Rows and Capabilities as Modal Effects

Wenhao Tang Sam Lindley

POPL, Rennes, France, 16th Jan 2026

THE UNIVERSITY of EDINBURGH

Effects and Effect Types

Effects are the way programs interact with their environment

Including I/O, concurrency, exceptions, nondeterminism, probability

Effects and Effect Types

Effects are the way programs interact with their environment

Including I/O, concurrency, exceptions, nondeterminism, probability

Effect types (or effect systems) statically track use of effects in types

Many recent practical effect systems as based on **rows** (Koka) or **capabilities** (Effekt)

Rows and Capabilities

Row-Based Effect Types as in Koka

$$A \to^{\boldsymbol{E}} B$$

A function that may use effects in the row E when applied

Rows and Capabilities

Row-Based Effect Types as in Koka

$$A \to^{\boldsymbol{E}} B$$

A function that may use effects in the row E when applied

Capability-Based Effect Types as in Effekt

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

A block (second-class function) that binds capabilities $\overline{f : T}$ and may use them

Rows and Capabilities

Row-Based Effect Types as in Koka

$$A \rightarrow^{\boldsymbol{E}} B$$

A function that may use effects in the row E when applied

Capability-Based Effect Types as in Effekt

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

A block (second-class function) that binds capabilities $\overline{f : T}$ and may use them

Question: How to compare them formally and systematically?

Rows and Capabilities

Row-Based Effect Types as in Koka

$$A \to^{\boldsymbol{E}} B$$

A function that may use effects in the row E when applied

Capability-Based Effect Types as in Effekt

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

A block (second-class function) that binds capabilities $\overline{f : T}$ and may use them

Question: How to compare them formally and systematically?

Challenge: Their effect tracking mechanisms are ***entangled*** *with function types.*

A Detour: CBV, CBN, and CBPV

Translating between CBV and CBN is rather involved

A Detour: CBV, CBN, and CBPV

Translating between CBV and CBN is rather involved

because when to suspend and force computations is ***entangled*** with functions

A Detour: CBV, CBN, and CBPV

Translating between CBV and CBN is rather involved

because when to suspend and force computations is ***entangled*** with functions

CBPV smoothly subsumes both by ***decoupling*** thunking and forcing from functions

A Detour: CBV, CBN, and CBPV

Translating between CBV and CBN is rather involved

because when to suspend and force computations is ***entangled*** with functions

CBPV smoothly subsumes both by ***decoupling*** thunking and forcing from functions

Why not decoupling effect tracking from function types?

A Uniform Framework for Effect Types

Modal Effect Types (Met) *decouple* effect tracking from function types via modalities

Modal Effect Types

WENHAO TANG, The University of Edinburgh, United Kingdom
LEO WHITE, Jane Street, United Kingdom
STEPHEN DOLAN, Jane Street, United Kingdom
DANIEL HILLERSTRÖM, The University of Edinburgh, United Kingdom
SAM LINDLEY, The University of Edinburgh, United Kingdom
ANTON LORENZEN, The University of Edinburgh, United Kingdom

A Uniform Framework for Effect Types

Modal Effect Types (MET) *decouple* effect tracking from function types via modalities

Modal Effect Types

WENHAO TANG, The University of Edinburgh, United Kingdom
LEO WHITE, Jane Street, United Kingdom
STEPHEN DOLAN, Jane Street, United Kingdom
DANIEL HILLERSTRÖM, The University of Edinburgh, United Kingdom
SAM LINDLEY, The University of Edinburgh, United Kingdom
ANTON LORENZEN, The University of Edinburgh, United Kingdom

This OOPSLA'25 paper proposes MET and shows how it provides modular effectful types in practice without using effect polymorphism

A Uniform Framework for Effect Types

Modal Effect Types (Met) *decouple* effect tracking from function types via modalities

Our contribution: A framework for encoding and comparing effect systems based on Met

A Uniform Framework for Effect Types

Modal Effect Types (MET) ***decouple*** effect tracking from function types via modalities

Our contribution: A framework for encoding and comparing effect systems based on MET

