CBPV + effects
CBPV + coeffects

Stephanie Weirich

joint work with Cassia Torczon, Emmanuel Suárez Acevedo, Shubh Agrawal and Joey Velez-Ginorio

March 25, 2024
Modal type distinction
T is a monad

Graded modality
\( \diamond \phi \) is a monad

Type-and-effect system grades "ambient" computational monad
Modal type distinction
! is a comonad

Graded modal type
\(\square_n\) is a comonad

Type-and-coeffect system grades "ambient" comonad
annotations in typing context
What is this talk about?

1. Extending CBPV’s type system with effect tracking
2. Extending CBPV’s type system with coeffect tracking
3. Extending CBPV’s type system with effect and coeffect tracking (1 slide)

Why CBPV?
What is this talk about?

1. Extending CBPV’s type system with effect tracking
2. Extending CBPV’s type system with coeffect tracking
3. Extending CBPV’s type system with effect *and* coeffect tracking (1 slide)

Why CBPV?

Effects and Coeffects can be tracked in types using graded monads and comonads. But this requires us to isolate effects and coeffects in dedicated structures.
What is this talk about?

1. Extending CBPV’s type system with effect tracking
2. Extending CBPV’s type system with coeffect tracking
3. Extending CBPV’s type system with effect and coeffect tracking (1 slide)

Why CBPV?

Effects and Coeffects can be tracked in types using graded monads and comonads. But this requires us to isolate effects and coeffects in dedicated structures.

But, CBPV already makes the ambient monad and comonad explicit. We just need to grade it!
What is this talk about?

1. Extending CBPV's type system with effect tracking
2. Extending CBPV's type system with coeffect tracking
3. Extending CBPV's type system with effect *and* coeffect tracking (1 slide)

Why CBPV?

Effects and Coeffects can be tracked in types using graded monads and comonads. But this requires us to isolate effects and coeffects in dedicated structures.

But, CBPV already makes the ambient monad and comonad explicit. We just need to grade it!

And, CBPV is a polarized type system: we can observe the duality between effects and coeffects, and understand their interactions with evaluation order.
We can track effects with types

\[ \Gamma \vdash_{\text{eff}} e : \phi \tau \]

An effect annotation \( \phi \) tells us what happens when \( e \) is evaluated.

For example,

- To track running time, \( \phi \) is natural number that counts executions of an effectful “tick” term.
We can track effects with types

\[ \Gamma \vdash_{\text{eff}} e : \phi \quad \tau \]

An effect annotation \( \phi \) tells us what happens when \( e \) is evaluated.

For example,

- To track running time, \( \phi \) is a natural number that counts executions of an effectful “tick” term.
- With algebraic effects, \( \phi \) is the set of operations triggered during computation.
We can track effects with types

\[ \Gamma \vdash_{\text{eff}} e : \phi \tau \]

An effect annotation \( \phi \) tells us what happens when \( e \) is evaluated.

For example,

- To track running time, \( \phi \) is natural number that counts executions of an effectful “tick” term.
- With algebraic effects, \( \phi \) is the set of operations triggered during computation.
- To precisely trace logging or other outputs, \( \phi \) is a list of strings.
We can track effects with types

\[ \Gamma \vdash_{\text{eff}} e : \phi \tau \]

\[
\begin{align*}
\text{lam-eff-unit} \\
\Gamma \vdash_{\text{eff}} () : \varepsilon \text{ unit} \\
\text{lam-eff-var} \\
\Gamma \vdash_{\text{eff}} x : \varepsilon \tau \\
\text{lam-eff-abs} \\
\Gamma, x : \tau_1 \vdash_{\text{eff}} e : \phi \tau_2 \\
\end{align*}
\]

\[
\begin{align*}
\text{lam-eff-app} \\
\Gamma \vdash_{\text{eff}} e_1 : \phi_1 \tau_1 \to \tau_2 \\
\Gamma \vdash_{\text{eff}} e_2 : \phi_2 \tau_1 \\
\Gamma \vdash_{\text{eff}} e_1 e_2 : \phi_1 \phi_2 \phi_3 \tau_2 \\
\text{lam-eff-sub} \\
\phi_1 \leq_{\text{eff}} \phi_2 \\
\Gamma \vdash_{\text{eff}} e : \phi_2 \tau \\
\text{lam-eff-tick} \\
\Gamma \vdash_{\text{eff}} \text{tick} : \varepsilon \text{ Tick unit} \\
\end{align*}
\]

To track effects throughout the computation, need a pre-ordered monoid.

