Towards Building Trustworthy Systems Software

Ronghui Gu
Computer Science Department
Columbia University
Motivation

- Software Systems Run Everywhere (we have to trust)

- Program Errors (untrusted)
Motivation

Program Errors
Toyota’s **Killer System**

- does not meet **spec**

- a single stack overflow bug
Can we rely on more tests?
Program testing can be used to show the presence of bugs, but never to show their absence.

— Edsger Dijkstra
Test v.s. Formal Verification

Specification

\((x+1)^2\)

Code
Test v.s. Formal Verification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x+1)^2)</td>
<td>(x + x*x + x + 1)</td>
</tr>
<tr>
<td>Specification</td>
<td>Code</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>((x+1)^2)</td>
<td>(x + x\times x + x + 1)</td>
</tr>
</tbody>
</table>

- \(x = 0\)
- \(x = 1\)
<table>
<thead>
<tr>
<th>Specification</th>
<th>Code</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x+1)^2)</td>
<td>(x + x*x + x + 1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 + 0 + 0 + 1 = 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x = 0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(x = 1)</td>
<td></td>
</tr>
</tbody>
</table>
## Test v.s. Formal Verification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Code</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x+1)^2)</td>
<td>(x + x\times x + x + 1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 + 0 + 0 + 1 = 1</td>
<td>(x = 0)</td>
</tr>
<tr>
<td>4</td>
<td>1 + 1 + 1 + 1 = 4</td>
<td>(x = 1)</td>
</tr>
</tbody>
</table>
### Test v.s. Formal Verification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Code</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Specification

$$(x+1)^2$$

<table>
<thead>
<tr>
<th>$x$</th>
<th>Code</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x + x + x + 1$</td>
<td>$0 + 0 + 0 + 1 = 1$</td>
</tr>
<tr>
<td>4</td>
<td>$x + x + x + 1$</td>
<td>$1 + 1 + 1 + 1 = 4$</td>
</tr>
</tbody>
</table>
Formal methods are the only reliable way to achieve security and privacy in computer systems.

— NSF SFM Report [2016]

Formal Verification

- mathematically prove
- program meets specification
- under all inputs
- under all execution

Complete formal verification is the only known way to guarantee that a system is free of programming errors.
### Test v.s. Formal Verification

<table>
<thead>
<tr>
<th>Specification</th>
<th>Code</th>
<th>Formal Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>((x + 1)^2)</td>
<td>(x + x^2 + x + 1)</td>
<td>associative law</td>
</tr>
<tr>
<td></td>
<td>((x + x^2) + (x + 1))</td>
<td>factorising</td>
</tr>
<tr>
<td></td>
<td>(x(1 + x) + (x + 1))</td>
<td>commutative law</td>
</tr>
<tr>
<td></td>
<td>(x(x + 1) + (x + 1))</td>
<td>factorising</td>
</tr>
<tr>
<td></td>
<td>((x + 1)^2)</td>
<td>definition</td>
</tr>
</tbody>
</table>
Specification: the program will always terminate?
The **halting problem** is the problem of **deciding** whether a given program will ever **halt** or not.
Alan Turing proved (in 1936) that a general algorithm to solve the halting problem for all possible programs cannot exist.
Halting Problem

paradox(p):= if (v(p(p))) then true else infloop()

if paradox(paradox) loops forever, then v(paradox(paradox))=true. Thus, paradox(paradox) terminates.

Contradiction.
Halting Problem

paradox(p) := if \( v(p(p)) \) then true else infloop()

if paradox(paradox) halts, then \( v(paradox(paradox)) = \text{false} \). Thus, paradox(paradox) loops forever. Contradiction.
Is there a simple, specific program that we cannot decide whether it will ever halt or not?
Collatz Conjecture

For an arbitrary positive integer $i$

- If $i$ is even, divide it by 2.
- If $i$ is odd, triple it and add 1.

