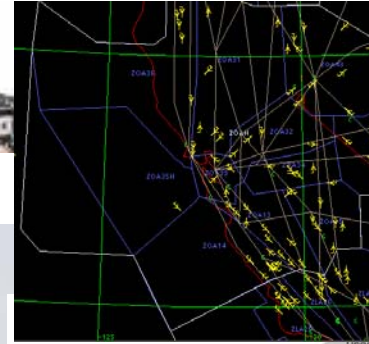
Consumer devices



How much energy must the sensor node harvest for RSA encryption?

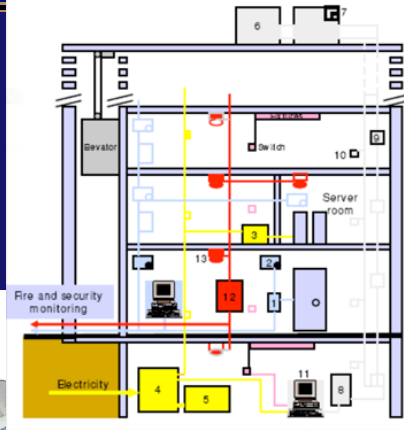
Energy-limited sensor nets, bio-medical apps, etc.

Cyber-Physical Systems (CPS): Orchestrating networked computation with physical systems

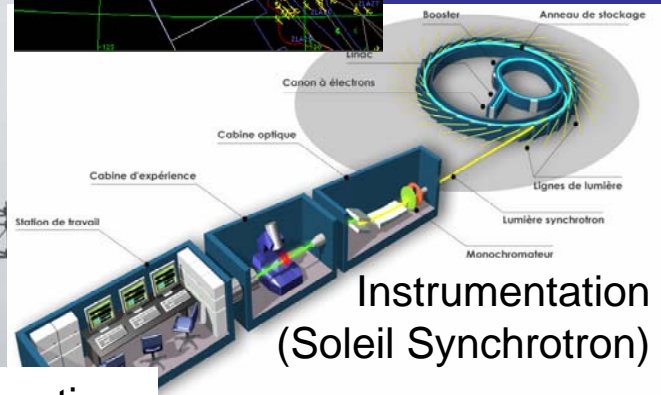


Transportation
(Air traffic
control at
SFO)

Building Systems

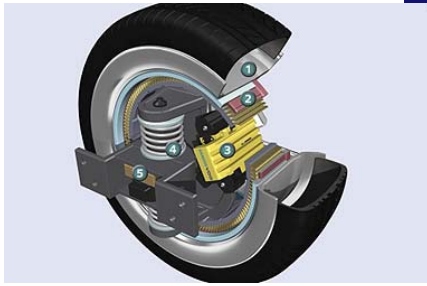


Telecommunications

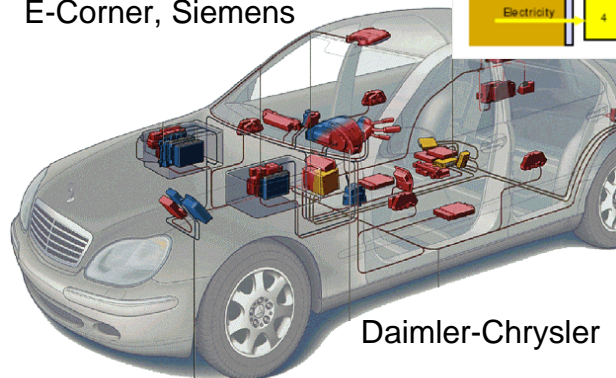


Instrumentation
(Soleil Synchrotron)

Automotive



E-Corner, Siemens



Daimler-Chrysler

Military systems:



Courtesy of Doug Schmidt

Power generation and distribution



Courtesy of
General Electric

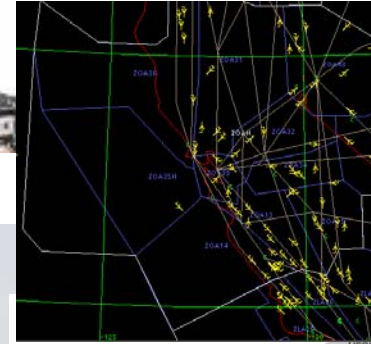
Factory automation



Courtesy of Kuka Robotics Corp.

