Implementing a Capability Machine model into Iris

Aïna Linn Georges Alix Trieu Lars Birkedal

Aarhus University

ageorges@cs.au.dk

October 25, 2019

Overview

Layers of Abstraction

Programming Languages

Assembly Language

Machine Code

. . .

Layers of Abstraction

Programming Languages

- Local State Encapsulation

Assembly Language

Machine Code

...

Layers of Abstraction

Programming Languages

- Local State Encapsulation
- Well Bracketed Control Flow

Assembly Language

Machine Code

...

Layers of Abstraction

Programming Languages

- Local State Encapsulation
- Well Bracketed Control Flow

Assembly Language

- Programs lie in Memory, Program Counter, ...

Machine Code

. . .

Layers of Abstraction

Programming Languages

- Local State Encapsulation
- Well Bracketed Control Flow

Assembly Language

- Programs lie in Memory, Program Counter, ...
- Arbitrary Pointer Manipulation

Machine Code

...

Layers of Abstraction

Programming Languages

- Local State Encapsulation
- Well Bracketed Control Flow

Assembly Language

- Programs lie in Memory, Program Counter, ...
- Arbitrary Pointer Manipulation
- Arbitrary Jumps

Machine Code

. . .

Layers of Abstraction

Programming Languages

- Local State Encapsulation
- Well Bracketed Control Flow

Assembly Language

- Programs lie in Memory, Program Counter, ...
- Arbitrary Pointer Manipulation
- Arbitrary Jumps

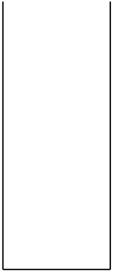
Machine Code

- Instruction Decoding, Cache, etc.

. . .

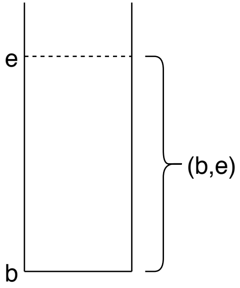
Capability Machine

Capability: An unforgeable token of authority



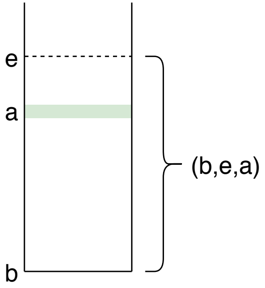
Capability Machine

Capability: An unforgeable token of authority



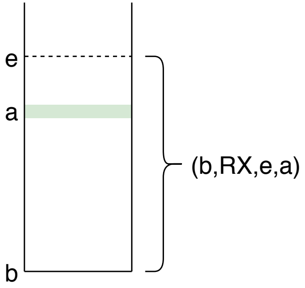
Capability Machine

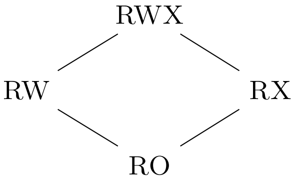
Capability: An unforgeable token of authority



Capability Machine

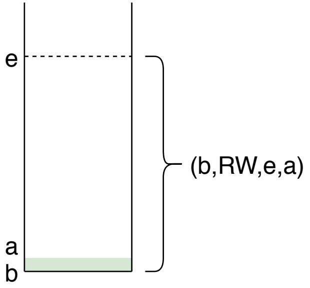
Capability: An unforgeable token of authority





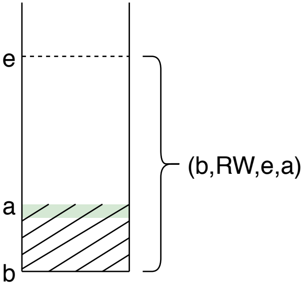
Capability Safety

Local State Encapsulation



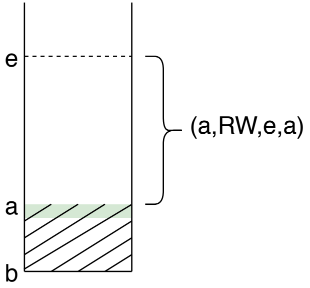

```
push r_stk 1
scall r
pop r_stk r_1
assert r_1 1
halt
```

