

Implementing a Capability Machine model into Iris

Aïna Linn Georges

Alix Trieu

Lars Birkedal

Aarhus University

ageorges@cs.au.dk

January 11, 2020

Introduction

- ▶ Capability machines allow for fine grained control over pointer permissions
- ▶ Good target for secure compilation
- ▶ In particular: we are interested in enforcing certain higher level abstractions such as *local state encapsulation* as *well-bracketed control flow* at the lowest level of the machine
- ▶ We need tools to reason about these subtle properties in a language that does not enforce them
- ▶ These tools are elaborate and complex: we want to mechanize them, and facilitate the process of using them

Introduction

- ▶ Capability machines allow for fine grained control over pointer permissions
- ▶ **Good target for secure compilation**
- ▶ In particular: we are interested in enforcing certain higher level abstractions such as *local state encapsulation* as *well-bracketed control flow* at the lowest level of the machine
- ▶ We need tools to reason about these subtle properties in a language that does not enforce them
- ▶ These tools are elaborate and complex: we want to mechanize them, and facilitate the process of using them

Introduction

- ▶ Capability machines allow for fine grained control over pointer permissions
- ▶ Good target for secure compilation
- ▶ In particular: we are interested in enforcing certain higher level abstractions such as *local state encapsulation* as *well-bracketed control flow* at the lowest level of the machine
- ▶ We need tools to reason about these subtle properties in a language that does not enforce them
- ▶ These tools are elaborate and complex: we want to mechanize them, and facilitate the process of using them

Introduction

- ▶ Capability machines allow for fine grained control over pointer permissions
- ▶ Good target for secure compilation
- ▶ In particular: we are interested in enforcing certain higher level abstractions such as *local state encapsulation* as *well-bracketed control flow* at the lowest level of the machine
- ▶ We need tools to reason about these subtle properties in a language that does not enforce them
- ▶ These tools are elaborate and complex: we want to mechanize them, and facilitate the process of using them

Introduction

- ▶ Capability machines allow for fine grained control over pointer permissions
- ▶ Good target for secure compilation
- ▶ In particular: we are interested in enforcing certain higher level abstractions such as *local state encapsulation* as *well-bracketed control flow* at the lowest level of the machine
- ▶ We need tools to reason about these subtle properties in a language that does not enforce them
- ▶ These tools are elaborate and complex: we want to mechanize them, and facilitate the process of using them

Capability Machines

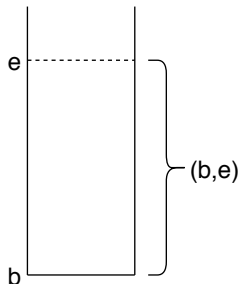
Capability Machine

Capability: An unforgeable token of authority



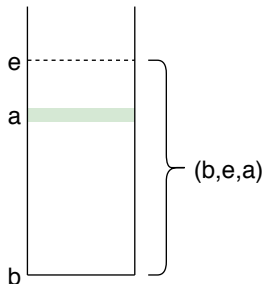
Capability Machine

Capability: An unforgeable token of authority



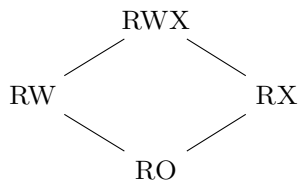
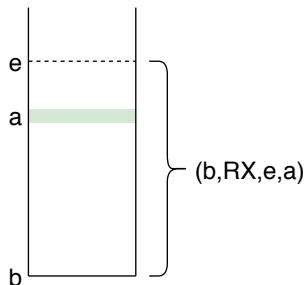
Capability Machine

Capability: An unforgeable token of authority



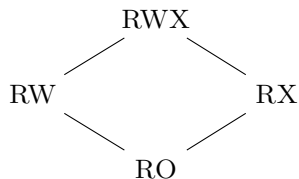
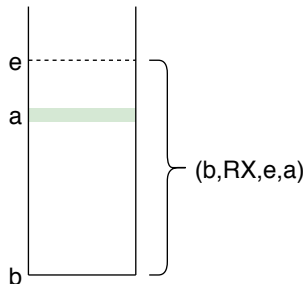
Capability Machine