[Lucassen and Gifford 1988, Katsumata 2014]
We can track effects with types

\[ \Gamma \vdash_{\text{eff}} e : \phi \tau \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lam-eff-unit</td>
<td>[ \frac{}{\Gamma \vdash_{\text{eff}} () : \varepsilon \text{unit}} ]</td>
</tr>
<tr>
<td>lam-eff-var</td>
<td>[ \frac{x : \tau \in \Gamma}{\Gamma \vdash_{\text{eff}} x : \varepsilon \tau} ]</td>
</tr>
<tr>
<td>lam-eff-abs</td>
<td>[ \frac{}{\Gamma \vdash_{\text{eff}} \lambda x. e : \varepsilon \tau_1 \rightarrow \tau_2} ]</td>
</tr>
<tr>
<td>lam-eff-app</td>
<td>[ \frac{\Gamma \vdash_{\text{eff}} e_1 : \phi_1 \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash_{\text{eff}} e_2 : \phi_2 \tau_1}{\Gamma \vdash_{\text{eff}} \ e_1 e_2 : \phi_1 \phi_2 \phi_3 \ varepsilon \tau_2} ]</td>
</tr>
<tr>
<td>lam-eff-sub</td>
<td>[ \frac{\phi_1 \leq_{\text{eff}} \phi_2}{\Gamma \vdash_{\text{eff}} e : \phi_2 \tau} ]</td>
</tr>
<tr>
<td>lam-eff-tick</td>
<td>[ \frac{}{\Gamma \vdash_{\text{eff}} \text{tick} : \text{Tick} \text{unit}} ]</td>
</tr>
</tbody>
</table>

To track effects throughout the computation, need a **pre-ordered monoid**.

These rules are specific to a **call-by-value** semantics.

[Lucassen and Gifford 1988, Katsumata 2014]
We can track effects with types

\[ \Gamma \vdash_{eff} e : \phi \tau \]

To track effects throughout the computation, need a pre-ordered monoid.

These rules are specific to a call-by-value semantics.

If we had a call-by-name semantics, we would need different rules. (And different types!)

[Lucassen and Gifford 1988, Katsumata 2014]
CBPV

CBPV is designed to model effects and subsume both CBV and CBN evaluation.

\[ \Gamma \vdash V : A \quad \Gamma \vdash M : B \]

CBPV is polarized: separate positive and negative types.

(value type) \quad A ::= \text{unit} \mid U B \\
(value) \quad V ::= x \mid () \mid \{ M \}

(computation type) \quad B ::= A \to B \mid F A \\
(computation) \quad M ::= \lambda x.M \mid M V \mid V! \\
\quad \quad \text{ | return } V \mid x \leftarrow M \text{ in } N

The type constructors \( U \) and \( F \) form an \textit{adjunction} between values and computations.

- \( U F A \) is a monad
- \( F U B \) is a comonad
Let's extend the CBPV type system to track effects.

\[
\begin{align*}
\Gamma & \vdash_{\text{eff}} V : A \\
\Gamma & \vdash_{\text{eff}} M : \phi \ B
\end{align*}
\]

We'll record latent effects in the thunk type as \( U_\phi B \).

\[
\begin{align*}
\text{(value type)} & \quad A \ ::= \ \text{unit} \mid U_\phi B \\
\text{(value)} & \quad V \ ::= \ x \mid \{M\}
\end{align*}
\]

\[
\begin{align*}
\text{(computation type)} & \quad B \ ::= \ A \to B \mid FA \\
\text{(computation)} & \quad M \ ::= \ \lambda x. M \mid MV \mid V! \\
& \quad \text{or} \quad \text{return} \ V \mid x \leftarrow M \text{ in } N
\end{align*}
\]
CBPV + effect tracking

Let’s extend the CBPV type system to track effects.

$$\Gamma \vdash_{\text{eff}} V : A$$  
$$\Gamma \vdash_{\text{eff}} M : \phi \ B$$

We’ll record latent effects in the thunk type as $U_{\phi} B$.

\begin{align*}
(value \ type) & \quad A & := & \text{unit} | U_{\phi} B \\
(value) & \quad V & := & x | \{M\}
\end{align*}

\begin{align*}
(computation \ type) & \quad B & := & A \rightarrow B | FA \\
(computation) & \quad M & := & \lambda x. M | MV | V! \\
& & & | \quad \text{return} \ V | x \leftarrow M \text{ in } N | \text{tick}
\end{align*}

and add example effect: \textbf{tick}.

