CertiCore: Verified ucore by Serval

Zhilei Han, Xingyu Xie

CST, Tsinghua University

March 2020

Contents

Contents

What is verification?

To verify your codes satisfy your specification.

```
// An example of Dafny
method Sort(a: int, b: int, c: int)
        returns (x: int, y: int, z: int)
    ensures x <= y <= z
{
    x, y, z := a, b, c;
    if z < y { y, z := z, y; }
    if y < x { x, y := y, x; }
    if z < y { y, z := z, y; }
}
```

Three methodologies of software verification

Methodology	Prover	Verified software
Interactive	Coq, Isabelle/HOL	seL4, CertiKOS, ...
Auto-active	Dafny & Boogie	Ironclad, Komodo, ...
Push-button	Serval	Yggdrasil, Hyperkernel, ...

Table: Three Methodologies in software verification

Contents

Overview of Serval

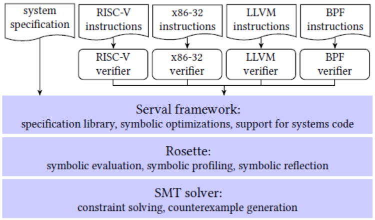
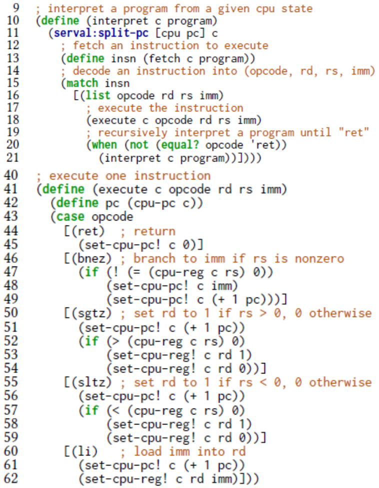


Figure: The Serval verification stack. Curved boxes denote verification input and rounded-corner boxes denote verifiers.

Limitations of verified software

- Finite interfaces
- Bounded loops
- Interrupts disabled in privileged mode

Symbolic interpreter as verifier



```
; interpret a program from a given cpu state
(define (interpret c program)
  (serval:split-pc [cpu pc] c
    ; fetch an instruction to execute
    (define insn (fetch c program))
    ; decode an instruction into (opcode, rd, rs, imm)
    (match insn
      [(list opcode rd rs imm)
        ; execute the instruction
        (execute c opcode rd rs imm)
        ; recursively interpret a program until "ret"
        (when (not (equal? opcode 'ret))
          (interpret c program))])))
; execute one instruction
(define (execute c opcode rd rs imm)
  (define pc (cpu-pc c))
  (case opcode
    [(ret)  ; return
      (set-cpu-pc! c 0)]
    [(bnez) ; branch to imm if rs is nonzero
      (if (! (= (cpu-reg c rs) 0))
          (set-cpu-pc! c imm)
          (set-cpu-pc! c (+ 1 pc)))]
    [(sgtz) ; set rd to 1 if rs > 0, 0 otherwise
      (set-cpu-pc! c (+ 1 pc))
      (if (> (cpu-reg c rs) 0)
          (set-cpu-reg! c rd 1)
          (set-cpu-reg! c rd 0))]
    [(sltz) ; set rd to 1 if rs < 0, 0 otherwise
      (set-cpu-pc! c (+ 1 pc))
      (if (< (cpu-reg c rs) 0)
          (set-cpu-reg! c rd 1)
          (set-cpu-reg! c rd 0))]
    [(li)   ; load imm into rd
      (set-cpu-pc! c (+ 1 pc))
      (set-cpu-reg! c rd imm)]))
```

Figure: A fragment of ToyRISC interpreter using Serval (in Rosette)