Local State Encapsulation



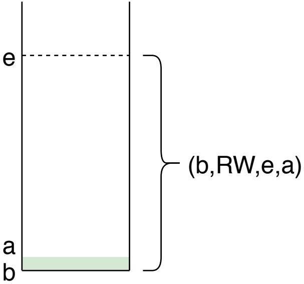

```
push r_stk 1
scall r
pop r_stk r_1
assert r_1 1
halt
```

Local State Encapsulation




```
push r_stk 1
scall r
pop r_stk r_1
assert r_1 1
halt
```

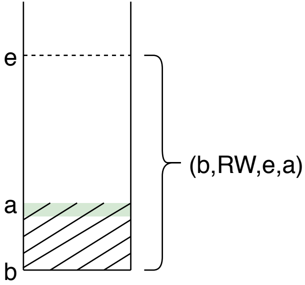
Well Bracketed Control Flow




```
push r_stk 1
scall r ([r])
pop r_stk r_1
assert r_1 1
push r_stk 2
scall r
halt
```

Capability Safety

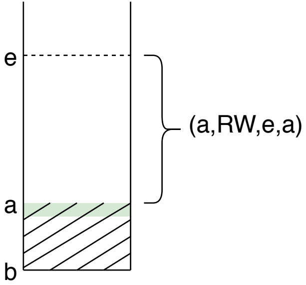
Well Bracketed Control Flow




```
push r_stk 1
scall r ([r])
pop r_stk r_1
assert r_1 1
push r_stk 2
scall r
halt
```

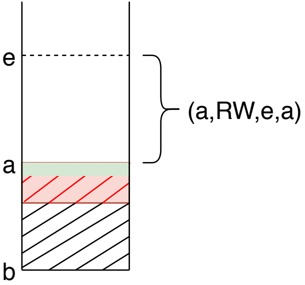
Capability Safety

Well Bracketed Control Flow



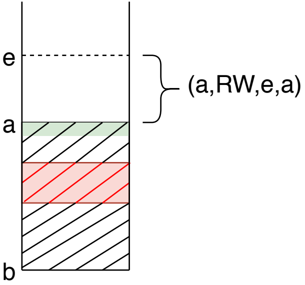

```
push r_stk 1
scall r ([r])
pop r_stk r_1
assert r_1 1
push r_stk 2
scall r
halt
```

Well Bracketed Control Flow



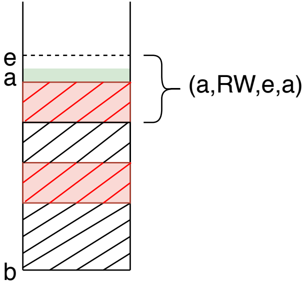

```
push r_stk 1
scall r ([r])
pop r_stk r_1
assert r_1 1
push r_stk 2
scall r
halt
```

Well Bracketed Control Flow




```
push r_stk 1
scall r ([r])
pop r_stk r_1
assert r_1 1
push r_stk 2
scall r
halt
```

Well Bracketed Control Flow



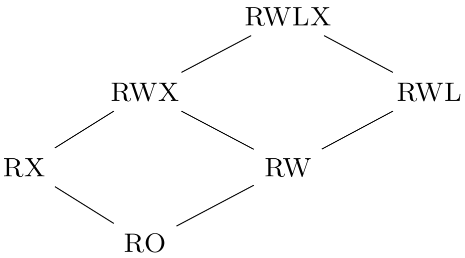

```
push r_stk 1
scall r ([r])
pop r_stk r_1
assert r_1 1
push r_stk 2
scall r
halt
```

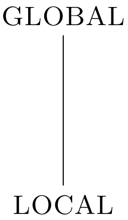
Local Capabilities

(p,**Local**,b,e,a) (p,**Global**,b,e,a)

Local Capabilities

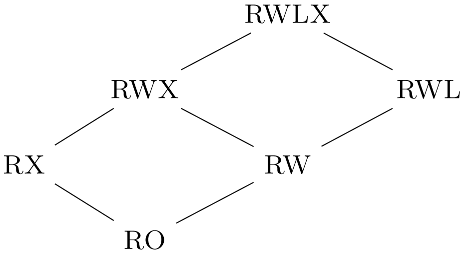
(p,**Local**,b,e,a) (p,**Global**,b,e,a)