Capability: An unforgeable token of authority



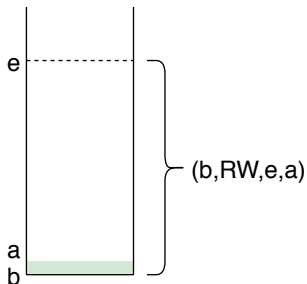
Capability Machine

Capability: An unforgeable token of authority



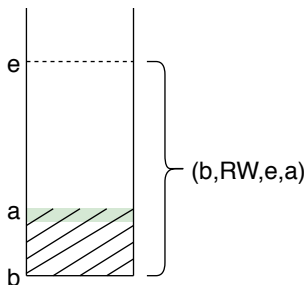
Enforcing Well Bracketed Control Flow using Capabilities

Well Bracketed Control Flow



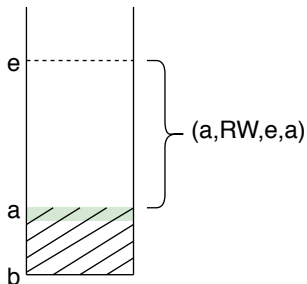
```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 push r_stk 2
6 scall r
7 halt
```

Well Bracketed Control Flow



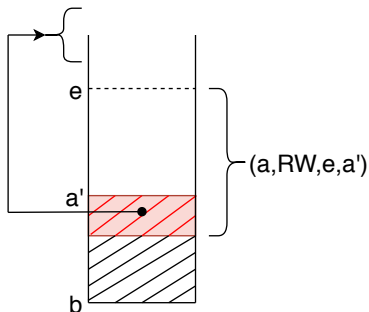
```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 push r_stk 2
6 scall r
7 halt
```

Well Bracketed Control Flow



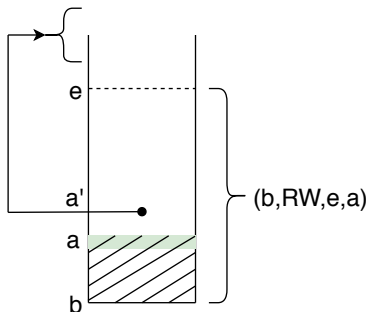
```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 push r_stk 2
6 scall r
7 halt
```


Well Bracketed Control Flow



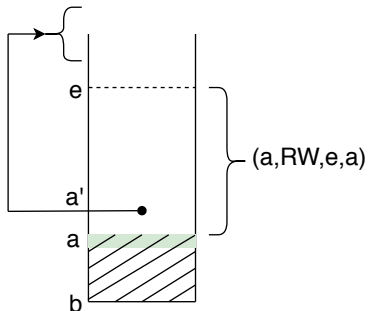
```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 push r_stk 2
6 scall r
7 halt
```

Well Bracketed Control Flow



```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 push r_stk 2
6 scall r
7 halt
```

Well Bracketed Control Flow

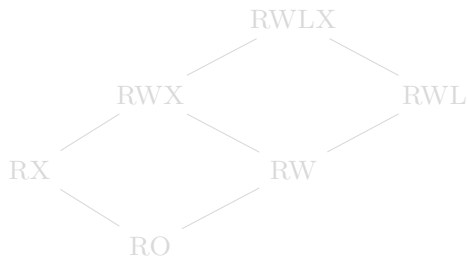


```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 push r_stk 2
6 scall r
7 halt
```

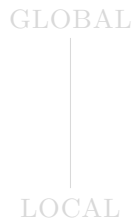
Local Capabilities

Local Capabilities

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

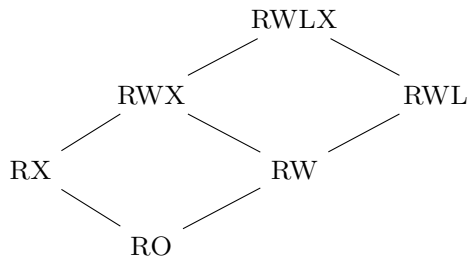


(p, **Global**, b, e, a)