Koka as MET: $$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$

Effekt as MET: $$[\![(\overline{A}, \overline{f : T}) \Rightarrow B]\!] = \forall \overline{f^{*}}. \langle \overline{f^{*}} \rangle (\overline{[\![A]\!]} \to \overline{[f^{*}][\![T]\!]} \to [\![B]\!])$$

Modal Effect Types

Effect Contexts

A typing judgement tracks the ***ambient effect context***

$$\vdash \lambda x.\mathbf{do}\ \texttt{yield}\ x \ : \ \texttt{Int} \rightarrow \texttt{1} \ @ \ \texttt{yield}$$

Effect Contexts

A typing judgement tracks the ***ambient effect context***

$$\vdash \lambda x.\textbf{do}\ \texttt{yield}\ x \ :\ \texttt{Int} \to \texttt{1}\ @\ \texttt{yield}$$

Effect contexts propagate through types and terms

$$\vdash \lambda f.\lambda x.f\ x \ :\ (\texttt{Int} \to \texttt{1}) \to \texttt{Int} \to \texttt{1}\ @\ \texttt{yield}$$

Effect Contexts

A typing judgement tracks the ***ambient effect context***

$$\vdash \lambda x.\mathbf{do}\ \texttt{yield}\ x \ :\ \texttt{Int} \to \texttt{1} \ @\ \texttt{yield}$$

Effect contexts propagate through types and terms

$$\vdash \lambda f.\lambda x.f\ x \ :\ \underset{@\ \texttt{yield}}{(\texttt{Int} \to \texttt{1})} \to \underset{@\ \texttt{yield}}{\texttt{Int} \to \texttt{1}} \ @\ \texttt{yield}$$

Effect Contexts

A typing judgement tracks the ***ambient effect context***

$$\vdash \lambda x.\textbf{do}\ \texttt{yield}\ x \ :\ \texttt{Int} \to \texttt{1} \ @\ \texttt{yield}$$

Effect contexts propagate through types and terms

$$\vdash \lambda f.\lambda x.f\ x \ :\ (\underset{@\ \texttt{yield}}{\texttt{Int} \to \texttt{1}}) \to \underset{@\ \texttt{yield}}{\texttt{Int} \to \texttt{1}} \ @\ \texttt{yield}$$

A natural notion of sub-effecting

$$\vdash \lambda x.\textbf{do}\ \texttt{yield}\ x \ :\ \texttt{Int} \to \texttt{1} \ @\ \texttt{yield}, \texttt{ask}$$

Absolute Modalities

Effect contexts are part of typing judgements, ***not types***

We use *modalities* to track effects in types

Absolute Modalities

Effect contexts are part of typing judgements, ***not types***

We use *modalities* to track effects in types

An ***absolute modality*** $[E]$ changes the ambient effect context to E

$$\frac{}{\Gamma \vdash \mathbf{mod}_{[\mathtt{yield}]}\,(\lambda x.\mathbf{do}\ \mathtt{yield}\ x) : [\mathtt{yield}](\mathtt{Int} \to 1) \ @\ \mathtt{ask}}$$

Absolute Modalities

Effect contexts are part of typing judgements, ***not types***

We use *modalities* to track effects in types

An ***absolute modality*** $[E]$ changes the ambient effect context to E

$$\frac{}{\Gamma \vdash \mathbf{mod}_{[\mathtt{yield}]}\,(\lambda x.\mathbf{do}\ \mathtt{yield}\ x) \ :\ [\mathtt{yield}](\mathtt{Int} \to \mathtt{1})\ @\ \mathtt{ask}}$$

The absolute modality [yield] changes the ambient effect context ask to yield

Absolute Modalities

Effect contexts are part of typing judgements, ***not types***

We use *modalities* to track effects in types

An ***absolute modality*** $[E]$ changes the ambient effect context to E

$$\frac{\Gamma, \text{🔒}_{[\texttt{yield}]} \vdash \lambda x.\mathbf{do}\ \texttt{yield}\ x : \texttt{Int} \to \texttt{1}\ @\ \texttt{yield}}{\Gamma \vdash \mathbf{mod}_{[\texttt{yield}]}\ (\lambda x.\mathbf{do}\ \texttt{yield}\ x) : [\texttt{yield}](\texttt{Int} \to \texttt{1})\ @\ \texttt{ask}}$$