\begin{align*}
\text{eff-tick} \\
\Gamma \vdash_{\text{eff}} \textbf{tick} : \text{.Tick} \ F \text{unit}
\end{align*}
CBPV with effect tracking

\[ \Gamma \vdash_{\text{eff}} V : A \]

\begin{align*}
\text{eff-var} & \quad x : A \in \Gamma \\
\Gamma & \vdash_{\text{eff}} x : A
\end{align*}

\begin{align*}
\text{eff-thunk} & \quad \Gamma \vdash_{\text{eff}} M : \phi \ B \\
\Gamma & \vdash_{\text{eff}} \{ M \} : U_\phi B
\end{align*}

\begin{align*}
\text{eff-unit} & \quad \Gamma \vdash_{\text{eff}} () : \text{unit}
\end{align*}

\[ \Gamma \vdash_{\text{eff}} M : \phi \ B \]

\begin{align*}
\text{eff-abs} & \quad \Gamma, x : A \vdash_{\text{eff}} M : \phi \ B \\
\Gamma & \vdash_{\text{eff}} \lambda x. M : \phi \ A \rightarrow B
\end{align*}

\begin{align*}
\text{eff-app} & \quad \Gamma \vdash_{\text{eff}} M : \phi \ A \rightarrow B \\
\Gamma & \vdash_{\text{eff}} V : A \\
\Gamma & \vdash_{\text{eff}} MV : \phi \ B
\end{align*}

\begin{align*}
\text{eff-force} & \quad \Gamma \vdash_{\text{eff}} V : U_\phi B \\
\Gamma & \vdash_{\text{eff}} V! : \phi \ B
\end{align*}

\begin{align*}
\text{eff-ret} & \quad \Gamma \vdash_{\text{eff}} V : A \\
\Gamma & \vdash_{\text{eff}} \text{return} V : \varepsilon \ FA
\end{align*}

\begin{align*}
\text{eff-letin} & \quad \Gamma, x : A \vdash_{\text{eff}} N : \phi_2 B \\
\Gamma & \vdash_{\text{eff}} x \leftarrow M \text{ in } N : \phi_1 \cdot \phi_2 B
\end{align*}

\begin{align*}
\text{eff-sub} & \quad \Gamma \vdash_{\text{eff}} M : \phi_1 B \\
\phi_1 & \leq_{\text{eff}} \phi_2 \\
\Gamma & \vdash_{\text{eff}} M : \phi_2 B
\end{align*}

[Kammar and Plotkin 2012, Kammar, Lindley, Oury 2013]
Effect soundness

Key result of type system is *effect soundness*: the type system bounds effects that could occur at runtime.

Big-step operational semantics: \( \rho \vdash_{\text{eff}} M \downarrow T \# \phi \) counts ticks while evaluating computation \( M \) to terminal \( T \).

**Theorem**

If \( \emptyset \vdash_{\text{eff}} M : \phi \text{ F A and } \emptyset \vdash_{\text{eff}} M \downarrow \text{return } W \# \phi' \text{ then } \phi' \leq_{\text{eff}} \phi \).

**Proof.**

Uses logical relations.
What about coeffects?

Coeffects track how input values contribute to the output result.

- Bounded linear types
- Whether functions use their arguments
- Differential privacy (how sensitive are function outputs to their inputs)
- Whether functions are monotonic
- Information-flow
- ...

(Technically, these are examples of *structured* coeffects.)
What about coeffects?

Coeffects track how input values contribute to the output result.

- Bounded linear types
- Whether functions use their arguments
- Differential privacy (how sensitive are function outputs to their inputs)
- Whether functions are monotonic
- Information-flow
- ...

(Technically, these are examples of structured coeffects.)

We mark variables in the context with coeffects $q$ (short for quantity).
Coeffect examples

• For bounded linear types, we can use natural numbers.

\[ x : ^1 \text{int}, y : ^3 \text{int}, z : ^0 \text{int} \vdash_{\text{coeff}} x + (y + y) : \text{int} \]
Coeffect examples

- For *bounded linear types*, we can use natural numbers.

\[ x : 1 \text{ int}, y : 3 \text{ int}, z : 0 \text{ int} \vdash_{\text{coeff}} x + (y + y) : \text{int} \]

- For *relevance analysis*, 0 marks arguments that are not used and \( \omega \) marks arguments that *may* be used.