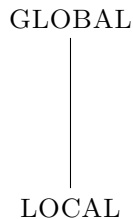


Local Capabilities

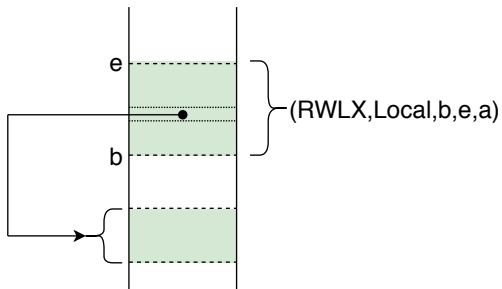
(p, **Local**, b, e, a)



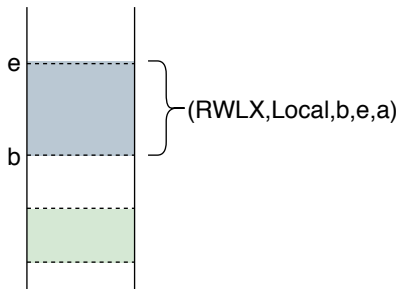
(p, **Global**, b, e, a)



Calling Convention



Calling Convention



Reasoning about Capability Safety

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

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

- ▶ 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**

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{n, (RW, g, b, e, a) | \dots\} \cup \dots$$

- ▶ World-circularity problem
 - ▶ Step indexing
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction:*

\sqsubseteq_{pub} *and* \sqsubseteq_{priv}

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{\textcolor{red}{n}, (RW, g, b, e, a) | \dots\} \cup \dots$$

- ▶ World-circularity problem
 - ▶ *Step indexing*
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction:*

\sqsubseteq_{pub} *and* \sqsubseteq_{priv}

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{ \textcolor{red}{n}, (RW, g, b, e, a) | \exists r, W(r) = \iota_{[b,e]} \} \cup \dots$$

- ▶ World-circularity problem
 - ▶ *Step indexing*
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction:*

\sqsubseteq_{pub} *and* \sqsubseteq_{priv}

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{ \textcolor{red}{n}, (RW, g, b, e, a) | \exists r, W(r) = \iota_{[b,e]} \} \cup \dots$$

- ▶ World-circularity problem
 - ▶ **Step indexing**
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction*:

\sqsubseteq_{pub} *and* \sqsubseteq_{priv}

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{ \textcolor{red}{n}, (RW, g, b, e, a) | \exists r, W(r) \stackrel{\textcolor{red}{n}}{=} \iota_{[b,e]} \} \cup \dots$$

- ▶ World-circularity problem
 - ▶ **Step indexing**
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction*:

\sqsubseteq_{pub} *and* \sqsubseteq_{priv}

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{ \textcolor{red}{n}, (RW, g, b, e, a) | \exists r, W(r) \stackrel{\textcolor{red}{n}}{=} \iota_{[b,e]} \} \cup \dots$$

- ▶ World-circularity problem
 - ▶ **Step indexing**
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction*:

\sqsubseteq_{pub} *and* \sqsubseteq_{priv}

Step-indexed Kripke Logical Relation

$$\mathcal{V}(W) \triangleq \{ \textcolor{red}{n}, (RW, g, b, e, a) | \exists r, W(r) \stackrel{\textcolor{red}{n}}{=} \iota_{[b,e]} \} \cup \dots$$

- ▶ World-circularity problem
 - ▶ **Step indexing**
- ▶ The world may evolve: we need future world relation
 - ▶ Local capabilities are revoked whereas Global capabilities are not, *the relation needs to model this distinction*:

$$\sqsubseteq_{pub} \qquad \text{and} \qquad \sqsubseteq_{priv}$$

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - an Iris primer

Iris: Higher-order Concurrent Separation Logic Framework

- ▶ Foundational
- ▶ Implemented in Coq – equipped with an interactive proof mode
- ▶ Framework – embed any language and its operational semantics into Iris
- ▶ Comes equipped with:
 - ▶ Invariants
 - ▶ Ghost state
 - ▶ Always and Later Modalities

We can take advantage of Iris' step-indexed model and invariants to mechanize step-indexed Kripke logical relations with recursive worlds in a succinct and elegant way.