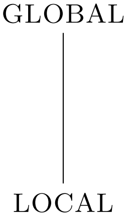
Local Capabilities

(p,**Local**,b,e,a)



well-bracketed

(p,**Global**,b,e,a)



not well-bracketed

Expressing Capability Safety

Expressing Capability Safety

- using a Program Logic

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system
- **using a Logical Relation on an untyped (or uni-typed) language to capture semantic properties of the language**

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system
- **using a Logical Relation on an untyped (or uni-typed) language to capture semantic properties of the language**
    1. embed the language into Iris

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system
- **using a Logical Relation on an untyped (or uni-typed) language to capture semantic properties of the language**
    1. embed the language into Iris
    2. define a program logic by proving Hoare Triples

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system
- **using a Logical Relation on an untyped (or uni-typed) language to capture semantic properties of the language**
  1. embed the language into Iris
  2. define a program logic by proving Hoare Triples
  3. define the logical relation

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system
- **using a Logical Relation on an untyped (or uni-typed) language to capture semantic properties of the language**
  1. embed the language into Iris
  2. define a program logic by proving Hoare Triples
  3. define the logical relation
  4. prove the fundamental theorem of logical relations

Expressing Capability Safety

- using a Program Logic
- using a Logical Relation to capture invariants on the type system
- **using a Logical Relation on an untyped (or uni-typed) language to capture semantic properties of the language**
  1. embed the language into Iris
  2. define a program logic by proving Hoare Triples
  3. define the logical relation
  4. prove the fundamental theorem of logical relations
  5. use the logical relation to prove examples that rely on LSE and WBCF with calls to unknown adversary

Abstract Instructions

$$(reg, mem) \to (reg', mem')$$

Abstract Instructions

$$(reg, mem) \to (reg', mem')$$

- Instr Executable

Abstract Instructions

$$(reg, mem) \rightarrow (reg', mem')$$

- Instr Executable
- Instr Halted

Abstract Instructions

$$(reg, mem) \rightarrow (reg', mem')$$

- Instr Executable
- Instr Halted
- Instr Failed

Abstract Instructions

$$(reg, mem) \to (reg', mem')$$

- Instr Executable
- Instr Halted $\to$ HaltedV
- Instr Failed $\to$ FailedV

Hoare Triples of the Program Logic: Success

$$
\begin{aligned}
& \quad decode(w) = \text{Load dst src} \\
& \quad \wedge \text{ isCorrectPC } ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
& \quad \wedge \text{ readAllowed } p_{src} \ \wedge \ \text{withinBounds } (b_{src}, e_{src}, a_{src}) \\
& \quad \wedge \ p_{pc} \sqsubseteq p'_{pc} \ \wedge \ p_{src} \sqsubseteq p'_{src} \\
& \{\{\{ \text{ PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
& \quad * \, dst \mapsto_r w_{dst} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
& \quad * \, a_{src} \mapsto_a [p'_{src}] w_{src} \}\}\} \\
& \quad \text{Instr Executable} \\
& \{\{\{ \ \}\}\}
\end{aligned}
$$

Hoare Triples of the Program Logic: Success

$$
\begin{array}{l}
\quad decode(w) = \text{Load dst src} \\
\quad \wedge\ \text{isCorrectPC}\ ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
\quad \wedge\ \text{readAllowed}\ p_{src}\ \wedge\ \text{withinBounds}\ (b_{src}, e_{src}, a_{src}) \\
\quad \wedge\ p_{pc} \sqsubseteq p'_{pc}\ \wedge\ p_{src} \sqsubseteq p'_{src} \\
\{\{\{\ \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
\quad * dst \mapsto_r w_{dst} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
\quad * a_{src} \mapsto_a [p'_{src}]w_{src}\}\}\} \\
\quad \text{Instr Executable} \\
\{\{\{\ \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc} + 1)\}\}\}
\end{array}
$$