The absolute modality $[\texttt{yield}]$ changes the ambient effect context `ask` to `yield`

The lock $\text{🔒}_{[\texttt{yield}]}$ tracks the changes of effect contexts

Relative Modalities

A ***relative modality*** $\langle E \rangle$ extends the ambient effect context with effects E

$$\frac{}{\Gamma \vdash \mathbf{mod}_{\langle \mathtt{yield} \rangle}\ (\lambda x.\mathbf{do}\ \mathtt{yield}\ (\mathbf{do}\ \mathtt{ask}\ ()))\ :\ \langle \mathtt{yield} \rangle (\mathtt{Int} \to \mathtt{1})\ @\ \mathtt{ask}}$$

Relative Modalities

A ***relative modality*** $\langle E \rangle$ extends the ambient effect context with effects E

$$\frac{}{\Gamma \vdash \mathbf{mod}_{\langle \mathtt{yield} \rangle}\ (\lambda x.\mathbf{do}\ \mathtt{yield}\ (\mathbf{do}\ \mathtt{ask}\ ()))\ :\ \langle \mathtt{yield} \rangle (\mathtt{Int} \to \mathtt{1})\ @\ \mathtt{ask}}$$

The relative modality $\langle \mathtt{yield} \rangle$ extends the ambient effect context `ask` with `yield`

Relative Modalities

A ***relative modality*** $\langle E \rangle$ extends the ambient effect context with effects E

$$\frac{\Gamma, \text{🔒}_{\langle \mathtt{yield} \rangle} \vdash \lambda x.\mathbf{do}\ \mathtt{yield}\ (\mathbf{do}\ \mathtt{ask}\ ()) : \mathtt{Int} \to \mathtt{1}\ @\ \mathtt{yield}, \mathtt{ask}}{\Gamma \vdash \mathbf{mod}_{\langle \mathtt{yield} \rangle}\ (\lambda x.\mathbf{do}\ \mathtt{yield}\ (\mathbf{do}\ \mathtt{ask}\ ())) : \langle \mathtt{yield} \rangle (\mathtt{Int} \to \mathtt{1})\ @\ \mathtt{ask}}$$

The relative modality $\langle \mathtt{yield} \rangle$ extends the ambient effect context `ask` with `yield`

Locks Control the Accessibility of Variables

An invalid judgement

$$f : \mathtt{Int} \to \mathtt{1} \nvdash \mathbf{mod}_{[\mathtt{yield}]}\,(\lambda x.f\,x) \;:\; [\mathtt{yield}](\mathtt{Int} \to \mathtt{1}) \;@\; \mathtt{ask}$$

Locks Control the Accessibility of Variables

An invalid judgement

$$\underset{@\ \mathtt{ask}}{f : \mathtt{Int} \to \mathtt{1}} \nvdash \mathbf{mod}_{[\mathtt{yield}]}\left(\underset{@\ \mathtt{yield}}{\lambda x. f\, x}\right) : [\mathtt{yield}](\mathtt{Int} \to \mathtt{1})\ @\ \mathtt{ask}$$

because f may use the `ask` operation

Locks Control the Accessibility of Variables

An invalid judgement

$$\underset{@\ \mathtt{ask}}{f : \mathtt{Int} \to \mathtt{1}} \nvdash \mathbf{mod}_{[\mathtt{yield}]} \left(\underset{@\ \mathtt{yield}}{\lambda x. f\, x} \right) : [\mathtt{yield}](\mathtt{Int} \to \mathtt{1}) \ @\ \mathtt{ask}$$

because f may use the `ask` operation

MET rejects its expected premise

$$f : \mathtt{Int} \to \mathtt{1}, \text{🔒}_{[\mathtt{yield}]} \nvdash \lambda x. f\, x : \mathtt{Int} \to \mathtt{1} \ @\ \mathtt{yield}$$

by not allowing f to be used after the lock $\text{🔒}_{[\mathtt{yield}]}$

Modality Elimination

We can make the premise well-typed by annotating the binding of f with [] (or [`yield`])

$$f :_{[]} \mathtt{Int} \xrightarrow[@\;\cdot]{} \mathtt{1}, \text{🔒}_{[\mathtt{yield}]} \vdash \lambda x^{\mathtt{Int}}.f\,x \;:\; \mathtt{Int} \to \mathtt{1} \;@\; \mathtt{yield}$$