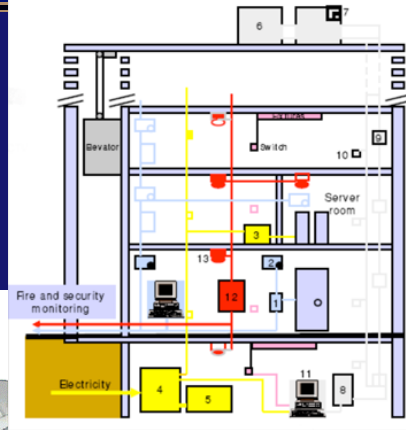
[E.A.Lee]

Cyber-Physical Systems (CPS): Orchestrating networked computation with physical systems

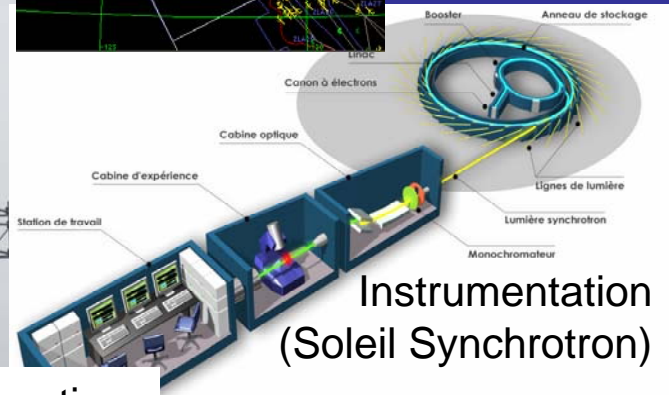


Transportation
(Air traffic
control at
SFO)

Building Systems

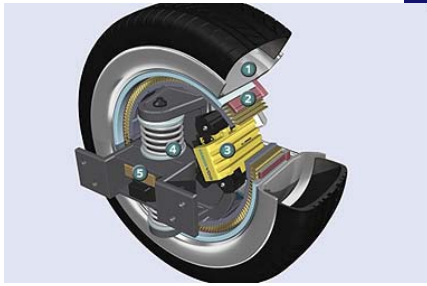


Telecommunications

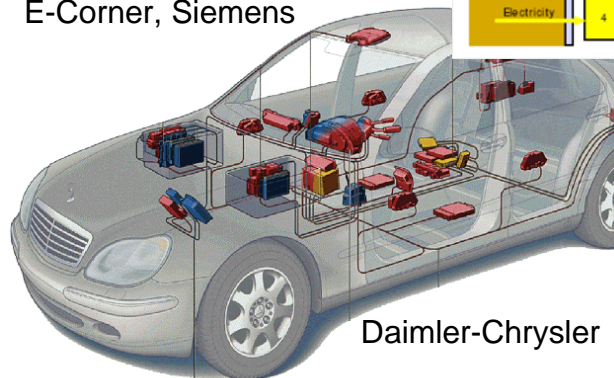


Instrumentation
(Soleil Synchrotron)

Automotive



E-Corner, Siemens



Daimler-Chrysler

Military systems:



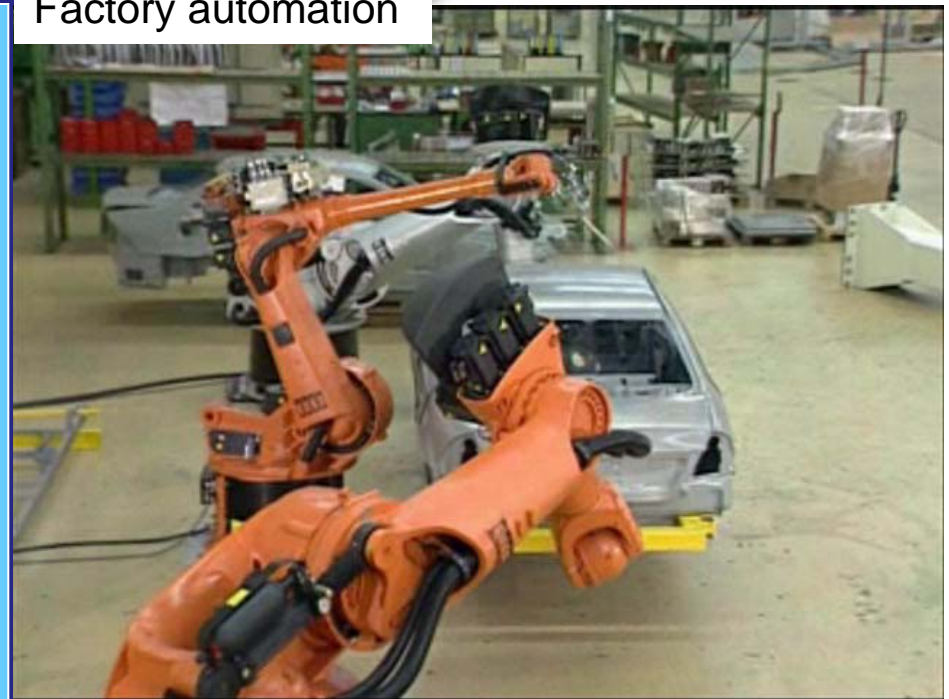
Courtesy of Doug Schmidt

Power generation and distribution



Courtesy of
General Electric

Factory automation



Courtesy of Kuka Robotics Corp.

Time is Central to Cyber-Physical Systems

Several timing analysis problems:

- Worst-case execution time (**WCET**) estimation
- Estimating **distribution** of execution times
- **Threshold** property: can you produce a test case that causes a program to violate its deadline?
- **Software-in-the-loop simulation**: predict execution time of particular program path

Challenge: Environment Modeling (Timing Analysis)

- Timing properties of the Program depend heavily on its environment
- Environment =
 - Processor & Memory Hierarchy
 - + Operating System, other processes/threads, ...
 - + Network
 - + I/O Devices
 - + ...
- Modeling the full environment is hard!
- However, we need a 'reasonably' precise environment model
 - Unlike traditional software verification

Relative Success of “Boolean” Software Verification

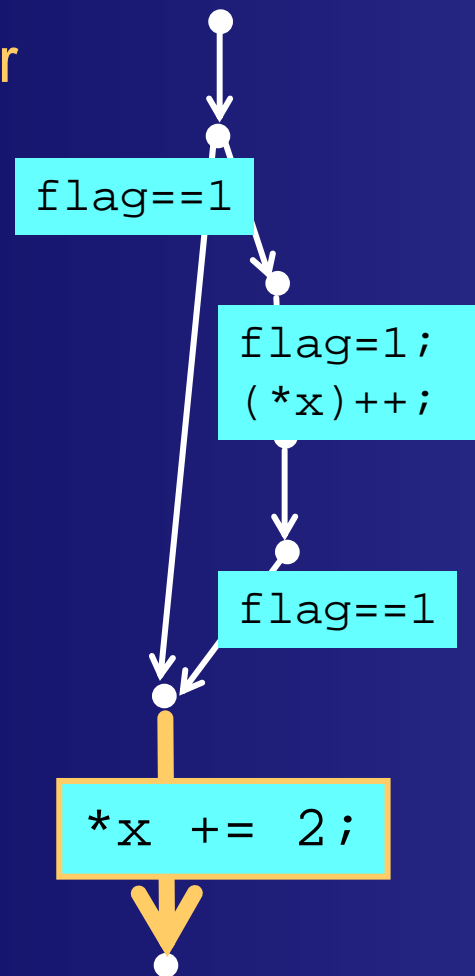
- From theoretical ideas to industrial practice in ~ 15 yrs

Some Reasons:

- Availability of open source software
- Well-defined target problems: Device drivers, memory safety, security vulnerabilities, concurrency, ...
- Value of bug finding
- **Coarse abstraction of environment OK**

Challenge of Timing Analysis: Example

On a processor
with a data
cache



CFG unrolled
to a DAG

Timing of an edge (basic block) depends on:

- **Program path** it lies on
- Initial **platform state**

Challenges:

- **Exponential number** of paths and platform states!
- **Lack of visibility** into platform state

