S2W: Verification using Small and Short Worlds

Rohit Sinha, Cynthia Sturton, Petros Maniatis, Sanjit Seshia, David Wagner

http://uclid.eecs.berkeley.edu/s2w

University of California, Berkeley

October 23, 2012
Hypervisors and Emulators

Hypervisor (or Virtual Machine Monitor):
- Virtualization layer that runs at the highest CPU privilege level
- Enables concurrent and isolated execution of multiple guest OSes

CPU Emulator
- Emulates a target CPU's behavior on a host CPU platform
Why verify hypervisors?

Virtualization is important for:
- Cloud computing
- Hosting legacy applications

Can we trust hypervisors?
- Virtualization code is complex
- Exploits demonstrated (e.g. Xen, Xbox 360)
Verification Challenges

- Large data structures
  - e.g. TLB, page tables, interrupt descriptor table, etc.

- Highly optimized implementation in C/C++
  - Need to generate and validate abstract model

- Deep properties such as equivalence and refinement
  - Need to reason about control and data
Verification Challenges

Example: **Bochs TLB + Paging**
Translates virtual to physical address

- Large data structures: TLB, 2-level page tables
- Equivalence Property
  - Without TLB: Invoke page walk to retrieve translation
  - With TLB: first check TLB for cached translation. Invoke page walk on TLB miss. (Bochs implementation)
S2W: Overview

- Construct a term level model of the C++ implementation
  - Large tables modeled as unbounded arrays, operations modeled using uninterpreted functions

- Bounded Model Checking (BMC) finds counter-examples up to a certain bound
  - Sound if reachability diameter known

- S2W: An abstraction based methodology
  - Construct abstract model
  - Compute reachability diameter of the abstract model
  - BMC until reachability diameter
S2W: Contribution

- Semi-automatic procedure for verifying systems with unbounded data structures
  - (Term-level) abstraction-based model checking
  - Heuristics for creating an abstract model
  - Heuristics for bounding the reachability diameter

- Evaluation: Bochs x86 paging, Shadow Paging, SecVisor, CAM, etc.
## Related Work

<table>
<thead>
<tr>
<th>Work</th>
<th>Application</th>
<th>Technique</th>
<th>Expressiveness</th>
<th>Guarantee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alkassar et al: Verification of Microsoft Hyper-V</td>
<td>Hypervisors</td>
<td>Deductive (VCC)</td>
<td>general</td>
<td>Sound, not complete</td>
</tr>
<tr>
<td>Franklin et al: Scalable Parametric Verification</td>
<td>Hypervisors</td>
<td>Small Model Theorem</td>
<td>restricted</td>
<td>Sound, complete</td>
</tr>
<tr>
<td>Klein et al: seL4: Formally Verified OS kernel</td>
<td>Microkernel</td>
<td>Theorem Proving (Isabelle)</td>
<td>general</td>
<td>Sound, not complete</td>
</tr>
<tr>
<td>Per Bjesse: Word-Level Sequential Memory Abstraction</td>
<td>Hardware</td>
<td>Small Model + CEGAR</td>
<td>restricted</td>
<td>Sound, not complete</td>
</tr>
<tr>
<td>Steve German: Theory of Abstraction for Arrays</td>
<td>Hardware</td>
<td>Static analysis + Model checking</td>
<td>restricted</td>
<td>Sound, complete</td>
</tr>
<tr>
<td>S2W</td>
<td>Hypervisors, Emulators</td>
<td>Abstraction + BMC</td>
<td>general</td>
<td>Sound, not complete</td>
</tr>
</tbody>
</table>
**Problem Definition**

- **System** $S$
  - Booleans, predicates, bit-vectors, uninterpreted functions, arrays
- **Environment** $E$
  - provides inputs from $S$ and consumes its outputs
  - arbitrary inputs to $S$ at each step
- **Model under verification** $M = S || E$
- **Safety property** $G \Phi$
  - $\Phi$ of the form $\forall x_1, x_2, \ldots x_n. \Phi(x_1, x_2, \ldots x_n)$
  - bitvectors $x_1, x_2, \ldots x_n$ too large to exhaustively case split

**Large Data Safety Verification**

Does $M$ satisfy $G \Phi$?
**S2W: Methodology**

**Model** $\mathcal{M}$ → **Term level model** → **Property** $\Phi$

- $\Phi$ inductive invariant? → yes
- no → small world
  - short world
  - BMC
    - $\Phi$?
      - valid → $\mathcal{M} \models G \Phi$
      - cex → Spurious?
        - yes → update abstraction
        - no → $\mathcal{M} \not\models G \Phi$

1. Induction
2. Small world abstraction
3. Compute short world bound
4. BMC

**Verification with S2W is sound, but not complete**
S2W first attempts 1-step induction on $\mathcal{M}$.

**Base case**

$$\text{Init}(s) \rightarrow \Phi(s) \quad (1)$$

**Induction**

$$\Phi(s) \land \mathcal{R}(s, s') \rightarrow \Phi(s') \quad (2)$$
Toy Example

**Inputs:** addr

**State:** mem, cache

**Transition:**
next(cache.addr) = addr
next(cache.data) = ITE(addr = cache.addr, cache.data, mem[addr])

**Property \( \Phi \)**

\[
\Phi \equiv \forall x. (addr = x) \rightarrow (\text{cache.addr} = addr \land \text{cache.addr} \neq 0) \rightarrow \text{cache.data} = \text{mem}[addr])
\]
Small World Abstraction

**Localization Abstraction**: Abstract away some state

**Dependence Set** $\mathcal{U}$

$\mathcal{U}$ is a set of expressions over state variables and inputs such that $\phi$ depends only on expressions in $\mathcal{U}$.