Modality Elimination

We can make the premise well-typed by annotating the binding of f with $[]$ (or $[\texttt{yield}]$)

$$f :_{[]} \texttt{Int}_{@} \to_{\cdot} \texttt{1}, \text{🔒}_{[\texttt{yield}]} \vdash \lambda x^{\texttt{Int}}.f\,x \;:\; \texttt{Int} \to \texttt{1} \;@\; \texttt{yield}$$

Such a binding is introduced by modality elimination (the default annotation is $\langle\rangle$)

$$\frac{\vdash V \;:\; [](\texttt{Int} \to \texttt{1}) \qquad f :_{[]} \texttt{Int} \to \texttt{1} \vdash M \;:\; A \;@\; \texttt{yield}}{\vdash \textbf{let mod}_{[]}\; f = V \;\textbf{in}\; M \;:\; A \;@\; \texttt{yield}}$$

Rows as Modal Effects

System F^ϵ: Row-Based Effect Types à la Koka

System F^ϵ formalises Koka's row-based effect system

Effect Handlers, Evidently

NINGNING XIE, Microsoft Research, USA
JONATHAN IMMANUEL BRACHTHÄUSER, University of Tübingen, Germany
DANIEL HILLERSTRÖM, The University of Edinburgh, United Kingdom
PHILIPP SCHUSTER, University of Tübingen, Germany
DAAN LEIJEN, Microsoft Research, USA

System F^{ϵ}: Row-Based Effect Types à la Koka

System F^{ϵ} formalises Koka's row-based effect system

Key idea: annotate each function arrow with a row of effects

$$A \xrightarrow{\boldsymbol{E}} B$$

A function that may use effects in the row E when applied

Examples of System F^{ϵ}

$$A \to^{E} B$$

Examples of System F^{ϵ}

$$A \rightarrow^{\boldsymbol{E}} B$$

A first-order effectful function

$$\lambda x.\mathbf{do}\ \texttt{yield}\ x \ :\ \texttt{Int} \rightarrow^{\texttt{yield}} \texttt{1}$$

Examples of System F^ϵ

$$A \to^{E} B$$

A first-order effectful function

$$\lambda x.\textbf{do}\ \texttt{yield}\ x \ : \ \texttt{Int} \to^{\texttt{yield}} \texttt{1}$$

A higher-order effect-polymorphic function

$$\Lambda \varepsilon.\lambda f.\lambda x.f\ x \ : \ \forall \varepsilon.(\texttt{Int} \to^{\varepsilon} \texttt{1}) \to \texttt{Int} \to^{\varepsilon} \texttt{1}$$

Encoding System F^{ϵ} in Met

$$[\![A \to^{E} B]\!] = [\![\![E]\!]\!]([\![A]\!] \to [\![B]\!])$$

Encoding System F^{ϵ} in MET

$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$

Encoding of the first-order effectful function

$$[\![\texttt{Int} \to^{\texttt{yield}} \texttt{1}]\!] \quad = \quad [\texttt{yield}](\texttt{Int} \to \texttt{1})$$

Encoding System F^{ϵ} in Met

$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$

Encoding of the first-order effectful function

$$[\![\texttt{Int} \to^{\texttt{yield}} \texttt{1}]\!] = [\texttt{yield}](\texttt{Int} \to \texttt{1})$$
$$[\![\lambda x.\mathbf{do}\ \texttt{yield}\ x]\!] = \mathbf{mod}_{[\texttt{yield}]}\ (\lambda x.\mathbf{do}\ \texttt{yield}\ x)$$

Encoding System F^{ϵ} in MET

$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$

Encoding of the first-order effectful function

$$[\![\mathtt{Int} \to^{\mathtt{yield}} \mathtt{1}]\!] \quad = \quad [\mathtt{yield}](\mathtt{Int} \to \mathtt{1})$$