\[ x : \omega \text{ int}, y : \omega \text{ int}, z : 0 \text{ int} \vdash_{\text{coeff}} x + (y + y) : \text{int} \]
Coeffect examples

• For *bounded linear types*, we can use natural numbers.

\[
x : ^1 \text{int}, y : ^3 \text{int}, z : ^0 \text{int} \vdash \text{coeff } x + (y + y) : \text{int}
\]

• For *relevance analysis*, 0 marks arguments that are not used and \( \omega \) marks arguments that *may* be used.

\[
x : ^\omega \text{int}, y : ^\omega \text{int}, z : ^0 \text{int} \vdash \text{coeff } x + (y + y) : \text{int}
\]

• For *data flow caching*, we want to provide access to prior values during streaming computation.

\[
x : ^1 \text{int}, y : ^0 \text{int} \vdash \text{coeff } (\text{prev } x) + x + y : \text{int}
\]

We can use natural numbers that track how many previous values are required.
We can track coeffects with types

Context comes with a list of coeffects for every variable.

\[ \gamma ::= \emptyset \mid \gamma, \varphi \]

We use notation to extend both at once:

\[ \gamma \cdot \Gamma, x : \varphi \tau = (\gamma, \varphi) \cdot (\Gamma, x : \tau) \]

This is for a call-by-name language [Abel and Bernardy 2020, Choudhury et al. 2021].
We can track coeffects with types

Context comes with a list of coeffects for every variable.

\[ \gamma ::= \emptyset \mid \gamma \cdot q \]

We use notation to extend both at once:

\[ \gamma \cdot \Gamma \cdot x : q \cdot \tau = (\gamma \cdot q) \cdot (\Gamma \cdot x : \tau) \]

\[ \gamma \cdot \Gamma \vdash_{\text{coeff}} e : \tau \]

**lam-coeff-var**

\[ \overline{0} \cdot \Gamma_1 \cdot x : 1 \cdot \tau, \overline{0} \cdot \Gamma_2 \vdash_{\text{coeff}} x : \tau \]

**lam-coeff-abs**

\[
\gamma \cdot \Gamma, (x : q \cdot \tau_1) \vdash_{\text{coeff}} e : \tau_2
\]

\[ \gamma \cdot \Gamma \vdash_{\text{coeff}} \lambda^q x. e : \tau_1 \rightarrow \tau_2 \]

**lam-coeff-appv**

\[
\gamma_1 \cdot \Gamma \vdash_{\text{coeff}} e_1 : \tau_1^q \rightarrow \tau_2
\]

\[
\gamma_2 \cdot \Gamma \vdash_{\text{coeff}} e_2 : \tau_1 \quad \gamma \equiv \gamma_1 + (q \land 1 \cdot \gamma_2)
\]

\[ \gamma \cdot \Gamma \vdash_{\text{coeff}} e_1 \cdot e_2 : \tau_2 \]

**lam-coeff-sub**

\[ \gamma_1 \cdot \Gamma \vdash_{\text{coeff}} e : \tau \]

\[ \gamma_2 \leq_{\text{co}} \gamma_1 \]

\[ \gamma_2 \cdot \Gamma \vdash_{\text{coeff}} e : \tau \]

Call-by-value language forces usage in application rule [Gavazzo 2018].
CBPV with coefffects

\[ \gamma \cdot \Gamma \vdash_{\text{coeff}} V : A \] (Value typing)

\[ \begin{aligned} &\text{coeff-var} \\ &0 \cdot \Gamma_1, x : A \vdash_{\text{coeff}} x : A \end{aligned} \]

\[ \gamma \cdot \Gamma \vdash_{\text{coeff}} \mathbf{0} : \text{unit} \]

\[ \gamma \cdot \Gamma \vdash_{\text{coeff}} \mathbf{\{M\}} : \text{U} \text{B} \] (Computation typing)

\[ \begin{aligned} &\text{coeff-unit} \\ &0 \cdot \Gamma \vdash_{\text{coeff}} () : \text{unit} \end{aligned} \]

\[ \begin{aligned} &\text{coeff-app} \\ &\gamma_1 \cdot \Gamma \vdash_{\text{coeff}} M : A^q \rightarrow B \\ &\gamma_2 \cdot \Gamma \vdash_{\text{coeff}} V : A \\ &\gamma \equiv \gamma_1 + (q \cdot \gamma_2) \\ &\gamma \cdot \Gamma \vdash_{\text{coeff}} MV : B \end{aligned} \]