Hoare Triples of the Program Logic: Success

$$
\begin{aligned}
& \quad decode(w) = \text{Load dst src} \\
& \wedge \text{ isCorrectPC } ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
& \wedge \text{ readAllowed } p_{src} \ \wedge \ \text{withinBounds } (b_{src}, e_{src}, a_{src}) \\
& \wedge \ p_{pc} \sqsubseteq p'_{pc} \ \wedge \ p_{src} \sqsubseteq p'_{src} \\
\{\{\{ & \ \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
& \quad * \ dst \mapsto_r w_{dst} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
& \quad * \ a_{src} \mapsto_a [p'_{src}]w_{src} \}\}\} \\
& \quad \text{Instr Executable} \\
\{\{\{ & \ \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc} + 1) * a_{pc} \mapsto_a [p'_{pc}]w \}\}\}
\end{aligned}
$$

Hoare Triples of the Program Logic: Success

$$
\begin{aligned}
& \quad decode(w) = \text{Load dst src} \\
& \land \ \text{isCorrectPC} \ ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
& \land \ \text{readAllowed} \ p_{src} \ \land \ \text{withinBounds} \ (b_{src}, e_{src}, a_{src}) \\
& \land \ p_{pc} \sqsubseteq p'_{pc} \ \land \ p_{src} \sqsubseteq p'_{src}
\end{aligned}
$$

$$
\begin{aligned}
\{\{\{ \ & \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
& * \ dst \mapsto_r w_{dst} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
& * \ a_{src} \mapsto_a [p'_{src}]w_{src} \}\}\} \\
& \text{Instr Executable} \\
\{\{\{ \ & \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc} + 1) * a_{pc} \mapsto_a [p'_{pc}]w \\
& * \ dst \mapsto_r w_{src} \}\}\}
\end{aligned}
$$

Hoare Triples of the Program Logic: Success

$$
\begin{aligned}
& \quad decode(w) = \text{Load dst src} \\
& \wedge \text{ isCorrectPC } ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
& \wedge \text{ readAllowed } p_{src} \;\wedge\; \text{withinBounds } (b_{src}, e_{src}, a_{src}) \\
& \wedge\; p_{pc} \sqsubseteq p'_{pc} \;\wedge\; p_{src} \sqsubseteq p'_{src} \\
\{\{\{ & \text{ PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
& * dst \mapsto_r w_{dst} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
& * a_{src} \mapsto_a [p'_{src}]w_{src} \}\}\} \\
& \quad \text{Instr Executable} \\
\{\{\{ & \text{ PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc} + 1) * a_{pc} \mapsto_a [p'_{pc}]w \\
& * dst \mapsto_r w_{src} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \}\}\}
\end{aligned}
$$

Hoare Triples of the Program Logic: Success

$$
\begin{aligned}
&\quad decode(w) = \text{Load dst src} \\
&\wedge \text{ isCorrectPC } ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
&\wedge \text{ readAllowed } p_{src} \ \wedge \ \text{withinBounds } (b_{src}, e_{src}, a_{src}) \\
&\wedge \ p_{pc} \sqsubseteq p'_{pc} \ \wedge \ p_{src} \sqsubseteq p'_{src} \\
&\{\{\{ \text{ PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
&\quad * dst \mapsto_r w_{dst} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
&\quad * a_{src} \mapsto_a [p'_{src}]w_{src}\}\}\} \\
&\quad \text{Instr Executable} \\
&\{\{\{ \text{ PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}+1) * a_{pc} \mapsto_a [p'_{pc}]w \\
&\quad * dst \mapsto_r w_{src} * src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src}) \\
&\quad * a_{src} \mapsto_a [p'_{src}]w_{src}\}\}\}
\end{aligned}
$$

Points to Predicate with Permissions

$$a \mapsto_a [RWL]w$$

Points to Predicate with Permissions

$$a \mapsto_a [RWL]w \rightarrow a \mapsto_a [RWL]((p, Local), b, e, l)$$

Points to Predicate with Permissions

$$
\begin{aligned}
a \mapsto_a [RWL]w &\rightarrow a \mapsto_a [RWL]((p, Local), b, e, l) \\
&\rightarrow a \mapsto_a [RW]((p, Local), b, e, l)
\end{aligned}
$$