Encoding of the higher-order effect-polymorphic function

$$[\![\forall \varepsilon.(\mathtt{Int} \to^{\varepsilon} \mathtt{1}) \to \mathtt{Int} \to^{\varepsilon} \mathtt{1}]\!] \quad = \quad \forall \varepsilon.[]([\varepsilon](\mathtt{Int} \to \mathtt{1}) \to [\varepsilon](\mathtt{Int} \to \mathtt{1}))$$

Encoding System F^ϵ in MET

$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$

Encoding of the first-order effectful function

$$[\![\texttt{Int} \to^{\texttt{yield}} \texttt{1}]\!] \quad = \quad [\texttt{yield}](\texttt{Int} \to \texttt{1})$$

Encoding of the higher-order effect-polymorphic function

$$\begin{aligned}
[\![\forall\varepsilon.(\texttt{Int} \to^{\varepsilon} \texttt{1}) \to \texttt{Int} \to^{\varepsilon} \texttt{1}]\!] \quad &= \quad \forall\varepsilon.[]([\varepsilon](\texttt{Int} \to \texttt{1}) \to [\varepsilon](\texttt{Int} \to \texttt{1})) \\
[\![\Lambda\varepsilon.\lambda f.\lambda x.f\,x]\!] \quad &= \quad \Lambda\varepsilon.\textbf{mod}_{[]}\ (\lambda f.\textbf{mod}_{[\varepsilon]}\ (\lambda x.\textbf{let mod}_{[\varepsilon]}\ f' = f\ \textbf{in}\ f'\ x))
\end{aligned}$$

Capabilities as Modal Effects

System C: Capability-Based Effect Types à la Effekt

System C formalises Effekt's capability-based effect system

Effects, Capabilities, and Boxes

From Scope-Based Reasoning to Type-Based Reasoning and Back

JONATHAN IMMANUEL BRACHTHÄUSER, University of Tübingen, Germany
PHILIPP SCHUSTER, University of Tübingen, Germany
EDWARD LEE, University of Waterloo, Canada
ALEKSANDER BORUCH-GRUSZECKI, EPFL, Switzerland

System C: Capability-Based Effect Types à la Effekt

System C formalises Effekt's capability-based effect system

Key idea: treat effects as capabilities provided by the context

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

A block (i.e., second-class function) that binds

- a list of arguments of types $\overline{A}$, and
- a list of capabilities $\overline{f : T}$ (i.e., block variables)

Examples of System C

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

Examples of System C

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

A first-order block that call the capability `yield` from the context

$$\texttt{yield} : \texttt{Int} \Rightarrow \texttt{1} \vdash \{(x : \texttt{Int}) \Rightarrow \texttt{yield}(x)\} \; : \; \texttt{Int} \Rightarrow \texttt{1}$$

Examples of System C

$$(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B$$

A first-order block that call the capability `yield` from the context

$$\texttt{yield} : \texttt{Int} \Rightarrow \texttt{1} \vdash \{(x : \texttt{Int}) \Rightarrow \texttt{yield}(x)\} \; : \; \texttt{Int} \Rightarrow \texttt{1}$$

A higher-order block that binds a capability f (a block variable)

$$\{(x : \texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow f(x)\} \; : \; (\texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \texttt{1}$$

Examples of System C (Contd.)

Blocks / capabilities are second-class, i.e., they cannot escape

$$\{(x : \texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow f(x)\} \;\; : \;\; (\texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \texttt{1}$$

Examples of System C (Contd.)

Blocks / capabilities are second-class, i.e., they cannot escape

$$\{(x : \texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow f(x)\} \;:\; (\texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \texttt{1}$$

In order to define a curried version we need to use boxes

$$\{(f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \textbf{box}\ \{(x : \texttt{Int}) \Rightarrow f(x)\}\} \;:\; (f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow (\texttt{Int} \Rightarrow \texttt{1}\ \textbf{at}\ \{f\})$$

Examples of System C (Contd.)