\[ \begin{aligned} &\text{coeff-force} \\ &\gamma \cdot \Gamma \vdash_{\text{coeff}} V! : \text{B} \end{aligned} \]

\[ \begin{aligned} &\text{coeff-abs} \\ &\gamma \cdot \Gamma, x : A \vdash_{\text{coeff}} M : B \\ &\gamma \cdot \Gamma \vdash_{\text{coeff}} \lambda x^q . M : A^q \rightarrow B \end{aligned} \]

\[ \begin{aligned} &\text{coeff-letin-v} \\ &\gamma \cdot \Gamma \vdash_{\text{coeff}} V : \text{F}_{q_1} A \\ &\gamma_2 \cdot \Gamma, x : A \vdash_{\text{coeff}} N : B \\ &\gamma \equiv (q_2 \cdot \gamma_1) + \gamma_2 \quad q_2 = q_2 \wedge 1 \\ &\gamma \cdot \Gamma \vdash_{\text{coeff}} x \leftarrow^{q_2} M \text{ in } N : B \end{aligned} \] (+subrules)
Coeffect soundness

To show coeffect soundness, we define an environment-based operational semantics that counts uses of variables.

\[ \gamma \cdot \rho \vdash_{\text{coeff}} V \Downarrow W \]  

\underline{Value rules}

\begin{align*}
\text{eval-coeff-val-var} & \quad \bar{0}_1 \cdot \rho_1, x \mapsto^1 W, \bar{0}_2 \cdot \rho_2 \vdash_{\text{coeff}} x \Downarrow W \\
\text{eval-coeff-val-unit} & \quad \bar{0} \cdot \rho \vdash_{\text{coeff}} () \Downarrow () \\
\text{eval-coeff-val-thunk} & \quad \gamma \cdot \rho \vdash_{\text{coeff}} \{M\} \Downarrow \text{clo}(\gamma \cdot \rho, \{M\}) \\
\text{eval-coeff-val-vsub} & \quad \gamma_1 \cdot \rho \vdash_{\text{coeff}} V \Downarrow W \\
 & \quad \gamma_2 \preceq \text{co} \gamma_1 \\
 & \quad \gamma_2 \cdot \rho \vdash_{\text{coeff}} V \Downarrow W
\end{align*}

Lemma (Coeffect soundness)

1. If \( \gamma \cdot \Gamma \vdash_{\text{coeff}} V : A \) then \( \gamma \cdot \rho \vdash_{\text{coeff}} V \Downarrow W \).
2. If \( \gamma \cdot \Gamma \vdash_{\text{coeff}} M : B \) then \( \gamma \cdot \rho \vdash_{\text{coeff}} M \Downarrow T \).
A strange semantics?

Although sound, this semantics doesn’t model resource usage.

\[
\gamma \cdot \rho \vdash_{\text{coeff}} M \downarrow T
\]

(Computation rules)

\[
eval\text{-coeff-comp-app-abs} \\
\begin{align*}
\gamma_1 \cdot \rho \vdash_{\text{coeff}} M \downarrow \text{clo}(\gamma' \cdot \rho', \lambda x^q. M') \\
\gamma_2 \cdot \rho \vdash_{\text{coeff}} V \downarrow W \\
\gamma' \cdot \rho', x \mapsto^q W \vdash_{\text{coeff}} M' \downarrow T \\
\gamma & \equiv \gamma_1 + q \cdot \gamma_2
\end{align*}
\]

\[
\gamma \cdot \rho \vdash_{\text{coeff}} M V \downarrow T
\]

Application rule “invents” resources when \( q \) is zero!
A strange semantics?

Although sound, this semantics doesn’t model \textit{resource usage}.

\[
\gamma \cdot \rho \vdash_{\text{coeff}} M \downarrow T
\]

(Computation rules)

\[
\begin{align*}
\text{eval-coeff-comp-app-abs} & \quad \gamma_1 \cdot \rho \vdash_{\text{coeff}} M \downarrow \text{clo}(\gamma' \cdot \rho', \lambda x^q. M') \\
& \quad \gamma_2 \cdot \rho \vdash_{\text{coeff}} V \downarrow W \\
& \quad \gamma' \cdot \rho', x \mapsto^q W \vdash_{\text{coeff}} M' \downarrow T \\
& \quad \gamma \equiv \gamma_1 + q \cdot \gamma_2 \\
\text{eval-coeff-comp-abs} & \quad \gamma \cdot \rho \vdash_{\text{coeff}} \lambda x^q.M \downarrow \text{clo}(\gamma \cdot \rho, \lambda x^q. M) \\
\end{align*}
\]

Application rule “invents” resources when \( q \) is zero!