Expressing Capability Safety in Iris - Challenges

- ▶ Region invariants: Iris invariants
- ▶ Future world relation: frame preserving updates and world satisfaction
- ▶ Step indexing: later modality

Challenges

- ▶ Iris was designed with more high level languages in mind, how do we embed a low level machine language into Iris
- ▶ Iris abstracts away certain details we want to reason about directly
- ▶ There is only one frame preserving update, we need to distinguish between two future world relations

Expressing Capability Safety in Iris - Challenges

- ▶ Region invariants: Iris invariants
- ▶ Future world relation: frame preserving updates and world satisfaction
- ▶ Step indexing: later modality

Challenges

- ▶ Iris was designed with more high level languages in mind, how do we embed a low level machine language into Iris
- ▶ Iris abstracts away certain details we want to reason about directly
- ▶ There is only one frame preserving update, we need to distinguish between two future world relations

Expressing Capability Safety in Iris - Challenges

- ▶ Region invariants: Iris invariants
- ▶ Future world relation: frame preserving updates and world satisfaction
- ▶ Step indexing: later modality

Challenges

- ▶ Iris was designed with more high level languages in mind, how do we embed a low level machine language into Iris
- ▶ Iris abstracts away certain details we want to reason about directly
- ▶ There is only one frame preserving update, we need to distinguish between two future world relations

Expressing Capability Safety in Iris - Challenges

- ▶ Region invariants: Iris invariants
- ▶ Future world relation: frame preserving updates and world satisfaction
- ▶ Step indexing: later modality

Challenges

- ▶ Iris was designed with more high level languages in mind, how do we embed a low level machine language into Iris
- ▶ Iris abstracts away certain details we want to reason about directly
- ▶ There is only one frame preserving update, we need to distinguish between two future world relations

Roadmap

- ▶ embed the language into Iris
- ▶ define a program logic by proving Hoare Triples
- ▶ define the logical relation – using Iris tools to solve the world circularity problem
- ▶ prove the fundamental theorem of logical relations
- ▶ use the logical relation to prove examples that rely on local state encapsulation and well-bracketed control flow with calls to unknown adversary

Roadmap

- ▶ embed the language into Iris
- ▶ define a program logic by proving Hoare Triples
- ▶ define the logical relation – using Iris tools to solve the world circularity problem
- ▶ prove the fundamental theorem of logical relations
- ▶ use the logical relation to prove examples that rely on local state encapsulation and well-bracketed control flow with calls to unknown adversary

Roadmap

- ▶ embed the language into Iris
- ▶ **define a program logic by proving Hoare Triples**
- ▶ define the logical relation – using Iris tools to solve the world circularity problem
- ▶ prove the fundamental theorem of logical relations
- ▶ use the logical relation to prove examples that rely on local state encapsulation and well-bracketed control flow with calls to unknown adversary

Roadmap

- ▶ embed the language into Iris
- ▶ define a program logic by proving Hoare Triples
- ▶ **define the logical relation – using Iris tools to solve the world circularity problem**
- ▶ prove the fundamental theorem of logical relations
- ▶ use the logical relation to prove examples that rely on local state encapsulation and well-bracketed control flow with calls to unknown adversary

Roadmap

- ▶ embed the language into Iris
- ▶ define a program logic by proving Hoare Triples
- ▶ define the logical relation – using Iris tools to solve the world circularity problem
- ▶ **prove the fundamental theorem of logical relations**
- ▶ use the logical relation to prove examples that rely on local state encapsulation and well-bracketed control flow with calls to unknown adversary

Roadmap

- ▶ embed the language into Iris
- ▶ define a program logic by proving Hoare Triples
- ▶ define the logical relation – using Iris tools to solve the world circularity problem
- ▶ prove the fundamental theorem of logical relations
- ▶ use the logical relation to prove examples that rely on local state encapsulation and well-bracketed control flow with calls to unknown adversary

Program Logic

Abstract Instructions

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

- ▶ Instr Executable
- ▶ Instr Halted \rightarrow HaltedV
- ▶ Instr Failed \rightarrow FailedV

