Voting Machines and Automotive Software: Explorations with SMT at Scale

Sanjit A. Seshia

EECS Department
UC Berkeley

Joint work with: Bryan Brady, Randy Bryant, Susmit Jha, Jon Kotker, John O’Leary, Alexander Rakhlin, Cynthia Sturton, David Wagner

March 2011
Three Stories

- **Verified Voting Machine**
  - High-confidence Interactive System
  - \textit{SMT solving can exponentially reduce the number of UI tests by humans}

- **GameTime**
  - Timing Analysis of Embedded Software
  - \textit{SMT solving can enable systematic measurement-based timing analysis}

- **UCLID / ATLAS**
  - Verification of High-Level Hardware Designs
  - \textit{SMT solving sometimes needs help! (automatic abstraction to suitable theories)}
Electronic Voting Machines

2010 U.S. elections statistics [verifiedvoting.org]
- 25% of registered voters had to use paperless electronic voting machines
- In 11 states, paperless voting accounts for most or all Election Day ballots

Concerns about correctness and security
Jefferson County Voters Continue To Raise Concerns About Voting Machines
“...voters complained that when they selected a particular candidate, another candidate’s name would light up.”
*KDFM-TV Channel Six News. Oct. 28, 2006*

Can You Count on Voting Machines?
“Sliding finger bug on the Diebold AccuVote-TSX ... machine would crash every few hundred ballots”
A Typical DRE

- **Contest**: a particular race on the ballot
  - E.g., Presidential
  - $k$ choices, pick $l$

- **Voter session**: a sequence of contests
  - Navigate back and forth

- **Cast**: commit all choices for all contests
  - The last step of a voter session
Our Contribution

- **Testing by humans + formal verification** can prove a voting machine will work correctly on election day.

- Designed a simplified voting machine and proved its correctness using formal methods:
  - Direct recording electronic voting machine (DRE) synthesized onto an FPGA
  - Verification by Model checking and SMT solving
  - **Finite, polynomial number of tests** (to be conducted by humans)

Correctness: Trace Equivalence

How to model?

Tester

Contest 1, no selections

Contest 1, “Alice”

Implementation

(1, ∅)

(1, {“Alice”})
Testing: What Tests are Sufficient?

What sequences \((b_1, b_2, b_3, \ldots, \text{cast})\) are sufficient for testing?

Problem: Infinitely many input sequences!
Consider for a single contest: Alice (A) vs. Bob (B)
Formal Verification to the Rescue

Verify the following properties on the code:

P0. The DRE implementation is deterministic

P1. Each unique output screen represents a unique internal state
   – output display function is injective (1-1) function of selection state and contest number

P2. The final cast vote record accurately reflects the selection state
Multiple Contests: Exponential Blowup

N contests, 1-of-k choice in each contest → $k^N$ total combinations

An SMT-based verification step can reduce the number of choices to simply $N \times k$!
Additional Properties to be Verified

P3. Contests are Independent: Updating the state of one contest has no effect on any other contest

P4. Navigation does not affect Selection: A navigation button does not affect the selection state of any contest

P5. Selection does not affect Navigation: A selection button does not navigate to a new contest
Verifying Independence/Determinism

Verify that a variable $v$ is a function of $W = \{w_1, w_2, \ldots w_k\}$ AND nothing else

- $\phi(S, S', I, O) \triangleq S' = \delta(S, I) \land O = \rho(S)$
- Check validity of the formula
  $$\{ \phi(S_1, S_1', I_1, O_1) \land \phi(S_2, S_2', I_2, O_2) \land \forall w \in W. \ w_1 = w_2 \}$$
  $$\Rightarrow v_1' = v_2'$$

- Encode next-state and output functions as logical formulas
- Check that value of $v$ is not affected by changes to variables other than $W$ (consider two runs in which $W$ variables have same initial value)
Experience with SMT Solvers

- **Original HW implementation**
  - Small screen, rendered in hardware
  - Bit-vector SMT solvers (circa 2009) worked fine
    - Beaver (developed in my group)