We can type this judgement, which says that \( x \) does not contribute to the final result.

\[
\begin{align*}
x :^0 A \vdash_{\text{coeff}} (\lambda y^0. \text{return } ()) \ x : \text{F unit}
\end{align*}
\]
Resource accounting semantics

Can discard unused values, without accounting for their resources

\[\gamma \cdot \rho \vdash_{\text{coeff}} M \Downarrow T\]

(Computation rules)

eval-lin-comp-app-abs

\[\gamma_1 \cdot \rho \vdash_{\text{lin}} M \Downarrow \text{clo}(\gamma' \cdot \rho', \lambda x^q . M')\]

\[\gamma_2 \cdot \rho \vdash_{\text{lin}} V \Downarrow W\]

\[(\gamma' \cdot \rho') , (x \mapsto^q W) \vdash_{\text{lin}} M' \Downarrow T\]

\[\gamma \equiv \gamma_1 + q \cdot \gamma_2\]

\[q \neq 0\]

\[\gamma \cdot \rho \vdash_{\text{lin}} M V \Downarrow T\]

eval-lin-comp-app-abs-zero

\[\gamma \cdot \rho \vdash_{\text{lin}} M \Downarrow \text{clo}(\gamma' \cdot \rho', \lambda x^0 . M')\]

\[(\gamma' \cdot \rho') , (x \mapsto^0 \bot) \vdash_{\text{lin}} M' \Downarrow T\]

\[\gamma \cdot \rho \vdash_{\text{lin}} M V \Downarrow T\]

eval-lin-comp-return

\[\gamma' \cdot \rho \vdash_{\text{lin}} V \Downarrow W\]

\[\gamma \equiv q \cdot \gamma'\]

\[q \neq 0\]

\[\gamma \cdot \rho \vdash_{\text{lin}} \text{return}_q V \Downarrow \text{return}_q W\]

eval-lin-comp-ret-zero

\[\bar{0} \cdot \rho \vdash_{\text{lin}} \text{return}_0 V \Downarrow \text{return}_0 \bot\]
Cannot discard effectful computations

\[ \gamma \cdot \rho \vdash_{\text{coeff}} M \downarrow T \]

(Computation rules)

eval-lin-comp-letin-ret

\[ \gamma_1 \cdot \rho \vdash_{\text{lin}} M \downarrow \text{return}_{q_1} W \]

\[ \gamma_2 \cdot \rho, \ x \leftarrow^{q_1 \cdot q_2'} W \vdash_{\text{lin}} N \downarrow T \]

\[ \gamma \equiv q_2' \cdot \gamma_1 + \gamma_2 \]

\[ q_2' = q_2 \land 1 \]

\[ \gamma \cdot \rho \vdash_{\text{lin}} x \leftarrow^{q_2} M \text{ in } N \downarrow T \]
Combined effects and co-effects

Can discard computations that are pure.

Let's track effects and co-effects together.

\[ \gamma \cdot \Gamma \vdash_{\text{full}} M : \phi B \]  

(Typing rule)

full-letin-zero

\[ \gamma_1 \cdot \Gamma \vdash_{\text{full}} M_1 : \varepsilon F q_1 A \]
\[ \gamma_2 \cdot \Gamma , x : 0 A \vdash_{\text{full}} M_2 : \phi B \]

\[ \gamma_2 \cdot \Gamma \vdash_{\text{full}} x \leftarrow 0 M_1 \text{ in } M_2 : \phi B \]

(Evaluation rule)

eval-full-comp-letin-zero

\[ \gamma \cdot \rho \vdash_{\text{full}} M \downarrow T \# \phi \]

\[ \gamma \cdot \rho , x \mapsto q_1 q_2' \not\vdash_{\text{full}} N \downarrow T \# \phi \]

\[ \gamma \cdot \rho \vdash_{\text{full}} x \leftarrow 0 M \text{ in } N \downarrow T \# \phi \]
Summary

- Augmented CBPV with effect and coeffect tracking.
- Effects describe computations, so annotate thunk type $U_\phi B$.
  Coeffects describe values, so annotate returner type $F_q A$
- $U_\phi F A$ is a graded monad in the value language.
  $F_q U B$ is a graded comonad in the computation language.
- Showed effect and coeffect soundness, even in the presence of a semantics that tracks resource usage.
- In the paper: Standard CBV and CBN translations are type, effect, coeffect preserving.
  Explains restrictions found in some CBV coeffect type systems. (CBN translation does not require the use of "letin").
- Proofs mechanized in Coq.
We can translate type-and-effect CBV to effect-tracking CBPV. The standard translation just works.

