Semi-automatic Verification of ISA Security Guarantees in the Form of Universal Contracts

(work in progress)

Sander Huyghebaert, Steven Keuchel, Dominique Devriese

Vrije Universiteit Brussel

SOFTWARE LANGUAGES LAB

Outline

Introduction

Universal Contracts

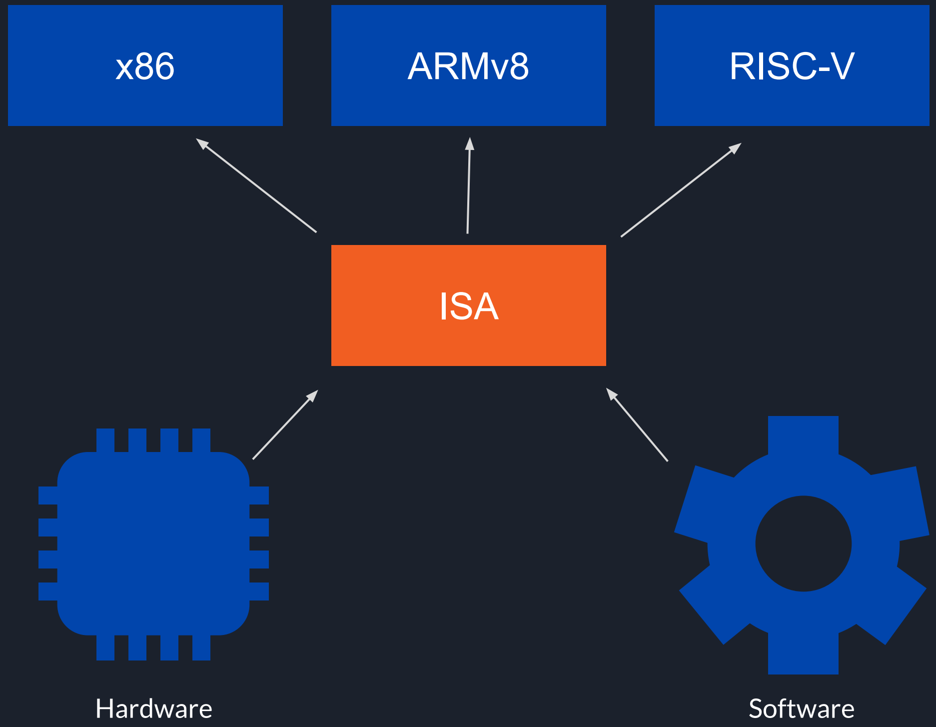
The MinimalCaps Capability Machine

Katamaran

Verifying MinimalCaps' Security Guarantees

Future Work

Introduction



Traditionally:
- Long manuals
- Prose

Recently:
- Formal & executable spec

Security Guarantees

Example: Intel SGX

“The SGX1 extensions allow an application to instantiate a protected container, referred to as an enclave. The enclave is a **trusted area of memory**, where critical aspects of the application functionality have hardware-enhanced **confidentiality** and **integrity protections**. New access controls to **restrict access** to software not resident in the enclave are also introduced. The SGX2 extensions allow additional flexibility in runtime management of enclave resources and thread execution within an enclave.”

- Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3D

Security Guarantees

Example: AMD64

“Only privileged software running at CPL=0 can manage the TLBs.”

> “Page translation is controlled by the PG bit in CR0 (bit 31). When CR0.PG is set to 1, page translation is enabled.”

“Most instructions used to access these resources are privileged and can only be executed while the processor is running at CPL=0, although some instructions can be executed at any privilege level.”

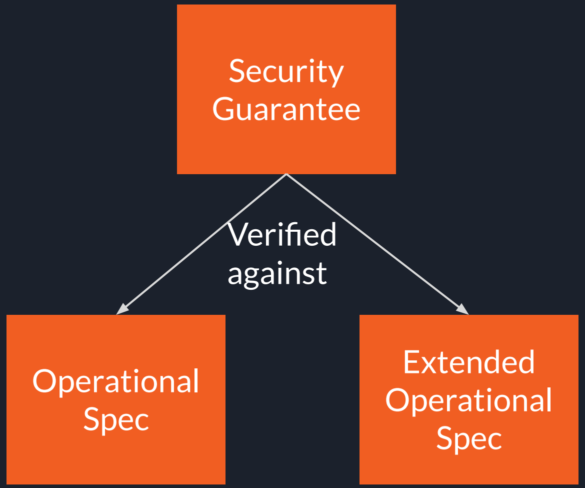
- AMD64 Architecture Programmer’s Manual Volume 2: System Programming

