Quantitative Verification and Synthesis of Systems

Sanjit A. Seshia
Assistant Professor
EECS, UC Berkeley

Software-at-Scale Workshop
August 2010
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.**
Outline

- Boolean vs. Quantitative Verification
- An Example: Timing Analysis
- Challenges: Environment Modeling
- Approach: Learning Program-Specific Environment Model
- Conclusion and Questions
Traditional “Boolean” Verification

System $S$  \hspace{1cm} Environment $E$  \hspace{1cm} Specification $\Phi$

VERIFIER

Does $S \parallel E \models \Phi$?

YES / NO
Boolean Verification on Models

Finite Automata System \( S \)  
Environment \( E \)  
Specification \( \Phi \)

VERIFIER

Does \( S \parallel E \models \Phi \) ?

Satisfiability Problem

YES / NO
Boolean Verification on Software

Program
System $S$

MODEL
Environment $E$

Logical Formula
Specification $\Phi$

e.g. Finite Automata

VERIFIER
Does $S \parallel E \models \Phi$?

Env model typically over-approximate

YES / NO
Success of Boolean Software Verification

- From theoretical ideas to industrial practice in ~ 15 yrs

Some Reasons:
- Availability of open source software
- Well-defined problems: Device drivers, memory safety, security vulnerabilities, concurrency, ...
- Value of bug finding
- Less sensitive to imprecise environment model
Quantitative Verification

At least one of these holds

YES / NO

S. A. Seshia
Quantitative Verification on Models

Timed Automata
System $S$ → Environment $E$ → Specification $\Phi$ → G(request $\to F_{\leq 5}$ response)

VERIFIER

Worst-case latency request $\to$ response
Quantitative Verification on Software

Program
System
S

MODEL?
Environment
E

G(request → F ≤5 response)
Specification
Φ

Timed Automaton

VERIFIER

Worst-case latency
request → response

S. A. Seshia
Challenge: Environment Modeling (Quantity = Time)

- Timing properties of the Program depend heavily on its environment

- Environment =
  - Processor & Memory Hierarchy
  - Operating System, other processes/threads, ...
  - Network
  - I/O Devices
  - ...

- Cannot construct a Program model independent of its environment!
  - Unlike Boolean version of the Verification problem
Simplifying the Problem

- **Program = Sequential, terminating program**
- **Runs uninterrupted**

- **Environment = Single-core Processor + Memory Hierarchy**
Simple Illustrative Program

while(!flag)
{
  flag = 1;
  *x++;
}
*x += 2;

Control-flow graph (loop bound = 1)

CFG unrolled to a DAG
On a processor with a data cache

Simple Illustrative Program

flag == 1

*\texttt{x}++;

flag == 1

*\texttt{x} += 2;

CFG unrolled to a DAG
Simple Illustrative Program

On a processor with a data cache

Case 1: x is originally in cache

flag==1

flag=1; *x++;

*x += 2;

CFG unrolled to a DAG

S. A. Seshia
On a processor with a data cache

Case 2: x is NOT originally in cache

CFG unrolled to a DAG

-1 6-

S. A. Seshia
Summary of Example

- Edge timing depends on
  1. Initial environment state
  2. Program path executed

- Challenge:
  - Exponentially-many env states!
  - Exponentially-many program paths!

- E.g. Finding Worst-case execution time bound
  - Very loose bound somewhat easy (need to consider timing anomalies)
  - Reasonably tight bound \( \Rightarrow \) VERY HARD
Current State-of-the-art for Timing Analysis

- Program = Sequential, terminating program
- Runs uninterrupted

PROBLEM:
Takes several man-months to construct!
Also: limited to extreme-case analysis

- Environment = Single-core Processor + Memory Hierarchy
One-size-fits-all Solution?

- Why aim to construct a SINGLE timing model for ALL programs?

- We are only interested in verifying a specific program!

- Why not construct a program-specific timing model?
  - Even done in Boolean verification
The GameTime Approach

[Seshia & Rakhlin, ICCAD’08, ACM TECS’10]

- Environment is an ‘adversary’ who picks weights on the edges of the control-flow graph
- Weights selected in two-phase approach
  - \( w \in \mathbb{R}^m \): path-independent weights
  - \( \pi \in \mathbb{R}^m \): path-dependent weights
- GameTime operation
  - Runs systematically generated tests
  - Automatically learns program-specific environment model from measurements
  - Predicts execution times (worst-case, distribution, etc.)
  - Theoretical guarantees rely on statistical assumptions about environment
Estimating Distribution of Execution Times for a Program (using GameTime)

For StrongARM processor
Verification made easier if timing behavior of environment is (more) predictable

How do we formalize this?
Formalizing Timing Predictability

An Attempt in the $w+\pi$ model:

1. Path predictability
   Fix initial env state
   Timing of an edge (basic block) varies little as a function of the path it lies on

2. Env state predictability
   Fix program path
   Timing of an edge (or path) varies little as a function of initial platform state
Expected Payoff / Metrics

- Software increasingly integrated with the physical world
  - Physical properties will be important
  - Needs quantitative verification

- Metrics: How to assess progress?
  - Lines of code?
  - Complexity of properties?
  - Complexity of hardware?
  - Type of software: sequential → multithreaded → distributed
Risk Factors

- Lack of Open-Source Benchmarks
  - Progress in Boolean software verification was driven by wide availability of open-source software
  - More challenging for quantitative verification!
  - Heavy dependence on platform makes it more challenging to experiment on same problems

- Hardware + Software Skills
  - Students need cross-cutting skills (or willingness to learn) to work in this area
Questions for Discussion

1. **Relevance beyond embedded**: What proportion of general software problems arise from unexpected timing effects?

2. **Timing predictability contract**: What does it mean for a hardware platform to have predictable timing?

3. **Compositionality**: How to do compositional timing (quantitative) analysis?

4. **Beyond timing**: What other quantitative properties are relevant for software? Energy consumption? Reliability?

5. **Formalisms**: What modeling formalisms and/or techniques are best suited to quantitative verification? E.g. weighted automata?

6. **Synthesis**: Program synthesis for quantitative requirements? Challenge problems?