Symbolic evaluation

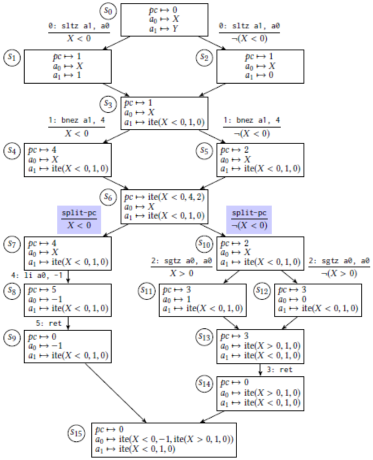

S_0 $pc \mapsto 0$ $a_0 \mapsto X$ $a_1 \mapsto Y$
0: sltz a1, a0 $X < 0$
0: sltz a1, a0 $\neg(X < 0)$
S_1 $pc \mapsto 1$ $a_0 \mapsto X$ $a_1 \mapsto 1$
S_2 $pc \mapsto 1$ $a_0 \mapsto X$ $a_1 \mapsto 0$
S_3 $pc \mapsto 1$ $a_0 \mapsto X$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
1: bnez a1, 4 $X < 0$
1: bnez a1, 4 $\neg(X < 0)$
S_4 $pc \mapsto 4$ $a_0 \mapsto X$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_5 $pc \mapsto 2$ $a_0 \mapsto X$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_6 $pc \mapsto \mathrm{ite}(X < 0, 4, 2)$ $a_0 \mapsto X$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
split-pc $X < 0$
split-pc $\neg(X < 0)$
S_7 $pc \mapsto 4$ $a_0 \mapsto X$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_{10} $pc \mapsto 2$ $a_0 \mapsto X$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
4: li a0, -1
2: sgtz a0, a0 $X > 0$
2: sgtz a0, a0 $\neg(X > 0)$
S_8 $pc \mapsto 5$ $a_0 \mapsto -1$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_{11} $pc \mapsto 3$ $a_0 \mapsto 1$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_{12} $pc \mapsto 3$ $a_0 \mapsto 0$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
5: ret
S_9 $pc \mapsto 0$ $a_0 \mapsto -1$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_{13} $pc \mapsto 3$ $a_0 \mapsto \mathrm{ite}(X > 0, 1, 0)$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
3: ret
S_{14} $pc \mapsto 0$ $a_0 \mapsto \mathrm{ite}(X > 0, 1, 0)$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$
S_{15} $pc \mapsto 0$ $a_0 \mapsto \mathrm{ite}(X < 0, -1, \mathrm{ite}(X > 0, 1, 0))$ $a_1 \mapsto \mathrm{ite}(X < 0, 1, 0)$

Figure: Symbolic evaluation of the sign program using the ToyRISC interpreter

Symbolic profile and optimization

Why Serval is the choice to write specification?

- Diagnosing performance bottlenecks: Rosette symbolic profiler.
- Symbolic optimizations. For example, to limit the values of some symbol.

Use Serval to verify an operating system

The properties

- absence of undefined behavior
- state-machine refinement
- safety properties

The models

- Execution model: shown in figure.
- Memory model: a set of disjoint blocks of three types.

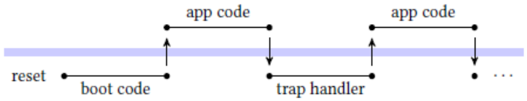


Figure: System execution: System execution: the lower half and the higher half denote execution in privileged and unprivileged modes, respectively; and arrows denote privilege-level transitions.

Lightweight

"Four person-weeks to verify an operating system."

- LoC of Serval framework: 1,244
- LoC of RISC-V verifier: 1,036
- LoC of LLVM verifier: 789
- LoC of CertiKOS: 2,847

Contents

Aim

- Tailor ucore with necessary simplification.
- Verify ucore on RISC-V step-by-step.

Contents

Work that is already done

- Paper Reading: *Scaling symbolic evaluation for automated verification of systems code with Serval*
- Reproduce the verifications in Serval paper.
- Environment setting: Travis-CI

Contents

Plan

1. Learn more details about the framework: Rosette, RISC-V verifier, LLVM verifier, verification of CertiKOS, verification of Komodo...
2. Verify a toy security monitor using on RISC-V and LLVM.
3. Verify and simplify ucore step-by-step.