Security Guarantees

Current Approach

- Informal ISA specs offer *promise* of security guarantee
  - “Security guarantee *X* offers *Y* / prevents attack *Z*”
  - Holds for future updates to the ISA
- Formal ISA specs *lack* security specifications
  - Focus is on operational specification

Universal Contracts
Motivation

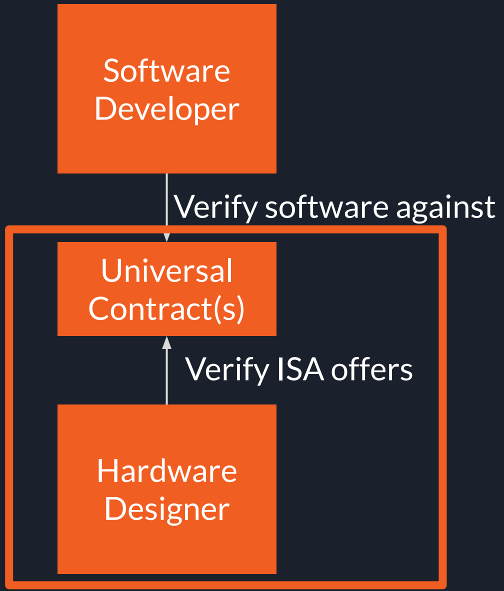


- Security guarantees should be
  - Part of ISA specification
  - Formal
  - Verifiable against operational spec
  - Specific enough for reasoning
  - Not overspecified
    - Optimizations and extensions should be possible
- Current approaches do *not* meet these requirements