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

- ▶ Instr Executable
- ▶ Instr Halted \rightarrow HaltedV
- ▶ Instr Failed \rightarrow FailedV

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

- ▶ Instr Executable
- ▶ Instr Halted \rightarrow HaltedV
- ▶ Instr Failed \rightarrow FailedV

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

- ▶ Instr Executable
- ▶ Instr Halted \rightarrow HaltedV
- ▶ Instr Failed \rightarrow FailedV

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

- ▶ Instr Executable
- ▶ Instr Halted \rightarrow HaltedV
- ▶ Instr Failed \rightarrow FailedV

A Capability Points-to Predicate

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

$$\begin{aligned} a \mapsto_a [RWL]w &\Rightarrow a \mapsto_a [RWL]((p, Local), b, e, l) \\ &\Rightarrow a \mapsto_a [RW]((p, Local), b, e, l) \\ &\not\Rightarrow a \mapsto_a [RW]((p', Local), b', e', l') \end{aligned}$$

A Unary Logical Relation for Reasoning about Semantic Properties of an Untyped Language

The Value Relation

A unary logical relation of an un-typed language

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

Challenge: distinguish between Local and Global capabilities:

- ▶ At the level of the value relation
- ▶ Model revocation

STS: A collection of state transition systems

$$\mathcal{V}((_{RW}, g), b, e, a) \triangleq \bigstar_{a \in [b, e]} \boxed{\exists w, a \mapsto_a [RW]_w * \mathcal{V}(w)}$$

A unary logical relation of an un-typed language

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

Challenge: distinguish between Local and Global capabilities:

- ▶ At the level of the value relation
- ▶ Model revocation

STS: A collection of state transition systems

$$\mathcal{V}((_{RW}, g), b, e, a) \triangleq \bigstar_{a \in [b, e]} \boxed{\exists w, a \mapsto_a [RW]_w * \mathcal{V}(w)}$$

A unary logical relation of an un-typed language

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

Challenge: distinguish between Local and Global capabilities:

- ▶ At the level of the value relation
- ▶ Model revocation

STS: A collection of state transition systems

$$\mathcal{V}((_{RW}, g), b, e, a) \triangleq \bigstar_{a \in [b, e]} \boxed{\exists w, a \mapsto_a [RW]_w * \mathcal{V}(w)}$$

A unary logical relation of an un-typed language

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

Challenge: distinguish between Local and Global capabilities:

- ▶ At the level of the value relation
- ▶ Model revocation

STS: A collection of state transition systems

$$\mathcal{V}((_{RW}, g), b, e, a) \triangleq \bigstar_{a \in [b, e]} \boxed{\exists w, a \mapsto_a [RW]_w * \mathcal{V}(w)}$$

The Value Relation

A unary logical relation of an un-typed language

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

Challenge: distinguish between Local and Global capabilities:

- ▶ At the level of the value relation
- ▶ Model revocation

STS: A collection of state transition systems

$$\mathcal{V}((_{RW}, g), b, e, a) \triangleq \bigstar_{a \in [b, e]} \boxed{\exists w, a \mapsto_a [RW]_w * \mathcal{V}(\Sigma)(w)}$$

The Value Relation

A unary logical relation of an un-typed language

$$\mathcal{V} : \textcolor{red}{STS} \rightarrow \textit{Word} \rightarrow \textit{iProp } \Sigma$$

Challenge: distinguish between Local and Global capabilities:

- ▶ At the level of the value relation
- ▶ Model revocation

STS: A collection of state transition systems

$$\mathcal{V}(\textcolor{red}{\Sigma})((\textcolor{blue}{r}_w, g), b, e, a) \triangleq \bigstar_{a \in [b, e]} \boxed{\exists w, a \mapsto_a [RW]_w * \mathcal{V}(\textcolor{red}{\Sigma})(w)}$$