Current State-of-the-art for Timing Analysis

```
Microsoft Development Environment (Sharp) - csporch  
File Edit View Debug Tools Window Help  
csporch [csporch]  
#define ESPICH CONSTRAINED  
#define ESPICH_HYBRID_MODEL  
#define ESPICH_M1  
#define ESPICH_M2  
#define ESPICH_MP1  
#define ESPICH_T1 1.000000000  
#define ESPICH_M3  
#define ESPICH_M4  
#define ESPICH_M5  
#define ESPICH_M6  
#define ESPICH_M7  
#define ESPICH_M8  
#define ESPICH_M9  
#define ESPICH_M10  
#define ESPICH_M11  
#define ESPICH_M12  
#define ESPICH_M13  
#define ESPICH_M14  
#define ESPICH_M15  
#define ESPICH_M16  
#define ESPICH_M17  
#define ESPICH_M18  
#define ESPICH_M19  
#define ESPICH_M20  
#define ESPICH_M21  
#define ESPICH_M22  
#define ESPICH_M23  
#define ESPICH_M24  
#define ESPICH_M25  
#define ESPICH_M26  
#define ESPICH_M27  
#define ESPICH_M28  
#define ESPICH_M29  
#define ESPICH_M30  
#define ESPICH_M31  
#define ESPICH_M32  
#define ESPICH_M33  
#define ESPICH_M34  
#define ESPICH_M35  
#define ESPICH_M36  
#define ESPICH_M37  
#define ESPICH_M38  
#define ESPICH_M39  
#define ESPICH_M40  
#define ESPICH_M41  
#define ESPICH_M42  
#define ESPICH_M43  
#define ESPICH_M44  
#define ESPICH_M45  
#define ESPICH_M46  
#define ESPICH_M47  
#define ESPICH_M48  
#define ESPICH_M49  
#define ESPICH_M50  
#define ESPICH_M51  
#define ESPICH_M52  
#define ESPICH_M53  
#define ESPICH_M54  
#define ESPICH_M55  
#define ESPICH_M56  
#define ESPICH_M57  
#define ESPICH_M58  
#define ESPICH_M59  
#define ESPICH_M60  
#define ESPICH_M61  
#define ESPICH_M62  
#define ESPICH_M63  
#define ESPICH_M64  
#define ESPICH_M65  
#define ESPICH_M66  
#define ESPICH_M67  
#define ESPICH_M68  
#define ESPICH_M69  
#define ESPICH_M70  
#define ESPICH_M71  
#define ESPICH_M72  
#define ESPICH_M73  
#define ESPICH_M74  
#define ESPICH_M75  
#define ESPICH_M76  
#define ESPICH_M77  
#define ESPICH_M78  
#define ESPICH_M79  
#define ESPICH_M80  
#define ESPICH_M81  
#define ESPICH_M82  
#define ESPICH_M83  
#define ESPICH_M84  
#define ESPICH_M85  
#define ESPICH_M86  
#define ESPICH_M87  
#define ESPICH_M88  
#define ESPICH_M89  
#define ESPICH_M90  
#define ESPICH_M91  
#define ESPICH_M92  
#define ESPICH_M93  
#define ESPICH_M94  
#define ESPICH_M95  
#define ESPICH_M96  
#define ESPICH_M97  
#define ESPICH_M98  
#define ESPICH_M99  
#define ESPICH_M100
```

- Program = Sequential, terminating program
- Runs uninterrupted

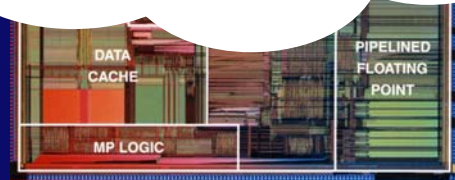
PROBLEM:

Can take several man-months to construct!

Also:

- Limited to extreme-case analysis
- Often requires additional platform specification from users

- Platform =
Simple Pipelined Processor
+ Data/Instruction Cache



Existing Approaches: One-size-fits-all?

- Why construct a **SINGLE** timing model for **ALL** programs?
- Only interested in analyzing a specific program.
- Why not **automatically synthesize** a program-specific timing model?



Promising Direction

(for timing analysis and quantitative verification in general)

- **Inductive Synthesis**
 - Automatically generate environment model through **active learning**
- **Active = Select behaviors from which to learn**
- **Use core verification techniques (SAT, SMT, model checking, ...) to generate selected behaviors**

- **Example: **GameTime** for timing analysis of software**

S. A. Seshia and A. Rakhlin, “Quantitative Analysis of Systems Using Game-Theoretic Learning”, ACM Trans. Embedded Computing Systems.

