Full Title*

Subtitle†

Anonymous Author(s)

Abstract

Text of abstract ....

Keywords keyword1, keyword2, keyword3

1 Introduction

Text of paper ...

2 Iris Primer

??

3 Iris Embedding of a Capability Machine

The setting: we want to show full abstraction from a high level language, to a low level capability machine. Some basic properties of a high level language: local state encapsulation of well bracketed control flow. These are not natively enforced by a capability machine. When calling an adversary, any guarantees will depend on the calling convention the compiler implements. For LSE, the calling convention should ensure that no passed capability give authority over local state. For WBCR, the situation is more subtle. Skorstensgaard et. al. give detailed explanations of multiple control flow attacks that would happen with a naive calling conventions.

With these goals in mind, it is important not to *accidentally* formalize a language where certain control flow abstractions are enforced. We also want to be as architecture independent as possible. This we are interested in allowing: arbitrary jumps, and (almost) arbitrary capability manipulations (within the given range of authority).

Iris is a higher-order concurrent separation logic framework. Its support of higher-order ghost state makes Iris a great candidate for complex reasoning about *state*. This is exactly what we need for reasoning about a machine, where computation *is* state manipulation.

Machine instructions operate through memory. Typically a machine will have a special register; the program counter, which contains a pointer to an address in memory. That address in turn will contain an integer, which can then be decoded to various machine dependent instructions, such as Load, Store, Jump, etc. Once an instruction has been executed, the program counter is incremented and will now point to the next instruction in memory. A jump now simply consist of a store into the program counter itself.

*Title note

†Subtitle note

PL'18, January 01–03, 2018, New York, NY, USA

2018.

4 Logical Relation

It is well know that defining and reasoning about logical relations for languages with higher-order store, due to the so called World-circularity problem [**?**].

4.1 Definition of the Logical Relation

The logical relation is a unary relation that relates a word to *capability safety* (fig. **??**). The definition is the Iris implementation of the logical relation defined by Skorstensgaard et. al. [**?**]. The value relation $\mathcal{V}$ is an Iris relation of type $World \rightarrow Word \rightarrow iProp$ (we will later discuss what the World type is). We then define the register relation $\mathcal{R}$ as the validity of each word that the register state maps to.

The expression relation should capture what it means for programs to be capability safe. However, as previously discussed, programs in a machine lie in memory, pointed to by the program counter. Strictly speaking, there are no "expressions" in such a machine language. Instead, the expression relation relates Words to the notion of program capability safety, where the word in question will be the program counter state.

4.2 The Iris Implementation of the Context

In the expression relation, the context contains the ghost resources necessary to run a program pointed to by the program counter. When the program has finished running, ownership of these resources is given back. To start out, the running program gets full ownership of all machine registers. A ghost register will be represented by the full ownership of the map from RegName to Word. We write

$$r \mapsto_r w$$

to indicate that registers r currently contains the word w. The context will contain such a statement for each register in the register map.

A program may also need the ghost resources for the pieces of memory it has authority over.

A Appendix

Text of appendix ...

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

Figure 1. Logical Relation for Expressions

$$\mathcal{R}(W)(r) \triangleq \forall (reg : \text{RegName} \backslash \text{PC}), \mathcal{V}(W)(r(reg))$$

Figure 2. Logical Relation for Register States

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

$$\mathcal{V}(W)(((\text{O}, g), b, e, a)) \triangleq \top$$

$$\left.\begin{array}{l}\mathcal{V}(W)(((\text{RO}, g), b, e, a)) \\ \mathcal{V}(W)(((\text{RW}, g), b, e, a)) \\ \mathcal{V}(W)(((\text{RWL}, g), b, e, a))\end{array}\right\} \triangleq \exists\, p', p \sqsubseteq p' * \text{read\_write\_cond(p',b,e)} \qquad \textit{where } p \textit{ is } \textsc{ro}, \textsc{rw}, \textsc{rwl} \textit{ resp.}$$

$$\left.\begin{array}{l}\mathcal{V}(W)(((\text{RX}, g), b, e, a)) \\ \mathcal{V}(W)(((\text{RWX}, g), b, e, a)) \\ \mathcal{V}(W)(((\text{RWLX}, g), b, e, a))\end{array}\right\} \triangleq \exists\, p', p \sqsubseteq p' * \text{read\_write\_cond(p',b,e)} \\ * \ \square \ \text{exec\_cond(W)(p,g,b,e)} \qquad \textit{where } p \textit{ is } \textsc{rx}, \textsc{rwx}, \textsc{rwlx} \textit{ resp.}$$

$$\mathcal{V}(W)(((\text{E}, g), b, e, a)) \triangleq \square \ \text{enter\_cond}(W)(\text{g,b,e,a})$$

Figure 3. Logical Relation for Words

$$\text{read\_write\_cond}(p, b, e) \triangleq \mathop{\Large *}_{a \in [b, e]} \text{rel}(a, p, \mathcal{V})$$

$$\text{exec\_cond(W)(p,g,b,e)} \triangleq \forall a \in [b\ e], W' \sqsubseteq^{g} W. \ \triangleright \ \mathcal{E}(W')(((p, g), b, e, a))$$

$$\text{enter\_cond(W)(g,b,e,a)} \triangleq \forall W' \sqsubseteq^{g} W. \ \triangleright \ \mathcal{E}(W)(((\text{RX}, g), b, e, a))$$

$$\text{where } (\sigma', \rho'_{pub}, \rho'_{priv}) \sqsubseteq^{Global} (\sigma, \rho_{pub}, \rho_{priv}) = (\sigma', \rho'_{priv}) \sqsubseteq (\sigma, \rho_{priv})$$
$$\text{and } (\sigma', \rho'_{pub}, \rho'_{priv}) \sqsubseteq^{Local} (\sigma, \rho_{pub}, \rho_{priv}) = (\sigma', \rho'_{pub}) \sqsubseteq (\sigma, \rho_{pub}))$$

Figure 4. Capability Conditions