Modal Logic and Capability-Safety

Neel Krishnaswami
University of Cambridge

FLOLAC 2024
Taipei, Taiwan

An Effectful Programming Language

$A ::= 1 \mid A \to B \mid \mathbb{N} \mid \mathrm{Chan}$

$e ::= () \mid \lambda x{:}A.\, e \mid e\ e' \mid n \mid x \mid \mathrm{print}(e, e')$
$\quad\mid \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2$

$\Gamma ::= \cdot \mid \Gamma, x : A$

$$\boxed{\Gamma \vdash e : A}$$

Typing Rules

$$\frac{}{\Gamma \vdash () : 1} \qquad \frac{}{\Gamma \vdash n : \mathbb{N}}$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x{:}A.\, e : A \to B} \qquad \frac{\Gamma \vdash e : A \to B \qquad \Gamma \vdash e' : A}{\Gamma \vdash e\ e' : B}$$

$$\frac{\Gamma \vdash e : \mathrm{Chan} \qquad \Gamma \vdash e' : \mathbb{N}}{\Gamma \vdash \mathrm{print}(e, e') : 1}$$

$$\frac{\Gamma \vdash e_1 : A \qquad \Gamma, x : A \vdash e_2 : C}{\Gamma \vdash \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 : C} \qquad \frac{x : A \in \Gamma}{\Gamma \vdash x : A}$$

Operational Semantics

$v ::= () \mid \lambda x{:}A.e \mid n \mid c$ ← channel names

$a ::= \cdot \mid wr(c, n)$

$$e \xrightarrow{a} e'$$

"e transitions to e', possibly writing a"

We write $e \longrightarrow e'$ when $a = \cdot$

Reduction Rules

$$\frac{}{(\lambda x{:}A.\,e)\ v \longrightarrow [v/x]e}$$

$$\frac{}{\mathrm{print}(c,n) \xrightarrow{\mathrm{wr}(c,n)} ()}$$

$$\frac{e_1 \xrightarrow{a} e_1'}{(e_1\ e_2) \xrightarrow{a} (e_1'\ e_2)}$$

$$\frac{e_2 \xrightarrow{a} e_2'}{v\ e_2 \xrightarrow{a} v\ e_2'}$$

$$\frac{}{\mathrm{let}\ x = v\ \mathrm{in}\ e_2 \longrightarrow [v/x]e_2}$$

$$\frac{e_1 \xrightarrow{a} e_1'}{\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 \xrightarrow{a} \mathrm{let}\ x = e_1'\ \mathrm{in}\ e_2}$$

- Call-by-value evaluation order
- Impure language

Evaluation Sequence

Consider an evaluation sequence

$$e_0 \xrightarrow{a_0} e_1 \xrightarrow{\cdot} e_2 \xrightarrow{a_1} e_3 \xrightarrow{\cdot} e_4 \xrightarrow{a_2} v$$

We write multistep evaluation as:

$$e_0 \xrightarrow{a_0 a_1 a_2}{}^{*} v$$

Impurity: Evaluation Order

```
let x = print(c, 0) in
let y = print(c, 1) in
()
```

$\xrightarrow{wr(c,0)\cdot wr(c,1)}{}^{*}\ ()$

```
let y = print(c, 1) in
let x = print(c, 0) in
()
```

$\xrightarrow{wr(c,1)\cdot wr(c,0)}{}^{*}\ ()$

Reordering expressions is not allowed

Impurity: Dropping Expressions

$F \triangleq \lambda f : 1 \to 1.\ f()$ vs $G \triangleq \lambda f : 1 \to 1.\ ()$

$$F\ (\lambda x : 1.\ \mathrm{print}(c, 0)) \xrightarrow{Wr(c,0)}{}^{*}\ ()$$

$$G\ (\lambda x : 1.\ \mathrm{print}(c, 0)) \longrightarrow^{*}\ ()$$

Impurity: Duplicating Expressions

$$\begin{array}{l} \text{let } x = \text{print}(c,0) \text{ in} \\ \text{let } y = x \text{ in} \\ () \end{array} \xrightarrow{\;Wr(c,0)\;}{}^{*} \; ()$$

$$\begin{array}{l} \text{let } x = \text{print}(c,0) \text{ in} \\ \text{let } y = \text{print}(c,0) \text{ in} \\ () \end{array} \xrightarrow{\;Wr(c,0)\cdot Wr(c,0)\;}{}^{*} \; ()$$

Managing Effects with Monads

Introduce $T(A)$ such that

$$\text{return} : A \to T(A)$$

$$\text{bind} : T(A) \to (A \to T\,B) \to T(B)$$

$$\text{print} : \text{Chan} \times \mathbb{N} \to T\,1$$

Now $\text{print}\,(c, n) : T(1)$

$e : T(A) \rightsquigarrow e$ may perform effects

Capability-Based Security

Traditional OS security:

- Anyone can refer to (e.g.) files by name (e.g. /home/neelk/foo.md)
- OS checks whether access is allowed via access control list

Capability-Based Security

Alternative (from 1970s! used in Fuschia)

1. Each object (e.g. file) has a unique, unforgeable id
2. Ids combine identity + authority
3. Clients control access via parameter-passing

Capability-Safety

All effects controlled by capabilities

Ambient Authority

$print\_int : \mathbb{N} \to 1$

Ambient Authority

$$\text{print\_int} : \mathbb{N} \longrightarrow 1$$

Calling print(0) does a write without a capability

Ambient Authority

$$\text{print\_int} : \text{String} \times \mathbb{N} \to \mathbb{1}$$

Example: print_int("foo.txt", 0)

Ambient Authority

$\text{print\_int} : \text{String} \times \mathbb{N} \to 1$

Example: print_int("foo.txt", 0)

Anyone can invent any string: filenames are <u>forgeable</u>

A Capability-Safe API

$$\text{print\_int} : \text{chan} \times \mathbb{N} \to 1$$

- Chan: abstract type of channels
- chan values are unforgeable (due to memory-safety + abstraction)

Capability-Safe Languages

- Imperative, memory-safe language
- Standard library APIs are <u>fully</u> capability-safe

Capability-Safe Languages

- Imperative, memory-safe language
- Standard library APIs are fully capability-safe

- Our little language is capability safe!

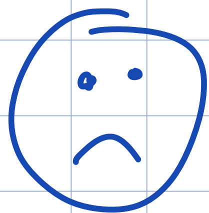
Capability-Taming

- Any ops with ambient authority break capability-safety

Capability-Taming

- Any ops with ambient authority break capability-safety
- To make a language capability-safe: completely rewrite std library

Capability-Taming

- Any ops with ambient authority break capability-safety
- To make a language capability-safe: completely rewrite std library
- E.g. CapJava, Caja (for JS) 

Capabilities and Types

Can we use <u>types</u> to track capabilities?

Values of Type	Own Capabilities?
1	NO
IN	NO
chan	YES

Capabilities and Types

Can we use <u>types</u> to track capabilities?

Values of Type	Own Capabilities?
1	NO
$\mathbb{N}$	NO
chan	YES
$A \times B$	?

Capabilities and Types

Can we use <u>types</u> to track capabilities?

Values of Type	Own Capabilities?
1	NO
$\mathbb{N}$	NO
chan	YES
$A \times B$	?
$\mathbb{N} \to 1$	?

Closures Capture Capabilities

$$f : \mathrm{chan} \vdash \lambda n : \mathbb{N}.\ \mathrm{print}(f, n) : \mathbb{N} \to 1$$

- fn is a closure capturing f
- So $\mathbb{N} \to 1$ accesses a capability even though $\mathbb{N}$ and 1 don't

Modal Logic to the Rescue

$\Box A$ values of A with <u>NO</u> capabilities

Modal Logic to the Rescue

$\Box A$ values of A with <u>NO</u> capabilities

$K : \Box A \times \Box B \longrightarrow \Box (A \times B)$

$T : \Box A \longrightarrow A$

$4 : \Box A \longrightarrow \Box \Box A$

Modal Logic to the Rescue

$\Box A$ values of A with **NO** capabilities

$K : \Box A \times \Box B \longrightarrow \Box(A \times B)$

$T : \Box A \longrightarrow A$

$4 : \Box A \longrightarrow \Box\Box A$

} Each axiom makes sense in terms of denial

An Effectful Modal Language

$A ::= 1 \mid A \to B \mid \mathbb{N} \mid \mathrm{Chan} \mid \Box A$

$e ::= () \mid \lambda x{:}A.\, e \mid e\ e' \mid n \mid x \mid \mathrm{print}(e, e')$
$\quad \mid \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2$
$\quad \mid \mathrm{box}(e) \mid \mathrm{let}\ \mathrm{box}(x) = e_1\ \mathrm{in}\ e_2$

$\Gamma ::= \cdot \mid \Gamma, x : A \qquad \Delta ::= \cdot \mid \Delta, x : A$

$$\Delta ; \Gamma \vdash e : A$$

Typing Rules

$$\frac{}{\Delta;\Gamma \vdash () : 1} \qquad \frac{}{\Delta;\Gamma \vdash n : \mathbb{N}}$$

$$\frac{\Delta;\Gamma, x:A \vdash e : B}{\Delta;\Gamma \vdash \lambda x{:}A.e : A \to B} \qquad \frac{\Delta;\Gamma \vdash e : A \to B \quad \Delta;\Gamma \vdash e' : A}{\Delta;\Gamma \vdash e\, e' : B}$$

$$\frac{\Delta;\Gamma \vdash e : \mathrm{Chan} \qquad \Delta;\Gamma \vdash e' : \mathbb{N}}{\Delta;\Gamma \vdash \mathrm{print}(e, e') : 1}$$

$$\frac{\Delta;\Gamma \vdash e_1 : A \qquad \Delta;\Gamma, x:A \vdash e_2 : C}{\Delta;\Gamma \vdash \mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 : C} \qquad \frac{x:A \in \Gamma}{\Delta;\Gamma \vdash x : A}$$