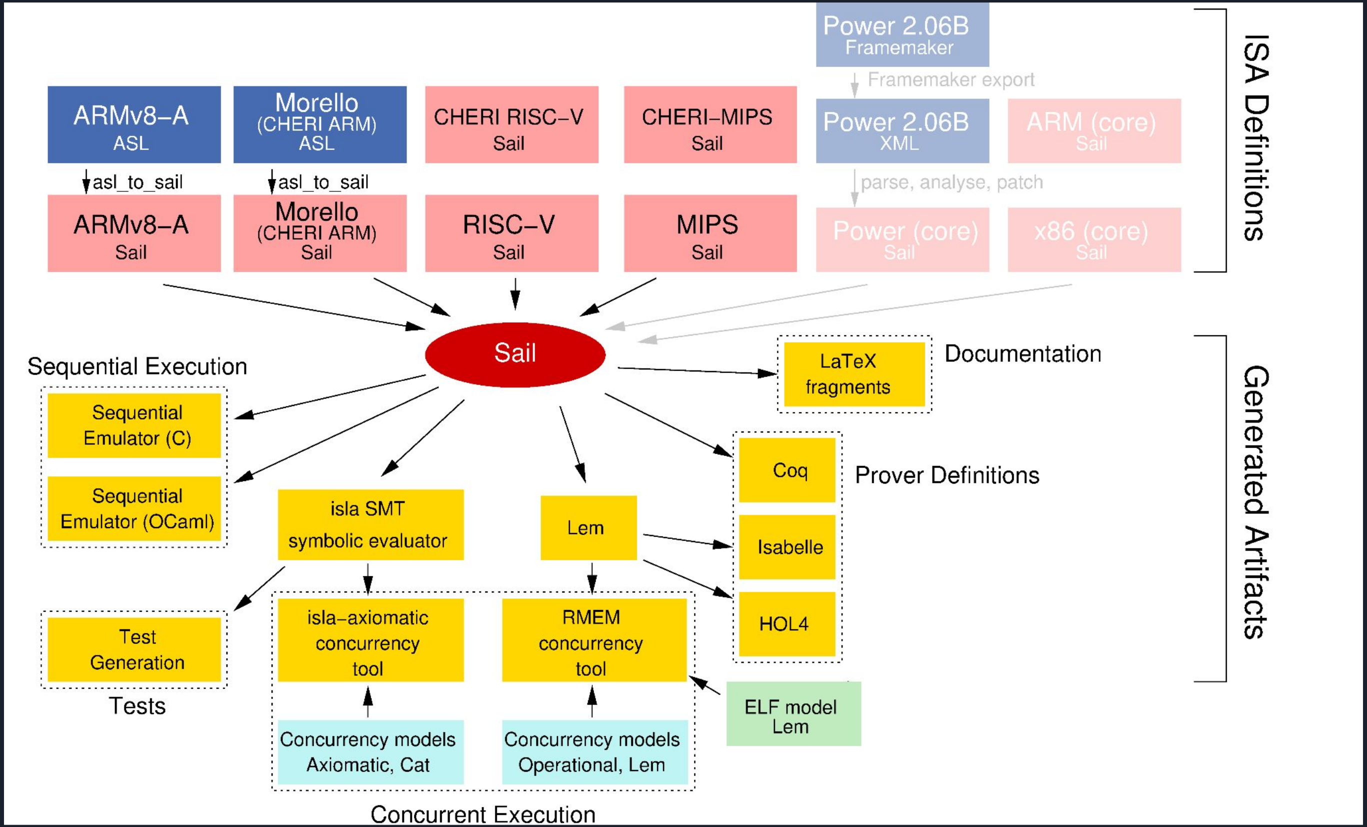
Universal Contracts
Concept

{{ security guarantee }} *ASM code* {{ security guarantee }}

- Formal security guarantee…
- … expressed as a contract
  - Upper bound of the authority
- Holds for *any* code
- Verifiable against operational specification of ISA
  - Sail

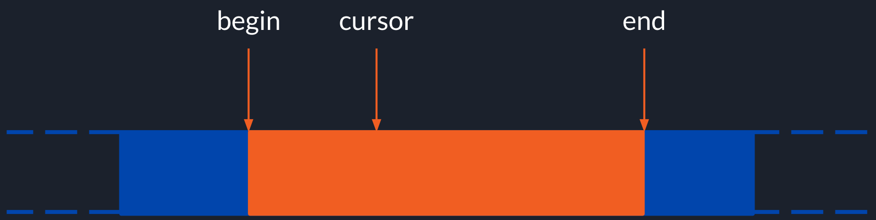


Sail


Power 2.06B
Framemaker
Framemaker export
ARMv8–A
ASL
Morello
(CHERI ARM)
ASL
CHERI RISC–V
Sail
CHERI–MIPS
Sail
Power 2.06B
XML
ARM (core)
Sail
asl_to_sail
asl_to_sail
parse, analyse, patch
ARMv8–A
Sail
Morello
(CHERI ARM)
Sail
RISC–V
Sail
MIPS
Sail
Power (core)
Sail
x86 (core)
Sail
ISA Definitions
Sail
Sequential Execution
Sequential
Emulator (C)
Sequential
Emulator (OCaml)
LaTeX
fragments
Documentation
Coq
Prover Definitions
isla SMT
symbolic evaluator
Lem
Isabelle
HOL4
Generated Artifacts
Test
Generation
Tests
isla–axiomatic
concurrency
tool
RMEM
concurrency
tool
ELF model
Lem
Concurrency models
Axiomatic, Cat
Concurrency models
Operational, Lem
Concurrent Execution

The MinimalCaps Capability Machine

Memory



Capability

- perm ∈{O, R, RW}
- begin : address
- cursor : address
- end : address

Hardware Guarantees

- Capabilities are unforgeable
- Permissions are checked
- Capability manipulation is safe

Capability Safety

Machine Invariant

$$(\exists\ c, pc \mapsto c \ast \mathcal{V}(c)) \ast (\forall\ r \in GPR.\ \exists\ w.\ r \mapsto w \ast \mathcal{V}(w))$$

Logical Relation $\mathcal{V}$

$$\mathcal{V}(w) \begin{cases} \mathcal{V}(z) = \text{True } (z \text{ is an integer}) \\ \mathcal{V}(O, -, -, -) = \text{True} \\ \mathcal{V}(R, b, e, -) = \ast_{a \in [b, e]}\ \exists\ \boxed{w, a \mapsto w \ast \mathcal{V}(w)} \\ \mathcal{V}(RW, b, e, -) = \ast_{a \in [b, e]}\ \exists\ \boxed{w, a \mapsto w \ast \mathcal{V}(w)} \end{cases}$$

