Quantitative Verification of Embedded Software: The GameTime Approach

Sanjit A. Seshia
Associate Professor
EECS Department
UC Berkeley

Students:

Collaborators: A. Rakhlin

April 2013
Verification “=” Synthesis

- Different from a definitional and complexity-theoretic viewpoint
- Similar from the viewpoint of algorithmic solution

Synthesis in Verification
  - The hard parts of verification involve synthesis “sub-tasks”

Verification in Synthesis
  - Synthesis typically involves a verification check (e.g., equivalence checking for circuits)

Artifacts Synthesized in Verification

- Inductive / auxiliary invariants
- Auxiliary specifications (e.g., pre/post-conditions, function summaries)
- Environment assumptions / Env interface specifications
- Abstraction functions / abstract models
- Interpolants
- Intermediate lemmas for compositional reasoning
- Theory lemma instances in SMT solving
- ...
Quantitative Verification of Embedded Software

Program / Env Model
Property
Verifier

∪
time, power, reliability, velocity, position, etc.

Models include quantitative parameters

Results only as accurate as the model (parameters)
Example: Deadline Properties

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

Safety-critical real-time embedded systems

Need to perform Timing Analysis
Challenge in Timing Analysis

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

NASA’s Toyota UA report (2011) mentions: “In practice...there are significant limitations” (in the state of the art in timing analysis).

CHALLENGE: ENVIRONMENT MODELING
Need a good model of the platform (processor, memory hierarchy, network, I/O devices, etc.)
This Talk

- What makes Timing Analysis Hard
- The GameTime Approach
  - Learning Program-Specific Environment Model
    - Inductive Synthesis
- Generalization: Induction + Deduction
  - Several applications in Verification & Synthesis
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 + Instruction/Data Cache
Complexity of a Timing Model: Path Space x Platform State Space

On a processor with a data cache

Timing of an edge (basic block) depends on:
- **Path** it lies on
- Initial **platform state**

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

Program CFG unrolled to a DAG

flag!=0

flag=1;
(*x)++;

flag!=0

*x += 2;
Example: Automotive Window Controller

~ 1000 lines of C code

~ $10^{16}$ paths
Outline

- What makes Timing Analysis Hard
- The GameTime Approach
  - Learning Program-Specific Environment Model
    - Inductive Synthesis
- Generalization: Induction + Deduction
  - Several applications in Verification & Synthesis
Our Approach and Contributions

Model the estimation problem as a Game
  – Tool vs. Platform

- Measurement-based, but minimal instrumentation
  – Perform *end-to-end* measurements of selected (linearly many) paths on platform

- Learn Environment Model
  – Similar to online shortest path in the ‘bandit’ setting

- Online, randomized algorithm: GameTime
  – Theoretical guarantee: can predict worst-case path with arbitrarily high probability under model assumptions

- Uses satisfiability modulo theories (SMT) solvers for test generation
The Game Formulation

- Complexity ‘=’ Path Space x Platform State Space
  (controllable)    (uncontrollable)

- Model as a 2-player Game: Tool vs. Platform
  - Tool selects program paths
  - Platform ‘selects’ its state (possibly adversarially)

- Questions:
  - What is a good class of platform models?
  - How to select paths so that we can learn an accurate platform model by executing those?
Platform Model

Platform selects weights for edges of the CFG

Models path-independent timing

Nominal weight on edge of unrolled CFG +

Path-specific perturbation

Models path-dependent timing
A Path is a Vector $\mathbf{x} \in \{0,1\}^m$

$(m = \#\text{edges})$

**Insight:**
Only need to sample a Basis of the space of paths

\[
\begin{align*}
\mathbf{x}_1 &= (1,1,1,0,0,1,1,0,0,1) \\
\mathbf{x}_2 &= (1,0,0,1,1,0,0,1,1,1) \\
\mathbf{x}_3 &= (1,1,1,0,0,0,0,1,1,1) \\
\mathbf{x}_4 &= (1,0,0,1,1,1,1,0,0,1)
\end{align*}
\]
Useful to compute certain special bases called “barycentric spanners”

\[ x_1 = (1,1,1,0,0,1,1,0,0,1) \]
\[ x_2 = (1,0,0,1,1,0,0,1,1,1) \]
\[ x_3 = (1,1,1,0,0,0,0,1,1,1) \]
\[ x_4 = (1,0,0,1,1,1,1,0,0,1) \]

\[ x_4 = x_1 + x_2 - x_3 \]
Timing Analysis Game (Our Model)

Played over several rounds $t = 1, 2, 3, \ldots, \tau$

At each round $t$:

- Tool picks $x_t$
- Platform picks $w_t$

Tool observes $l_t = x_t \cdot f(w_t + \pi_t)$

At round $\tau$:

- Tool makes prediction (longest path $x^*_\tau$)
- Tool wins iff its prediction is correct

Example:

$(-1, -1, -1, -1)$

$(5+7+1+11) - 4 = 20$
Theorem about Estimating Distribution (pictorial view)

\[ \mathcal{X} \in O(b\,\mu_{\text{max}}) \]

Mean Perturbation Assumption: \( \forall x \in 5 \text{ Paths} \) \[ |\mathbb{H} [x, \pi_t]| \preceq \mu_{\text{max}} \]

\( \xi \) is \( O(b\,\mu_{\text{max}}) \)
Some Experimental Results
(details in ICCAD’08, ACM TECS, FMCAD’11 papers)

- **GameTime is Efficient**
  - E.g.: $7 \times 10^{16}$ total paths vs. $< 200$ basis paths

- **Accurately predicts WCET for complex platforms**
  - I & D caches, pipeline, branch prediction, …

- **Basis paths effectively encode information about timing of other paths**
  - Found paths 25% longer than sampled basis

- **GameTime can accurately estimate the distribution of execution times with few measurements**
  - Measure basis paths, predict other paths

(details in ICCAD’08, ACM TECS, FMCAD’11 papers)
Recent Results

- **Timing analysis of interrupt-driven programs** [FMCAD 2011]
  - Idea: context-bounded analysis + GameTime

- **Energy estimation** on embedded devices
  - Use GameTime algorithm with iCount hardware [P. Dutta et al.]
Generalizing the GameTime Approach

- Identify “Synthesis Sub-task” in verification
  - Environment Modeling
- Make a Structure Hypothesis
  - \( w + \pi \) model for the platform
- Use Inductive Inference
  - learning from measurements
- Combine with Deductive Reasoning
  - SAT/SMT solving for test generation

Induction + Deduction + Structure

Other Projects

- **Switching logic synthesis** for hybrid systems
  - For safety and optimality
  - [Jha et al., ICCPS 2010, EMSOFT 2011]

- **Program synthesis**, malware analysis
  - [Jha et al., ICSE 2010]

- **Synthesizing fixed-point code** from floating-point specifications
  - [Jha & Seshia, 2011]

- **Controller synthesis from temporal logic**
  - [Li et al., MEMOCODE 2011]

- **Hardware verification**
  - [Brady et al., FMCAD 2011]