Modal Typing Rules

$$\frac{x : A \in \Delta}{\Delta ; \Gamma \vdash \underline{x} : A}$$

$$\frac{\Delta ; \cdot \vdash e : A}{\Delta ; \Gamma \vdash \mathrm{box}(e) : \Box A}$$

$$\frac{\Delta ; \Gamma \vdash e_1 : \Box A \qquad \Delta , x : A ; \Gamma \vdash e_2 : C}{\Delta ; \Gamma \vdash \mathrm{let}\ \mathrm{box}(x) = e_1\ \mathrm{in}\ e_2 : C}$$

Reduction Rules, with Box

$$\frac{e \xrightarrow{a} e'}{\mathrm{box}(e) \xrightarrow{a} \mathrm{box}(e')}$$

$$\frac{}{\mathrm{let}\ \mathrm{box}(x) = \mathrm{box}(v)\ \mathrm{in}\ e \longrightarrow [v/x]e}$$

$$\frac{}{(\lambda x : A.\, e)\ v \longrightarrow [v/x]e}$$

$$\frac{}{\mathrm{print}(c, n) \xrightarrow{wr(c,n)} ()}$$

$$\frac{e_1\ e_2 \xrightarrow{a} e_1'\ e_2}{(e_1\ e_2) \xrightarrow{a} (e_1'\ e_2)}$$

$$\frac{e_2 \xrightarrow{a} e_2'}{v\ e_2 \xrightarrow{a} v\ e_2'}$$

$$\frac{}{\mathrm{let}\ x = v\ \mathrm{in}\ e_2 \longrightarrow [v/x]e_2}$$

$$\frac{e_1 \xrightarrow{a} e_1'}{\mathrm{let}\ x = e_1\ \mathrm{in}\ e_2 \xrightarrow{a} \mathrm{let}\ x = e_1'\ \mathrm{in}\ e_2}$$

Type-based Reasoning

$\text{safe\_print} : \Box(\text{Chan} \to \mathbb{N} \to 1)$

Type-based Reasoning

$$\text{safe\_print} : \square(\text{Chan} \to \mathbb{N} \to 1)$$

- safe_print owns no channels

Type-based Reasoning

$$\text{safe\_print} : \square(\text{Chan} \to \mathbb{N} \to 1)$$

- safe_print owns no channels
- All channels it can access come from chan argument

Type-based Reasoning

$$\text{safe\_print} : \Box(\text{Chan} \to \mathbb{N} \to 1)$$

- safe-print owns no channels
- All channels it can access come from chan argument
- So it can <u>only</u> write to channels it is given : capability-safe

Type-based Reasoning

multi-print : $\square(\text{List Chan} \to \mathbb{N} \to 1)$

Type-based Reasoning

multi-print: $\square(\text{List Chan} \to \mathbb{N} \to 1)$

- multi-print owns no channels

Type-based Reasoning

$$\text{multi-print} : \Box\,(\text{List Chan} \to \mathbb{N} \to 1)$$

- multi-print owns no channels
- All channels it can access come from List chan argument

Type-based Reasoning

$$\text{multi-print} : \Box (\text{List Chan} \to \mathbb{N} \to 1)$$

- multi-print owns no channels
- All channels it can access come from List chan argument
- So it can <u>only</u> write to channels it is given : capability-safe

Encoding Purity

$$f : \Box(\Box A \to B)$$

- f owns no channels
- f's argument owns no channels
- So f(v) runs with no channels
- It does no writes
- f is purely functional!

Encoding Purity

$$f : \square(\square A \to B)$$

Encoding Purity

$$f : \Box(\Box A \to B)$$

- f owns no channels

Encoding Purity

$$f : \Box(\Box A \to B)$$

- f owns no channels
- f's argument owns no channels

Encoding Purity

$$f : \Box(\Box A \to B)$$

- f owns no channels
- f's argument owns no channels
- So f(v) runs with no channels
- It does no writes

Encoding Purity

$$f : \Box(\Box A \to B)$$

- f owns no channels
- f's argument owns no channels
- So $f(v)$ runs with no channels
- It does no writes
- f is purely functional!

Capability Taming with $\Box$

- $\Box(A \to B)$ capability-safe functions
- As you update the stdlib, mark updated functions w/ box
- Modal discipline tracks capability safety!
- GRADUAL rewrites now possible!

Proving It?

- How can we prove □A enforces capability safety?
- Needs <u>logical</u> <u>relations</u>