From World to state transition system collection

On paper:

$$\begin{aligned} \text{Region} = & \{ \text{Revoked} \} \uplus \\ & \{ \text{Temporary} \} \times \text{State} \times \text{Rels} \\ & \times (\text{State} \rightarrow (\text{Wor} \xrightarrow[\exists_{pub}]{mon, ne} \text{UPred}(\text{MemSeg}))) \uplus \\ & \{ \text{Permanent} \} \times \text{State} \times \text{Rels} \\ & \times (\text{State} \rightarrow (\text{Wor} \xrightarrow[\exists_{priv}]{mon, ne} \text{UPred}(\text{MemSeg}))) \end{aligned}$$

$$\text{World} = \mathbb{N} \rightarrow \text{Region}$$

In the Iris mechanization, we use a collection of state transition systems:

$$\Sigma : \mathbb{N} \rightarrow \text{States} \times \mathbb{N} \rightarrow \text{Rels}$$

The world circularity problem is now handled using Iris invariants and saved predicates.

From World to state transition system collection

On paper:

$$\begin{aligned} \text{Region} = & \{ \text{Revoked} \} \uplus \\ & \{ \text{Temporary} \} \times \text{State} \times \text{Rels} \\ & \times (\text{State} \rightarrow (\text{Wor} \xrightarrow[\exists_{pub}]{mon, ne} \text{UPred}(\text{MemSeg}))) \uplus \\ & \{ \text{Permanent} \} \times \text{State} \times \text{Rels} \\ & \times (\text{State} \rightarrow (\text{Wor} \xrightarrow[\exists_{priv}]{mon, ne} \text{UPred}(\text{MemSeg}))) \end{aligned}$$

$$\text{World} = \mathbb{N} \rightarrow \text{Region}$$

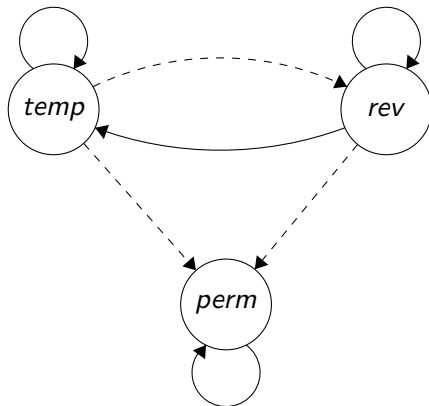
In the Iris mechanization, we use a collection of state transition systems:

$$\Sigma : \mathbb{N} \rightarrow \text{States} \times \mathbb{N} \rightarrow \text{Rels}$$

The world circularity problem is now handled using Iris invariants and saved predicates.

Standard STS

$$\Sigma : \mathbb{N} \rightarrow States \times \mathbb{N} \rightarrow Rels$$



What's new: capability machine viewpoint

- ▶ Mechanized formalization: currently ~ 25000 lines of Iris code
- ▶ At a higher level of abstraction
 - ▶ Step index \rightarrow later modality
 - ▶ World \rightarrow collection of state transition systems

What's new: Iris formalization viewpoint

- ▶ Formalization of a machine level language, with no distinction between program and memory
- ▶ Distinction between well-bracketed and non well-bracketed calls: using public/private transitions

What's new: capability machine viewpoint

- ▶ Mechanized formalization: currently ~ 25000 lines of Iris code
- ▶ At a higher level of abstraction
 - ▶ Step index \rightarrow later modality
 - ▶ World \rightarrow collection of state transition systems

What's new: Iris formalization viewpoint

- ▶ Formalization of a machine level language, with no distinction between program and memory
- ▶ Distinction between well-bracketed and non well-bracketed calls: using public/private transitions

What's new: capability machine viewpoint

- ▶ Mechanized formalization: currently ~ 25000 lines of Iris code
- ▶ At a higher level of abstraction
 - ▶ Step index \rightarrow later modality
 - ▶ World \rightarrow collection of state transition systems

What's new: Iris formalization viewpoint

- ▶ Formalization of a machine level language, with no distinction between program and memory
- ▶ Distinction between well-bracketed and non well-bracketed calls: using public/private transitions

Conclusion

- ▶ Embed a capability machine into Iris
- ▶ Define its program logic
- ▶ Mechanize a unary logical relation for an untyped capability machine language
- ▶ Prove the 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



Lau Skorstengaard, Dominique Devriese, and Lars Birkedal (2018)
Reasoning About a Machine with Local Capabilities
ESOP Programming Languages and Systems 475–501.