Blocks / capabilities are second-class, i.e., they cannot escape

$$\{(x : \texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow f(x)\} \quad : \quad (\texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \texttt{1}$$

In order to define a curried version we need to use boxes

$$\{(f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \textbf{box}\ \{(x : \texttt{Int}) \Rightarrow f(x)\}\} \quad : \quad (f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow (\texttt{Int} \Rightarrow \texttt{1}\ \textbf{at}\ \{f\})$$

box $\cdots$ turns a block into a first-class value

The result type $\texttt{Int} \Rightarrow \texttt{1}\ \textbf{at}\ \{f\}$ tracks that this boxed block uses the capability f

Encoding System C in Met

$$[\![(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B]\!] = \forall \overline{\boldsymbol{f}^*}.\langle \overline{\boldsymbol{f}^*} \rangle (\overline{[\![A]\!]} \to \overline{[\boldsymbol{f}^*][\![T]\!]} \to [\![B]\!])$$

Encoding System C in $\textsc{Met}$

$$[\![(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B]\!] = \forall \overline{\boldsymbol{f}^*}. \langle \overline{\boldsymbol{f}^*} \rangle (\overline{[\![A]\!]} \to \overline{[\boldsymbol{f}^*][\![T]\!]} \to [\![B]\!])$$

It looks quite involved since a block construction in System C does several things

$$\{(x : \mathtt{Int}, f : \mathtt{Int} \Rightarrow \mathtt{1}) \Rightarrow f(x)\} \;\; : \;\; (\mathtt{Int}, f : \mathtt{Int} \Rightarrow \mathtt{1}) \Rightarrow \mathtt{1}$$

(1) bind a both term- and type-level capability f

(2) this f may be invoked in the block body

(3) any capability from the context may also be invoked in the block body

Encoding Blocks of System C

A block construction in System C does several things:

(1) bind a both term- and type-level capability f

(2) this f may be invoked in the block body

(3) any capability from the context may also be invoked in the block body

Encoding Blocks of System C

A block construction in System C does several things:

(1) bind a both term- and type-level capability f

(2) this f may be invoked in the block body

(3) any capability from the context may also be invoked in the block body

Our encoding uses modalities to make them explicit

$$[\![(\texttt{Int}, f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow \texttt{1}]\!] = \forall f^*.\ \langle f^* \rangle(\texttt{Int} \to [f^*](\texttt{Int} \to \texttt{1}) \to \texttt{1})$$

For (1), we introduce a type-level variable f^* and wrap the argument with the modality $[f^*]$

For (2) and (3), we use the relative modality $\langle f^* \rangle$

Encoding Boxes of System C

Boxes in System C are encoded as absolute modalities

$$[\![(f : \texttt{Int} \Rightarrow \texttt{1}) \Rightarrow (\texttt{Int} \Rightarrow \texttt{1} \textbf{ at } \{f\})]\!] = \forall f^*.\ \langle f^* \rangle ([f^*](\texttt{Int} \to \texttt{1}) \to [f^*](\texttt{Int} \to \texttt{1}))$$

Wrapping Up

Comparing Rows and Capabilities

By encoding both System F$^\epsilon$ and System C into MET, we can easily compare them

System F$^\epsilon$ to MET:
$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$
$$[\![\forall \varepsilon.\, A]\!] = \forall \varepsilon.\, [\![A]\!]$$

System C to MET:
$$[\![(\overline{A}, \overline{f : T}) \Rightarrow B]\!] = \forall \overline{f^*}.\ \langle \overline{f^*} \rangle (\overline{[\![A]\!]} \to \overline{[f^*][\![T]\!]} \to [\![B]\!])$$
$$[\![T \text{ at } C]\!] = [[\![C]\!]][\![T]\!]$$

Two main observations:

1.
2.

Comparing Rows and Capabilities

By encoding both System F^ϵ and System C into MET, we can easily compare them

System F^ϵ to MET:

$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$

$$[\![\forall \varepsilon.\ A]\!] = \forall \varepsilon.\ [\![A]\!]$$

System C to MET:

$$[\![(\overline{A}, \overline{f : T}) \Rightarrow B]\!] = \forall \overline{f^*}.\ \langle \overline{f^*} \rangle (\overline{[\![A]\!]} \to \overline{[f^*][\![T]\!]} \to [\![B]\!])$$

$$[\![T \text{ at } C]\!] = [[\![C]\!]]\,[\![T]\!]$$

Two main observations:

1. different top-level modalities
2.