Points to Predicate with Permissions

$$
\begin{aligned}
a \mapsto_a [RWL]w &\rightarrow a \mapsto_a [RWL]((p, Local), b, e, l) \\
&\rightarrow a \mapsto_a [RW]((p, Local), b, e, l) \\
&\nrightarrow a \mapsto_a [RW]((p', Local), b', e', l')'
\end{aligned}
$$

Hoare Triples of the Program Logic: Failure

$$
\begin{array}{l}
\quad decode(w) = \text{Load dst src} \\
\quad \wedge \ \text{isCorrectPC} \ ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) \\
\quad \wedge \ \neg\text{readAllowed} \ p_{src} \ \vee \ \neg\text{withinBounds} \ (b_{src}, e_{src}, a_{src}) \\
\quad \wedge \ p_{pc} \sqsubseteq p'_{pc} \\
\{\{\{ \ \text{PC} \mapsto_r ((p_{pc}, g_{pc}), b_{pc}, e_{pc}, a_{pc}) * a_{pc} \mapsto_a [p'_{pc}]w \\
\quad * \ src \mapsto_r ((p_{src}, g_{src}), b_{src}, e_{src}, a_{src})\}\}\} \\
\quad \text{Instr Executable} \\
\{\{\{ \ \text{FailedV}, \top\}\}\}
\end{array}
$$

The Value Relation

A unary logical relation of an un-typed language

$$\mathcal{V} : \mathit{Word} \rightarrow \mathit{iProp}\ \Sigma$$

The Value Relation

A unary logical relation of an un-typed language

$$\mathcal{V} : \mathit{World} \to \mathit{Word} \to \mathit{iProp}\ \Sigma$$

- Reasoning about local state: World
  A collection of state transition systems

The Value Relation

A unary logical relation of an un-typed language

$$\mathcal{V} : \textcolor{red}{World} \to \ Word \to iProp\ \Sigma$$

- Reasoning about local state: World
  A collection of state transition systems

$$\mathcal{V}(W)(z) \triangleq \exists z' \in \mathbb{Z}. z = z'$$

$$\mathcal{V}(W)(((\text{RO}, g), b, e, a)) \triangleq \exists\ p', \text{RO} \sqsubseteq p' * \text{read\_write\_cond(p',b,e)}$$

$$\mathcal{V}(W)(((\text{RX}, g), b, e, a)) \triangleq \exists\ p', \text{RX} \sqsubseteq p' * \text{read\_write\_cond(p',b,e)}$$

$$* \ \square \ \text{exec\_cond(W)(p,g,b,e)}$$

The Execute Condition

$$\text{exec\_cond(W)(p,g,b,e)} \triangleq \begin{cases} \forall a \in [b\ e], W' \sqsupseteq_{pub} W. & \\ \quad \triangleright\ \mathcal{E}(W')((((p,g),b,e,a)) & g = Local \\ & \\ \forall a \in [b\ e], W' \sqsupseteq_{priv} W. & \\ \quad \triangleright\ \mathcal{E}(W')((((p,g),b,e,a)) & g = Global \end{cases}$$

The Expression Relation

$$
\begin{aligned}
\mathcal{E}(W)(pc) \triangleq \forall r, \mathcal{R}(W)(r) \; * \; & \text{context}(W)(r[\text{PC} := pc]) \\
\mathrel{-\!\!*} \; & \mathrm{WP}\ \text{Seq (Instr Executable)} \\
& \quad \{v, v = \mathit{HaltedV} \implies \exists W' r', W' \sqsubseteq_{priv} W \\
& \quad * \, \text{context}(W')(r')\}
\end{aligned}
$$