Derek Dreyer, Georg Neis, Lars Birkedal (2012)
The impact of higher-order state and control effects on local relational reasoning
Journal of Functional Programming 22(4-5) 477–528.



Derek Dreyer, Amal Ahmed, Lars Birkedal (2011)
Logical Step-Indexed Logical Relations
LMCS 7(2:16).

The Execute Condition

The Execute Condition

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

The Expression Relation

The Expression Relation

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

The Expression Relation

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

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

The Expression Relation

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

$$\text{context}(\Sigma)(r) = \left(\bigstar_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \wedge \text{full_map } r$$

The Expression Relation

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

$$\begin{aligned}\text{context}(\Sigma)(r) = & \left(\bigstar_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \wedge \text{full_map } r \\ & * \text{na_inv } \gamma_{\text{na}} \top\end{aligned}$$

The Expression Relation

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

$$\begin{aligned}\text{context}(\Sigma)(r) = & \left(\bigstar_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \wedge \text{full_map } r \\ & * \text{na_inv } \gamma_{na} \top \\ & * \text{sts_full } \Sigma\end{aligned}$$

The Expression Relation

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

$$\begin{aligned}\text{context}(\Sigma)(r) = & \left(\bigstar_{r_i \mapsto w \in r} r_i \mapsto_r w \right) \wedge \text{full_map } r \\ & * \text{na_inv } \gamma_{\text{na}} \top \\ & * \text{sts_full } \Sigma \\ & * \text{region } \Sigma\end{aligned}$$

The Fundamental Theorem of Logical Relations

The Fundamental Theorem of logical relations

If we can read a region, and every word in that region is safe, then
we can safely execute it

- ▶ "If we can read a region" : $p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}$
- ▶ "and every word in that region is safe":
 $\text{read_write_cond}(p, b, e)$
- ▶ "then we can safely execute it": $\mathcal{E}(\Sigma)((p, g), b, e, a)$

$$(p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}) \implies$$

$$\text{read_write_cond}(p, b, e) \implies \mathcal{E}(\Sigma)((p, g), b, e, a)$$

- ▶ "If we can read a region" : $p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}$
- ▶ "and every word in that region is safe":
 $\text{read_write_cond}(p, b, e)$
- ▶ "then we can safely execute it": $\mathcal{E}(\Sigma)((p, g), b, e, a))$

$$(p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}) \implies$$

$$\text{read_write_cond}(p, b, e) \implies \mathcal{E}(\Sigma)((p, g), b, e, a))$$

- ▶ "If we can read a region" : $p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}$
- ▶ "and every word in that region is safe":
 $\text{read_write_cond}(p, b, e)$
- ▶ "then we can safely execute it": $\mathcal{E}(\Sigma)((p, g), b, e, a))$

$$(p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}) \implies$$

$$\text{read_write_cond}(p, b, e) \implies \mathcal{E}(\Sigma)((p, g), b, e, a))$$

- ▶ "If we can read a region" : $p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}$
- ▶ "and every word in that region is safe":
 $\text{read_write_cond}(p, b, e)$
- ▶ "then we can safely execute it": $\mathcal{E}(\Sigma)((p, g), b, e, a)$

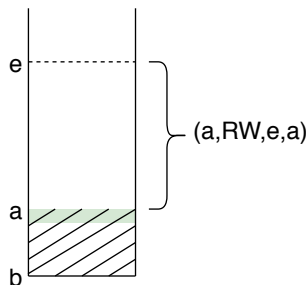
$$(p = \text{RX} \vee p = \text{RWX} \vee p = \text{RWLX}) \implies$$

$$\text{read_write_cond}(p, b, e) \implies \mathcal{E}(\Sigma)((p, g), b, e, a)$$

Reasoning about Unknown Code

Reasoning about Unknown Code

We use the fundamental theorem to reason about calls to an unknown adversary



```
1 push r_stk 1
2 scall r
3 pop r_stk r_1
4 assert r_1 1
5 halt
```

$$\mathcal{E}(\Sigma)(pc) \triangleq \forall r, \mathcal{R}(\Sigma)(r) * \text{context}(\Sigma)(r[PC := pc])$$