Comparing Rows and Capabilities

By encoding both System F$^\epsilon$ and System C into MET, we can easily compare them

System F$^\epsilon$ to MET:

$$[\![A \to^{E} B]\!] = [\![\![E]\!]\!]([\![A]\!] \to [\![B]\!])$$

$$[\![\forall \varepsilon.\ A]\!] = \forall \varepsilon.\ [\![A]\!]$$

System C to MET:

$$[\![(\overline{A}, \overline{f : T}) \Rightarrow B]\!] = \forall \overline{f^*}.\ \langle \overline{f^*} \rangle (\overline{[\![A]\!]} \to \overline{[f^*][\![T]\!]} \to [\![B]\!])$$

$$[\![T \textbf{ at } C]\!] = [\![\![C]\!]\!]\,[\![T]\!]$$

Two main observations:

1. different top-level modalities => System F$^\epsilon$ functions fully specify effects while System C blocks may use any capabilities from the context (unless boxed)
2.

Comparing Rows and Capabilities

By encoding both System F$^\epsilon$ and System C into MET, we can easily compare them

System F$^\epsilon$ to MET: $$[\![A \to^{\boldsymbol{E}} B]\!] = [[\![\boldsymbol{E}]\!]]([\![A]\!] \to [\![B]\!])$$

$$[\![\forall \varepsilon.\, A]\!] = \forall \varepsilon.\ [\![A]\!]$$

System C to MET: $$[\![(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B]\!] = \forall \overline{\boldsymbol{f}^*}.\ \langle \overline{\boldsymbol{f}^*} \rangle (\overline{[\![A]\!]} \to \overline{[\boldsymbol{f}^*][\![T]\!]} \to [\![B]\!])$$

$$[\![T \textbf{ at } \boldsymbol{C}]\!] = [[\![\boldsymbol{C}]\!]][\![T]\!]$$

Two main observations:

1. different top-level modalities
2. different uses of effect variables

Comparing Rows and Capabilities

By encoding both System F$^\epsilon$ and System C into MET, we can easily compare them

System F$^\epsilon$ to MET:
$$[\![A \to^{E} B]\!] = [[\![E]\!]]([\![A]\!] \to [\![B]\!])$$
$$[\![\forall \varepsilon.\, A]\!] = \forall \varepsilon.\ [\![A]\!]$$

System C to MET:
$$[\![(\overline{A}, \overline{f : T}) \Rightarrow B]\!] = \forall \overline{f^*}.\ \langle \overline{f^*} \rangle (\overline{[\![A]\!]} \to \overline{[f^*][\![T]\!]} \to [\![B]\!])$$
$$[\![T \ \mathbf{at}\ C]\!] = [[\![C]\!]][\![T]\!]$$

Two main observations:

1. different top-level modalities
2. different uses of effect variables => capabilities enable some form of implicit effect polymorphism

More in the Paper

Full formalisation of the uniform framework $\textsc{Met}(\mathcal{X})$

- Parameterised by effect structures $\mathcal{X}$ following Morris and McKinna[1] and Yoshioka et al.[2]
- Extensions including local labels and modality-parameterised handlers
- Proofs of type soundness and effect safety

Encodings of different effect systems

- Koka (System F^{ϵ}, System $F^{\epsilon+\mathrm{sn}}$) and Effekt (System C, System Ξ) with effect handlers
- Proofs of type and semantics preservation

[1] Morris and McKinna, "Abstracting Extensible Data Types: Or, Rows by Any Other Name", 2019.
[2] Yoshioka, Sekiyama, and Igarashi, "Abstracting Effect Systems for Algebraic Effect Handlers", 2024.

Takeaway

Decoupling effect tracking from functions provides the flexibility and expressivity to subsume row-based and capability-based effect systems

Koka as MET: $$[\![A \rightarrow^{\boldsymbol{E}} B]\!] = [[\![\boldsymbol{E}]\!]]([\![A]\!] \rightarrow [\![B]\!])$$

Effekt as MET: $$[\![(\overline{A}, \overline{\boldsymbol{f} : T}) \Rightarrow B]\!] = \forall \overline{\boldsymbol{f}^*}.\langle \overline{\boldsymbol{f}^*} \rangle (\overline{[\![A]\!]} \rightarrow \overline{[\boldsymbol{f}^*][\![T]\!]} \rightarrow [\![B]\!])$$