Derive $\mathcal{U}$ syntactically by traversing the expression graph of $\phi$

Abstract Model $\hat{\mathcal{M}}$ where:

- Terms in $\mathcal{U}$ are precisely modeled
- All other state havoced at each step

No spurious counter-example in our case studies.
Step 1: Fresh parameter $a$ for $x$

$$\phi(a) \triangleq (addr = a) \rightarrow ((cache.addr = addr \land cache.addr \neq 0) \rightarrow cache.data = mem[addr])$$ (4)

Step 2: Derive $\mathcal{U}$ syntactically

$$\mathcal{U} = \{cache.addr, cache.data, mem[a]\}$$ (5)
Computing Short World bound of $\hat{M}$

Reachability diameter $k$

For every trace $s_0, \ldots, s_{k+1}$ via input sequence $a_1, \ldots, a_{k+1}$,

there exists a trace $s'_0, \ldots, s'_k$ via input sequence $b_1, \ldots, b_k$

such that $s_{k+1} = s'_k$.

Alternating quantifiers!
To verify a hypothesized bound $k$:

For every trace $s_0, \ldots, s_{k+1}$ via input sequence $a_1, \ldots, a_{k+1}$,

there exists a trace $s'_0, \ldots, s'_k$ via k-length subsequence of $a_1, \ldots, a_{k+1}$

such that $s_{k+1} = s'_k$. 

Reachability bound
Toy Example [Subsequence Heuristic]

Reachability diameter
k = 2

Example:
- cache.addr_3 = cache.addr_2'
- cache.data_3 = cache.data_2'
- mem[a]_3 = mem[a]_2'
Another approach to instantiating existentially quantified variables in the reachability check

What is a gadget?
A sequence of symbolic state transitions that reaches a subset of all reachable states

Universal gadget set
Any reachable state of $\mathcal{M}$ is reached by at least one gadget in the universal gadget set.

Length of longest gadget bounds reachability diameter $k$. 
Computing Short World bound via Gadget Heuristic

Reachability bound

For every reachable state, there exists a gadget that reaches it.

$k$ is length of the longest gadget
Toy Example [Gadgets]

1. Gadget $g_1$:  
   \[\text{init}(\text{mem}[a]) = 0\]  
   followed by 1 step transition of $\hat{M}$

2. Gadget $g_2$:  
   \[\text{init}(\text{mem}[a]) = 1\]  
   followed by 1 step transition of $\hat{M}$

Reachability diameter $k = 1$
Bounded Model Checking

- Short world computes reachability diameter $k$ of $\hat{M}$

- Run BMC for $k$ steps:
  - If $\hat{M}$ satisfies $\Phi$ for $k$ steps, then $M \models G\Phi$.
  - Else counter-example...
    - If valid counter-example, then $M \not\models G\Phi$.
    - If spurious counter-example, then refine abstraction (expand $U$) and restart.
S2W can verify components of real hypervisors and emulators.

All experiments are performed using UCLID verifier with Plingeling SAT solver backend.
Bochs TLB + Paging unit

Property $\Phi$

$$\Phi_6 \equiv \forall v, p, r. \ (vaddr = v \land pl = p \land rwx = r) \rightarrow$$

legal $\rightarrow ((\text{pagefault}_{\text{TLB}} \Leftrightarrow \text{pagefault}_{\text{noTLB}}) \land$\n
$$\neg \text{pagefault}_{\text{noTLB}} \rightarrow (paddr_{\text{noTLB}} = paddr_{\text{TLB}}))$$

Dependence set $\mathcal{U}$

$$\mathcal{U} = \{\text{legal}, \text{TLB}[v], \text{mem}[v], \text{mem[mem[v]]}\}$$ (7)

subsequence check takes 1.5 hours, BMC takes 28 minutes
Shadow Paging

Property $\Phi$

$$\Phi_1 \doteq \forall i. (sPDT[i].p \land sPDT[i].pse) \rightarrow sPDT[i].addr < \text{LIMIT} \quad (8)$$

$$\Phi_2 \doteq \forall i, j. (sPDT[i].p \land \neg sPDT[i].pse \land sPT[j].p) \rightarrow sPT[j].addr < \text{LIMIT} \quad (9)$$

Dependence set $\mathcal{U}$

$$\mathcal{U} = \{sPDT[a_i], sPT[a_j]\} \quad (10)$$

diameter $k \leq 4$
gadget check takes 60 seconds, BMC takes 5 seconds
Property $\Phi$

$$
\Phi \equiv \forall a. \ (addr = a) \rightarrow legal \rightarrow \\
(out_{cam}^{\text{present}} \rightarrow (out_{cam}^{\text{data}} = out_{mem}^{\text{data}})) 
$$  \hspace{1cm} (11)

Dependence set $U$

$$
U = \{ \text{legal}, \ cam[map[a]], \ mem[a], \ map[a] \} 
$$  \hspace{1cm} (12)

$\text{diameter } k \leq 5$

subsequence check takes 5 seconds, BMC takes 4 seconds
And Some Other Hypervisors...

**SecVisor**
- Execution integrity: Only approved code runs in Kernel mode
- Proved via 1-step induction in 2 seconds

**sHype**
- Chinese Wall policy as implemented in Xen hypervisor
- Proved via 1-step induction in 6 seconds

---

Summary

• Contribution: S2W, a semi-automatic procedure for verifying systems with unbounded data structures
  • Abstraction-based model checking
  • Heuristics for creating an abstract model
  • Heuristics for computing the reachability diameter bound

• Ongoing work:
  • Model Validation
  • Generating term level models from C++
  • Automatic construction of gadgets

• Case studies available for download at uclid.eecs.berkeley.edu/s2w