* WP Seq (Instr Executable)

$$\{v, v = \text{Halted}V \implies \exists \Sigma' r', \Sigma' \sqsubseteq_{\text{priv}} \Sigma$$

* $\text{context}(\Sigma')(r')\}$

Proving the Fundamental Theorem

$$\begin{array}{l} * \text{ read_write_cond } a \\ a \in [b, e] \end{array} \quad (1)$$

$$* \mathcal{R}(r) \quad (2)$$

$$\begin{array}{l} * \text{ } reg \mapsto_r w \\ reg \hookrightarrow w \in r \end{array} \quad (3)$$

WP Seq (Instr Executable) $\{v, v = \text{Halted}V \implies$
 $\exists \Sigma' r', \Sigma' \sqsubseteq_{priv} \Sigma * \text{context}(\Sigma')(r')\}$

Proving the Fundamental Theorem

$$\begin{array}{l} * \text{ read_write_cond } a \\ a \in [b, e] \end{array} \quad (1)$$

$$* \mathcal{R}(r) \quad (2)$$

$$\begin{array}{l} * \\ \text{reg} \hookrightarrow w \in r \setminus \text{PC} \end{array} \text{reg} \mapsto_r w \quad (3)$$

$$* \text{PC} \mapsto_r (pc_g, pc_p, pc_b, pc_e, pc_a) \quad (4)$$

WP Seq (Instr Executable) $\{v, v = \text{Halted}V \implies$
 $\exists \Sigma' r', \Sigma' \sqsubseteq_{\text{priv}} \Sigma * \text{context}(\Sigma')(r')\}$

Proving the Fundamental Theorem

$$\begin{array}{l} * \text{ read_write_cond } a \\ a \in [b, e] \end{array} \quad (1)$$

$$* \mathcal{R}(r) \quad (2)$$

$$\begin{array}{l} * \\ \text{reg} \hookrightarrow w \in r \setminus \text{PC} \end{array} \text{ reg} \mapsto_r w \quad (3)$$

$$* \text{PC} \mapsto_r (pc_g, pc_p, pc_b, pc_e, pc_a) \quad (4)$$

$$* pc_a \mapsto_a [pc_p]w \quad (5)$$

WP Seq (Instr Executable) $\{v, v = \text{Halted}V \implies$

$$\exists \Sigma' r', \Sigma' \sqsubseteq_{\text{priv}} \Sigma * \text{context}(\Sigma')(r')\}$$

Proving the Fundamental Theorem

$$\text{decode}(w) = \text{Load } \text{dst } \text{src} \quad (1)$$

$$\ast \mathcal{R}(r) \quad (2)$$

$$\ast \quad \text{reg} \mapsto_r w \quad (3)$$

$\text{reg} \hookrightarrow w \in r \setminus \text{PC}$

$$\ast \text{PC} \mapsto_r (pc_g, pc_p, pc_b, pc_e, pc_a) \quad (4)$$

$$\ast pc_a \mapsto_a [pc_p]w \quad (5)$$

WP Seq (Instr Executable) $\{v, v = \text{Halted}V \implies$
 $\exists \Sigma' r', \Sigma' \sqsubseteq_{\text{priv}} \Sigma \ast \text{context}(\Sigma')(r')\}$

Proving the Fundamental Theorem

$$\text{decode}(w) = \text{Load } dst \ src \quad (1)$$

$$\ast \mathcal{R}(r) \quad (2)$$

$$\ast dst \mapsto_r w_{dst} \ast src \mapsto_r (p, g, b, e, a) \quad (3)$$

$$(4)$$

$$\ast PC \mapsto_r (pc_g, pc_p, pc_b, pc_e, pc_a) \quad (5)$$

$$\ast pc_a \mapsto_a [pc_p]w \quad (6)$$

WP Seq (Instr Executable) $\{v, v = \text{Halted}V \implies$

$$\exists \Sigma' r', \Sigma' \sqsubseteq_{priv} \Sigma \ast \text{context}(\Sigma')(r')\}$$