The Expression Relation

$$\mathcal{E}(W)(pc) \triangleq \forall r, \mathcal{R}(W)(r) \ * \ \text{context}(W)(r[\text{PC} := pc])$$
$$-\!\!* \ \ \mathrm{WP}\ \text{Seq (Instr Executable)}$$
$$\{v, v = \mathit{HaltedV} \implies \exists W' r', W' \sqsubseteq_{priv} W$$
$$* \ \text{context}(W')(r')\}$$

$$\text{context}(W)(r) = \ ?$$

The Expression Relation

$$
\begin{aligned}
\mathcal{E}(W)(pc) \triangleq \forall r, \mathcal{R}(W)(r) \ &* \ \mathsf{context}(W)(r[\mathsf{PC} := pc]) \\
&\mathrel{-\!\!*} \ \mathrm{WP}\ \mathsf{Seq}\ (\mathsf{Instr}\ \mathsf{Executable}) \\
&\quad \{v, v = HaltedV \implies \exists W' r', W' \sqsupseteq_{priv} W \\
&\quad * \ \mathsf{context}(W')(r')\}
\end{aligned}
$$

$$
\mathsf{context}(W)(r) = \left(\mathop{\Large *}_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \land \mathsf{full\_map}\ r
$$

The Expression Relation

$$\mathcal{E}(W)(pc) \triangleq \forall r, \mathcal{R}(W)(r) \ * \ \text{context}(W)(r[\text{PC} := pc])$$
$$\mathrel{-\!\!*} \ \mathrm{WP}\ \text{Seq (Instr Executable)}$$
$$\{v, v = HaltedV \implies \exists W' r', W' \sqsubseteq_{priv} W$$
$$* \ \text{context}(W')(r')\}$$

$$\text{context}(W)(r) = \Big(\mathop{\Huge *}_{r_i \mapsto w \in r} r_i \mapsto_r w \Big) \land \text{full\_map } r$$
$$* \ \text{na\_inv } \gamma_{na} \top$$

The Expression Relation

$$\mathcal{E}(W)(pc) \triangleq \forall r, \mathcal{R}(W)(r) \ * \ \text{context}(W)(r[\mathsf{PC} := pc])$$
$$\twoheadrightarrow \ \mathrm{WP}\ \mathsf{Seq}\ (\mathsf{Instr\ Executable})$$
$$\{v, v = HaltedV \implies \exists W' r', W' \sqsubseteq_{priv} W$$
$$* \ \text{context}(W')(r')\}$$

$$\text{context}(W)(r) = \left(\mathop{\Huge *}_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \land \mathsf{full\_map}\ r$$
$$* \ \mathsf{na\_inv}\ \gamma_{na} \top$$
$$* \ \mathsf{sts\_full}\ W$$

The Expression Relation

$$\mathcal{E}(W)(pc) \triangleq \forall r, \mathcal{R}(W)(r) \ * \ \text{context}(W)(r[\text{PC} := pc]) \\ \quad -\!\!* \ \mathrm{WP}\ \text{Seq (Instr Executable)} \\ \quad \{v, v = HaltedV \implies \exists W' r', W' \sqsupseteq_{priv} W \\ \quad * \ \text{context}(W')(r')\}$$

$$\text{context}(W)(r) = \left(\mathop{\huge *}_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \wedge \text{full\_map}\ r \\ * \ \text{na\_inv}\ \gamma_{na} \top \\ * \ \text{sts\_full}\ W \\ * \ \text{region}\ W$$

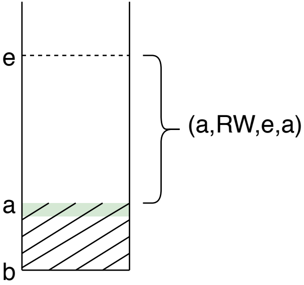
The Fundamental Theorem of Logical Relations

If we can read a region, and that region is safe, we can safely execute it

$$(p = \mathrm{RX} \vee p = \mathrm{RWX} \vee p = \mathrm{RWLX}) \implies$$
$$\text{read\_write\_cond } (p, b, e) \implies \mathcal{E}(W)(((p, g), b, e, a))$$

Capability Safety

We use the FTLR to reason about calls to an unknown adversary




```
push r_stk 1
scall r
pop r_stk r_1
assert r_1 1
halt
```

Conclusion

- Embedding of a Capability Machine into Iris
- The Program Logic
- A unary Logical Relation for an Untyped Capability Machine Language
- Fundamental Theorem of Logical Relations
- Reason about examples that rely on Local Stack Encapsulation and Well-Bracketed Control Flow with calls to an unknown adversary

References

John Smith (2012)
Title of the publication
Journal Name 12(3), 45 – 678.