\[
\begin{align*}
  [\text{unit}]_v &= \text{unit} \\
  [\tau_1 \to \tau_2]_v &= U_\phi ([\tau_1]_v \to F [\tau_2]_v)
\end{align*}
\]

\[
\begin{align*}
  [()]_v &= \text{return } () \\
  [x]_v &= \text{return } x \\
  [\lambda x.e]_v &= \text{return } \{ \lambda x. [e]_v \} \\
  [e_1 e_2]_v &= x \leftarrow [e_1]_v \text{ in } y \leftarrow [e_2]_v \text{ in } x!y \\
  [\text{tick}]_v &= \text{tick}
\end{align*}
\]

Theorem (Translation preserves types-and-effects)

If \( \Gamma \vdash_{\text{eff}} e : \phi \tau \) then \( [\Gamma]_v \vdash_{\text{eff}} [e]_v : \phi F [\tau]_v \).
We can also use the CBN translation for a source language with graded monads.

However, while $\mathbf{U FA}$ is a monad in CBPV, it is awkward to access.

**Theorem (Translation preserves types)**

If $\Gamma \vdash_{\text{mon}} e : \tau$ then $[\Gamma]_n \vdash_{\text{eff}} [e]_n :^e [\tau]_n$. 
CBN translation (Graded Monads!)

We can also use the CBN translation for a source language with graded monads. However, while $UFA$ is a monad in CBPV, it is awkward to access.

$$\begin{align*}
[T_\phi \tau]_n &= F U_\phi F U_\epsilon [\tau]_n
\end{align*}$$

Theorem (Translation preserves types)

If $\Gamma \vdash_{\text{mon}} e : \tau$ then $[\Gamma]_n \vdash_{\text{eff}} [e]_n :^\epsilon [\tau]_n$. 
CBN translation (Graded Monads!)

We can also use the CBN translation for a source language with graded monads. However, while $UFA$ is a monad in CBPV, it is awkward to access.

\[
[T_\phi \tau]_n = F U_\phi F U_\varepsilon [\tau]_n
\]

\[
[\text{return } e]_n = \text{return} \{\text{return} \{[e]_n\}\}
\]

Theorem (Translation preserves types)

If $\Gamma \vdash_{\text{mon}} e : \tau$ then $[\Gamma]_n \vdash_{\text{eff}} [e]_n :^\varepsilon [\tau]_n$. 
CBN translation (Graded Monads!)

We can also use the CBN translation for a source language with graded monads.

However, while $\textbf{U} \textbf{F} \textbf{A}$ is a monad in CBPV, it is awkward to access.

\[
[T_\phi \tau]_n = \textbf{FU}_\phi \textbf{FU}_\varepsilon [\tau]_n
\]

\[
[\text{return } e]_n = \text{return} \{ \text{return} \{ [e]_n \} \}
\]

\[
[\text{bind } x = e_1 \text{ in } e_2]_n = \text{return} \{ y \leftarrow [e_1]_n \text{ in } x \leftarrow y! \text{ in } z \leftarrow [e_2]_n \text{ in } z! \}
\]

Theorem (Translation preserves types)

\[\text{If } \Gamma \vdash_{\text{mon}} e : \tau \text{ then } [\Gamma]_n \vdash_{\text{eff}} [e]_n :^\varepsilon [\tau]_n.\]
CBN translation (Graded Monads!)

We can also use the CBN translation for a source language with graded monads. However, while $UFA$ is a monad in CBPV, it is awkward to access.

$$\text{[T}_\phi \tau \text{]}_n = FU_\phi FU_\varepsilon \text{[}\tau\text{]}_n$$

$$\text{[return } e\text{]}_n = \text{return } \{ \text{return } \{ [e]_n \} \}$$

$$\text{[bind } x = e_1 \text{ in } e_2\text{]}_n = \text{return } \{ y \leftarrow [e_1]_n \text{ in } x \leftarrow y! \text{ in } z \leftarrow [e_2]_n \text{ in } z! \}$$

$$\text{[tick]\]}_n = \text{return } \{ x \leftarrow \text{tick in return } \{ \text{return } x \} \}$$

Theorem (Translation preserves types)

If $\Gamma \vdash_{mon} e : \tau$ then $[\Gamma]_n \vdash_{eff} [e]_n :^{\varepsilon} [\tau]_n$. 
CBN translation (coeffects!)