Contract

Execute

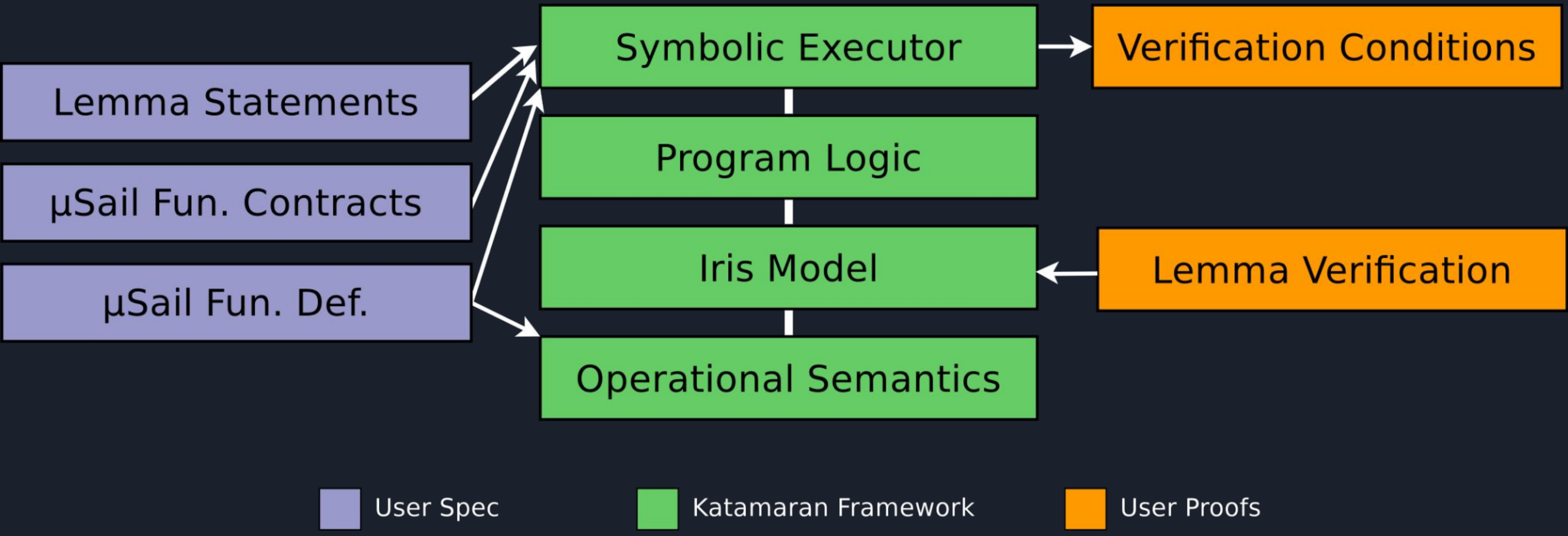
$\{\{ (\exists\ c\ .\ pc \mapsto c \ast \mathcal{V}(c)) \ast (\forall\ r \in GPR\ .\ \exists\ w\ .\ r \mapsto w \ast \mathcal{V}(w)) \}\}$

```
function execute() : bool :=
  let c := call read_reg_cap pc in
  let n := call read_mem c in
  match n with
  | inl n =>
    let i := call decode n in
    call exec_instr i
  | inr c => fail
  end
```

$\{\{ (\exists\ c\ .\ pc \mapsto c \ast \mathcal{V}(c)) \ast (\forall\ r \in GPR\ .\ \exists\ w\ .\ r \mapsto w \ast \mathcal{V}(w)) \}\}$

Katamaran

Semi-automatic separation logic verifier



Contracts

Selection

$\{\{\ \mathcal{V}(c)\ \}\}$ read_mem c $\{\{\ w\ .\ \mathcal{V}(w) \ast \mathcal{V}(c)\ \}\}$

$\{\{\ r \mapsto w\ \}\}$ read_reg r $\{\{\ v\ .\ v = w \ast r \mapsto w\ \}\}$

$\{\{\ r \mapsto w\ \}\}$ read_reg_cap r $\{\{\ c\ .\ c = w \ast r \mapsto w\ \}\}$

$\{\{\ \mathcal{V}(c) \ast \mathcal{V}(w)\ \}\}$ write_mem c w $\{\{\ \mathcal{V}(c)\ \}\}$

$\{\{\ pc \mapsto c \ast \mathcal{V}(c)\ \}\}$ update_pc $\{\{\ \exists c\ .\ pc \mapsto c \ast \mathcal{V}(c)\ \}\}$