The GameTime Approach: Overview

Game-Theoretic Online Learning + Satisfiability Solving Modulo Theories (SMT)

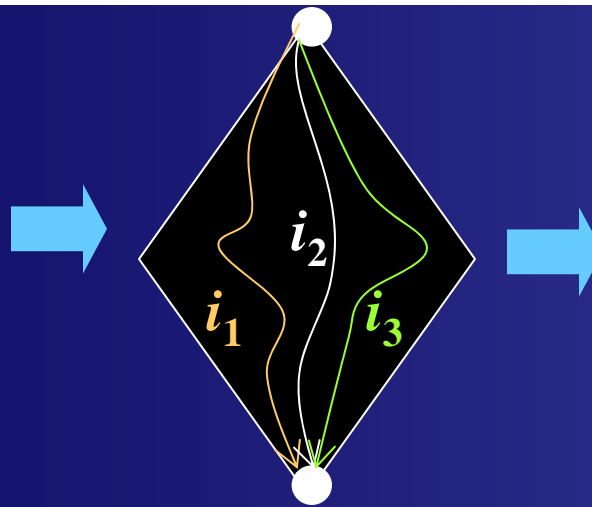
```

Microsoft Development Environment (VS2005) - c:\proj\...
File Edit View Debug Tools Window Help

#pragma once
#define EPOCHS 100000000
#define EPOCHS_IT 1
#define EPOCHS_IT 2
#define EPOCHS_IT 1
#define EPOCHS_IT 1.00000000
#define EPOCHS_IT 1
#define EPOCHS_IT 1
#define EPOCHS_IT 1
#define EPOCHS_IT 1
#define EPOCHS_IT 1
#define EPOCHS_IT 1
#define EPOCHS_IT 1
static double EPOCHS_IT[] =
{ 0.492021, 0.0, 0.0, 0.492021, 0.999999, 0.0,
  0.0, 0.999999, 1.0, 0.0, 0.0 };

static double EPOCHS_G[] =
{ 0.1, -1.0, 0.0 };

static double EPOCHS_H[] =
{ 0.492021, 0.999999, 0.0, 0.0, 0.492021, 0.0,
  1.0, 0.492021, 0.492021, 0.0, 0.0, 0.999999, 0.0,
  1.0, 0.492021, 0.0, 0.0, -1.0, 0.0,
  0.999999, 0.0, 0.492021, 0.0, 0.0, 1.0, 0.492021, 0.999999,
  0.0, 0.492021, 0.999999, 0.492021 };
    
```



i_1	42
i_2	75
i_3	101
\vdots	\vdots



LEARNING ALGORITHM



PREDICT TIMING PROPERTIES
(worst-case, distribution, etc.)