- 5, 16, 8, 4, 2, 1
- 13, 40, 20, 10, 5, …

```c
void collatz (int i) {
    while (i > 1) {
        if (i is even) 
            i = i / 2;
        else
            i = 3 * i + 1;
    }
}
```
Collatz Conjecture

\[ i = 6171 \]
takes 261 steps
Limitations

All the programs
Limitations

All the programs

- Paris metro line 14 (1998)
- Flight control of A380 (2005)
- CompCert C compiler (2005, Coq)
- seL4 sequential kernel (2009)

What about concurrent programs?
Some of the significant results that were accomplished using Coq are proofs for the four color theorem, the development of CompCert (a fully verified compiler for C), the development at Harvard of a verified version of Google's software fault isolation, and most recent, the fully specified and verified hypervisor OS kernel CertiKOS.

— Acm
Informally, if \( M_1 \rightarrow L_1 \) and \( M_2 \rightarrow L_2 \), the functions of \( M_2 \) can be used to tie our notion of behavior refinement into the calculus. This allows us to break down the problem of verifying a layer implementations of the union of the two interfaces.

The judgment \( L_1 \vdash_R M : L_2 \) captures all we need to know about \( M \) over \( L_1 \). Any property about \( M \) can be proved using \( L_2 \) alone. No need to look at \( M \) again.

This rule can be combined with the rules of the calculus to faithfully implement \( L_2 \) on top of \( L_1 \). Similarly, given a concrete language semantics, we will want to define a corresponding deep specification in \( L_1 \). The semantics of a module can be understood as the effect of the typing rules with respect to this interpretation.
Deep Specification

\[ L_1 \quad + \quad L_2 \quad = \quad L_1 + L_2 \]

\[ R_1 \quad M_1 \quad R_1 \quad M_2 \]

\[ L_0 \quad R_0 \quad M_0 \quad L_0 \]
Deep Specification

\[
\begin{array}{c}
L_1 \oplus L_2 \\
R_1 \\
M_1 \oplus M_2 \\
L \\
\oplus \\
L \\
\end{array}
\]

\[
\begin{array}{c}
R_0 \\
M_0 \\
L_0 \\
L \\
\end{array}
\]
Deep Specification

CPU0

R
L
M
L
R
M
L
M
L

CPU1

R'
L'
M'
L'
R'
M'
L'
M'
L'
Deep Specification

T SysCall Layer
(pe, ikern, ihost, ipt, AT, PT, ptp, pbit, kctxp, Htcpb, Htqp, cid, chanp, uctxp, npt, hctx, vmst)

Thread Wakeup/Kill/Sleep/Yield  Pt_Read  Get/Set/Uctx  Palloc/Free  Cid_Get
Sys Channel Send/Recv/Wait/Check  Sys_Yield  Sys_Get/Exit Reason  Sys_Get/Eip
Sys Check Shadow/Pending Event  SysProc Create  Sys_Set_Seg  Sys_Inject
Sys Get Exit Io Width/Port/Rep/Str/Write/Eip  Sys_Set_Interrupt  Sys_Npt_Instr
Vmcbinit  PageFault Handler  Sys Reg Get/Set  Sys Sync  Sys Run  Vm Exit

T SysCall Layer
(mm/proc/virt.abs)

T Trap Layer
(mm/proc/virt.abs)

T TrapArg Layer
(mm/proc/virt.abs)

V SVM Layer
(mm/proc.abs, npt, hctx, vmst)
How about blockchain systems?

All the programs

Blockchain systems

Verified
How about blockchain systems?

Blockchain Market Cap
$14 billion (12/2016) $300 billion (12/2017)

Number of smart contracts
0.1 million (6/2016) 1 million (12/2017)

No need to trust a single, centralized authority
How about blockchain systems?

“

In math we trust.