- **Moved to combined HW-SW implementation**
  - Larger screen, more complex GUI, rendered in software
  - Bit-vector solvers no longer scaled
  - Solution: Use quantified linear arithmetic with uninterpreted functions and arrays; compositional reasoning
    - 2009: Still too difficult for SMT solvers, Z3 returned “unknown”
    - 2011: Progress! Z3 solves it.
Timing Analysis of Embedded Software

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

Can the pacemaker software trigger a pace more frequently than prescribed?
The Challenge of Timing Analysis

Several timing analysis problems:

- Worst-case execution time (WCET) estimation
- **Threshold** property: can a program take more/less time than it is supposed to?
- Estimating **distribution** of execution times
- **Software-in-the-loop simulation**: predict execution time of particular program path

Challenge: Platform Modeling
Factors affecting Execution Time

- Processor (pipelining, branch prediction, …)
- Caches
- Virtual memory
- Dynamic dispatch
- Power management (voltage scaling)
- Memory management (garbage collection)
- Just-in-time (JIT) compilation
- Multitasking (threads and processes)
- Networking
- …
Current State-of-the-art for Timing Analysis

- Program = Sequential, terminating program
- Runs uninterrupted

PROBLEM: Can take several man-months to construct!

Also: limited to extreme-case analysis

- Platform = Single-core Processor + Data/Instruction Cache
Our Approach: GameTime

- Automatically infer a program-specific timing model of the platform from systematic measurements

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

- SMT solver generates tests for chosen paths
  - Typically: conjunctions of atomic formulas
  - Quantifier-free BV + UF$s + Arrays
  - Less need for incrementality (don’t incrementally grow a path formula, less sharing amongst path formulas)
The GameTime Approach: Overview

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

- PROGRAM
- CONTROL-FLOW GRAPH
- EXTRACT BASIS PATHS
- SMT SOLVER GENERATES TEST INPUTS
- LEARNING ALGORITHM
- MEASURE EXECUTION TIMES
- PREDICT TIMING PROPERTIES (worst-case, distribution, etc.)

Example: Automotive Window Controller

- ~ 1000 lines of C code
- $7 \times 10^{16}$ program paths

Number of basis paths explored by GameTime: < 200

SMT queries: Max time about a second

Accurately predicts lengths of non-basis paths
Term-Level Modeling for H/W Verification

- **Data Abstraction**: View Data as Symbolic “Terms”

  \[ x_0 \quad x_1 \quad x_2 \quad \ldots \quad x_{n-1} \quad \Rightarrow \quad x \]

- **Function Abstraction**: Abstract Functional Units as Uninterpreted (partially-interpreted) functions

  \[ \text{ALU} \quad \Rightarrow \quad f \]
Modeling for Hardware Verification

- Symbolic (e.g. integer) data
- Uninterpreted functions & predicates
  Bit-vectors + uninterpreted functions + arrays + integers

- Fixed-width words of bits
- Standard arithmetic and logical operators
  Bit-vectors (+ arrays)

- Individual bits
- Boolean operations
  Bit Level
Impact of Term-Level Abstraction

- ATLAS: Automatic Term-Level Abstraction

- Abstracting to term level generates much easier SMT problems

- Experience on processor and low-power designs
  - QF_BV $\rightarrow$ QF_AUFBV
  - Speedup of 5X-100X (using all leading solvers, this number for Boolector)

Other Explorations with SMT

- **Program Synthesis from I/O Examples** [ICSE’10]
  - Applied to reverse engineering of malware
  - SMT solvers used to generate examples and candidate programs

- **CalCS: SMT solving for non-linear convex constraints** [FMCAD’10]
  - Applied to verification of hybrid systems

- **Verification and Synthesis of Network-on-Chip Designs** [DATE’11, DAC’11]

http://www.eecs.berkeley.edu/~sseshia
http://uclid.eecs.berkeley.edu