$\{\{\ \mathcal{V}(w)\ \}\}$ duplicate_safe w $\{\{\ \mathcal{V}(w) \ast \mathcal{V}(w)\ \}\}$

$\{\{\ \mathcal{V}(c)\ \}\}$ move_cursor c c' $\{\{\ \mathcal{V}(c) \ast \mathcal{V}(c')\ \}\}$

Verifying MinimalCaps' Security Guarantees

```
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
function exec_sd(rs : GPR, rb : GPR, immediate : int) : bool :=
  let base_cap := call read_reg_cap rb in
  let (perm, beg, end, cursor) := base_cap in
  let c := (perm, beg, end, cursor + immediate) in
  let w := call read_reg rs in
  use lemma (duplicate_safe w) ;;
  use lemma (move_cursor base_cap c) ;;
  call write_mem c w ;;
  call update_pc ;;
  true
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
```

Verifying MinimalCaps' Security Guarantees

```
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
function exec_sd(rs : GPR, rb : GPR, immediate : int) : bool :=
  let base_cap := call read_reg_cap rb in
  let (perm, beg, end, cursor) := base_cap in
  let c := (perm, beg, end, cursor + immediate) in
  let w := call read_reg rs in
  {{ rb ↦ base_cap ✱ 𝒱(base_cap) ✱ rs ↦ w ✱ 𝒱(w) ...}}
  use lemma (duplicate_safe w) ;;
  use lemma (move_cursor base_cap c) ;;
  call write_mem c w ;;
  call update_pc ;;
  true
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
```

Verifying MinimalCaps' Security Guarantees

```
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
function exec_sd(rs : GPR, rb : GPR, immediate : int) : bool :=
  let base_cap := call read_reg_cap rb in
  let (perm, beg, end, cursor) := base_cap in
  let c := (perm, beg, end, cursor + immediate) in
  let w := call read_reg rs in
  {{ rb ↦ base_cap ✱ 𝒱(base_cap) ✱ rs ↦ w ✱ 𝒱(w) ...}}
  use lemma (duplicate_safe w) ;;
  use lemma (move_cursor base_cap c) ;;
  {{ rb ↦ base_cap ✱ 𝒱(base_cap) ✱ rs ↦ w ✱ 𝒱(w) ✱ 𝒱(w) ✱ 𝒱(c) ...}}
  call write_mem c w ;;
  call update_pc ;;
  true
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
```

Verifying MinimalCaps' Security Guarantees

```
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
function exec_sd(rs : GPR, rb : GPR, immediate : int) : bool :=
  let base_cap := call read_reg_cap rb in
  let (perm, beg, end, cursor) := base_cap in
  let c := (perm, beg, end, cursor + immediate) in
  let w := call read_reg rs in
  use lemma (duplicate_safe w) ;;
  use lemma (move_cursor base_cap c) ;;
  {{ rb ↦ base_cap ✱ 𝒱(base_cap) ✱ rs ↦ w ✱ 𝒱(w) ✱ 𝒱(w) ✱ 𝒱(c) ...}}
  call write_mem c w ;;
  {{ rb ↦ base_cap ✱ 𝒱(base_cap) ✱ rs ↦ w ✱ 𝒱(w) ✱ 𝒱(c) ...}}
  call update_pc ;;
  true
{{ (∃ c . pc ↦ c ✱ 𝒱(c)) ✱ (∀ r ∈ GPR . ∃ w . r ↦ w ✱ 𝒱(w)) }}
```

Future Work

Accomplished goals | Currently working on

Capability Safety	Soundness Katamaran	Object Capabilities	Case Studies	Proof Automation	Larger ISAs	Complex ISAs	Realistic ISAs
Capability safety of the MinimalCaps machine	Soundness proof of Katamaran	Add support for object capabilities to the MinimalCaps case study	Verification of security properties of ISAs with different security primitives	Further improve proof automation of Katamaran	Scale up the number of instructions in ISAs we consider	Introduce features such as concurrency, interrupts, ...	Verify security properties of real ISAs, i.e. RISC-V, CHERI-RISC-V, ...

Conclusion

- Security Guarantees
  - Formalized with Universal Contracts
  - Part of *security guarantee* specification
  - Verified against *operational* specification
- Case Study: MinimalCaps
  - Capability safety
- Katamaran
  - Semi-automatic separation logic verifier

Thank you!