— SC Zhang
Implementations are error-prone
TheDAO: 1 bug (double-spend attack), $50 million lost
BEC: 1 bug (integer overflow), $5 billion scam
EduCoin: 1 bug (integer overflow), 2 billion token lost

Huge attack benefits
$630 million has been lost to hackers in 12/2017

Smart contracts are open-sourced to hackers and hard to fix once deployed
CertiK - Invested by SC’s DanhuaVC
Protecting Smart Contracts via DeepSpec
Protecting Smart Contracts via DeepSpec

Smart contract

Contract labeled with specifications

Labeling

Smart labeling using deep learning techniques

Customized labeling by verification experts
Decomposing a complex verification (or proof) task into smaller ones that are easy to solve via a layer-based approach.
Protecting Smart Contracts via DeepSpec

Composing the proofs to form an end-to-end guarantee
Detecting BEC Bugs

![CertiK Report](image)

**CertiK Report**

**Score**: 0

**contract Beauty**

```solidity
contract Beauty {
    using SafeMath for uint256;

    event Transfer(address sender, address receiver, uint256 value);

    mapping(address => uint256) balances;
    /*@CTE overflow*/
    @tag assume_completion
    @post _has_overflow == false
    /

    This code violates the specification
    Counter Example:
    Before Execution:
    _value = 0x2000000000000000000000000000000000000000000000000000000000000000
    _has_assertion_failure = False
    _has_overflow = False
    
    After Execution:
    cnt = 0x0
    _value = 0x2000000000000000000000000000000000000000000000000000000000000000
    _has_assertion_failure = False
    _has_overflow = True

    function batchTransfer(address[] _receivers, uint256 _value)
    {
        public returns (bool) {
            uint cnt = _receivers.length;
            uint256 amount = uint256(cnt) * _value;
            require(amount <= 20);
            for (uint i = 0; i < cnt; i++) {
                balances[msg.sender] = balances[msg.sender].sub(amount);
            }
        }
    }
    }

Follow Counter Example
CertiK - Invested by SC's DanhuaVC

- 170+ Audits
- 100K+ LOC Audited
- $4B+ Secured
CertiK - Invested by SC’s DanhuaVC

All the programs

Blockchain systems

Verified
CertiK - Invested by SC’s DanhuaVC

All the programs

Blockchain systems

Verified
What about quantum programs (joint work with IBM and UChicago)?

The gate computation of GHZ state and its implementation are shown as below:

\[
\text{module GHZ (qubit* q, int n)} \\
\text{@pre n = length q} \\
\text{@pre n ≥ 1} \\
\text{@pre \( \otimes q[0, ..., n] = |0\rangle^{\otimes n} \)} \\
\text{@post \( \otimes q[0, ..., n] = \frac{1}{\sqrt{2}} |0\rangle^{\otimes n} + \frac{1}{\sqrt{2}} |1\rangle^{\otimes n} \)} \\
\{ \\
\text{H(q, n, 0);} \\
\text{for (int i = 1; i < n; i ++)} \\
\text{\@inv \( \otimes q[0, ..., i] = \frac{1}{\sqrt{2}} |0\rangle^{\otimes i} + \frac{1}{\sqrt{2}} |1\rangle^{\otimes i} \)} \\
\text{\@inv \( \otimes q[i, ..., n] = |0\rangle^{\otimes n-i} \)} \\
\text{\@inv 1 ≤ i ≤ n} \\
\{ \\
\text{CNOT(q, n, i-1, i);} \\
\} \\
\}
\]
Proved by induction

\[ i = 1 \quad \Rightarrow \quad \bigotimes q_1[0, \ldots, i] = H(|0\rangle) = \frac{1}{\sqrt{2}} |0\rangle \otimes i + \frac{1}{\sqrt{2}} |1\rangle \otimes i \]
\( \otimes q_i[0, \ldots, i] = \frac{1}{\sqrt{2}} |0\rangle \otimes i + \frac{1}{\sqrt{2}} |1\rangle \otimes i \) and \( q_i[i] = |0\rangle \)

**CNOT** \((q, n, i-1, i)\)

\[ \otimes q_{i+1}[0, \ldots, i+1] = \frac{1}{\sqrt{2}} |0\rangle \otimes i+1 + \frac{1}{\sqrt{2}} |1\rangle \otimes i+1 \]
Certified Quantum Programs

All the programs

Verified

Quantum programs
Certified Quantum Programs

- Quantum programs
- All the programs
- Verified
No consistent system is capable of proving all truths about the arithmetic of the natural numbers.
Einstein said he went to his office at the Institute for Advanced Study “just to have the privilege of walking home with Kurt Gödel.”