PROGRAM

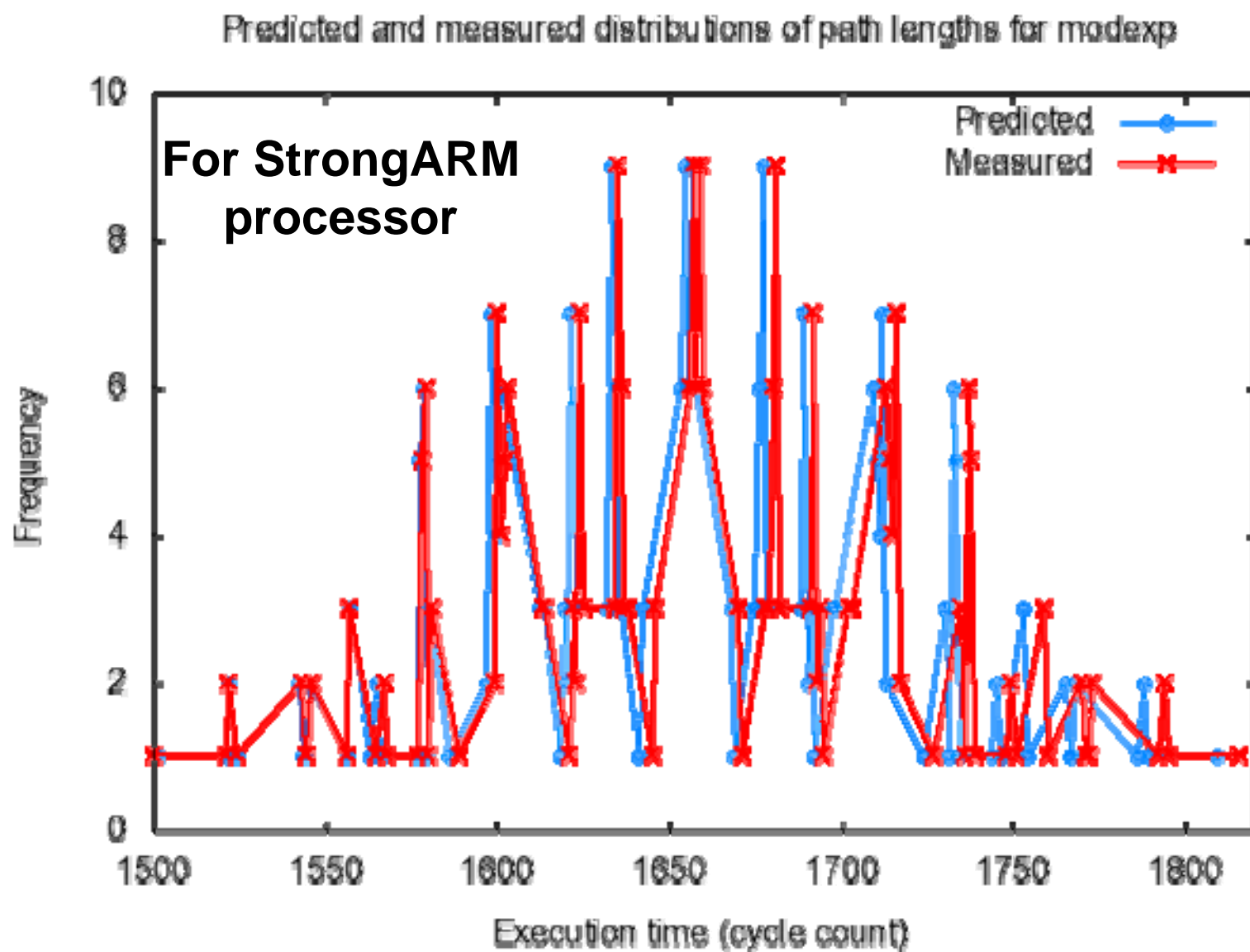
CONTROL-FLOW GRAPH

MEASURE EXECUTION TIMES

EXTRACT BASIS PATHS
SMT SOLVER GENERATES TEST INPUTS

Publication: S. A. Seshia and A. Rakhlin, "Quantitative Analysis of Systems Using Game-Theoretic Learning", ACM Trans. Embedded Computing Systems.

Estimating the Distribution of Times for Modular Exponentiation: predictions from 9 measurements in blue, actual 256 measurements in red



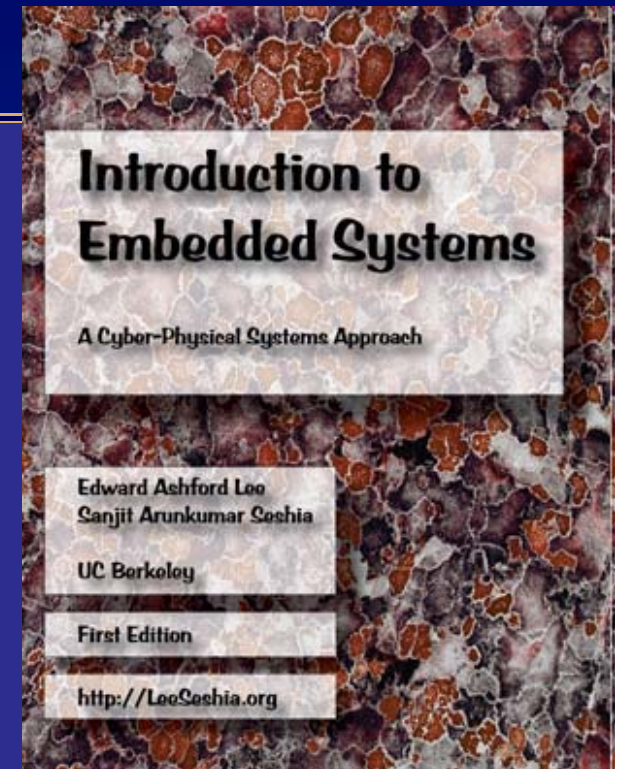
Potential Barriers (from Academic Perspective)

■ Student Skills

- Students need cross-cutting skills (or willingness to learn)
- Hardware + Software + Formal Methods
- EE + CS
- New UG course at Berkeley on Embedded Systems (EECS 149)

■ Lack of Open-Source Benchmarks

- More challenging for “quantitative” software verification!
 - Heavy dependence on hardware platform



Summary

- **Quantitative Verification of Cyber-Physical Software Systems**
- **Challenge: Environment modeling**
 - Current manual methods too tedious and error-prone
- **Proposed Approach: Automatic model generation by Inductive Synthesis**
 - Active Learning + Traditional verification techniques (e.g., SAT/SMT)
 - One instance: **GameTime** for timing analysis of software
 - Perhaps a killer app for synthesis methods?