Standard translation of CBN to CBPV just works.

\[
\begin{align*}
\text{[unit]}_n &= F_1 \text{ unit} \\
\text{[} \tau_1^q \to \tau_2 \text{]}_n &= (U \text{ [} \tau_1 \text{]}_n)^q \to [\tau_2]_n \\
\text{[} \Gamma, x : \tau \text{]}_n &= [\Gamma]_n, x : U [\tau]_n \\
\text{[()}_n &= \text{return}_1 () \\
\text{[} x \text{]}_n &= x! \\
\text{[} \lambda x.e \text{]}_n &= \lambda x. [e]_n \\
\text{[} e_1 e_2 \text{]}_n &= [e_1]_n \{[e_2]_n\}
\end{align*}
\]

Theorem (Translation preserves types and coeffects)

If \( \gamma \cdot \Gamma \vdash_{\text{coeff}} e : \tau \) then \( \gamma \cdot [\Gamma]_n \vdash_{\text{coeff}} [e]_n : [\tau]_n \)}. 

Interlude: Two kinds of products

CBPV has two forms of products: pairs of values and pairs of computations. The former are eliminated with pattern matching and the latter by projection.

Linear logic has two forms of conjunction: \textit{additive} $\&$ (aka with) and \textit{multiplicative} products $\otimes$ (aka tensor).

The former shares resources during construction, the latter does not.

\[
\begin{align*}
\text{coeff-pair} & \\
\gamma_1 \cdot \Gamma \vdash \text{coeff } V_1 : A_1 \\
\gamma_2 \cdot \Gamma \vdash \text{coeff } V_2 : A_2 \\
\hline
\gamma_1 + \gamma_2 \cdot \Gamma \vdash \text{coeff } (V_1, V_2) : A_1 \times A_2 \\
\end{align*}
\]

\[
\begin{align*}
\text{coeff-split} & \\
\gamma_1 \cdot \Gamma \vdash \text{coeff } V : A_1 \times A_2 \\
\gamma_2 \cdot \Gamma, x_1 : q A_1, x_2 : q A_2 \vdash \text{coeff } N : B \\
\gamma \equiv (q \cdot \gamma_1) + \gamma_2 \\
\gamma \cdot \Gamma \vdash \text{coeff } \text{case}_q V \text{ of } (x_1, x_2) \to N : B
\end{align*}
\]

\[
\begin{align*}
\text{coeff-cpair} & \\
\gamma \cdot \Gamma \vdash \text{coeff } M_1 : B_1 \\
\gamma \cdot \Gamma \vdash \text{coeff } M_2 : B_2 \\
\hline
\gamma \cdot \Gamma \vdash \text{coeff } \langle M_1, M_2 \rangle : B_1 \& B_2 \\
\end{align*}
\]

\[
\begin{align*}
\text{coeff-fst} & \\
\gamma \cdot \Gamma \vdash \text{coeff } M : B_1 \& B_2 \\
\hline
\gamma \cdot \Gamma \vdash \text{coeff } M.1 : B_1
\end{align*}
\]
Interlude: Four kinds of products

But it doesn’t have to be this way.
Can have “with” products in the value language, eliminated by projection.

\[
\begin{align*}
\text{coeff-vwith} & \quad \gamma \cdot \Gamma \vdash \text{coeff } V_1 : A_1 \\
& \quad \gamma \cdot \Gamma \vdash \text{coeff } V_2 : A_2 \\
& \quad \gamma \cdot \Gamma \vdash \text{coeff } \langle V_1, V_2 \rangle : A_1 \& A_2
\end{align*}
\]

Can have tensor products in the computation language, eliminated by pattern matching.

\[
\begin{align*}
\text{coeff-ctensor} & \quad \gamma_1 \cdot \Gamma \vdash \text{coeff } M_1 : B_1 \\
& \quad \gamma_2 \cdot \Gamma \vdash \text{coeff } M_2 : B_2 \\
& \quad \gamma_1 + \gamma_2 \cdot \Gamma \vdash \text{coeff } (M_1, M_2) : B_1 \times B_2
\end{align*}
\]

\[
\begin{align*}
\text{coeff-csplit} & \quad \gamma_1 \cdot \Gamma \vdash \text{coeff } M : B_1 \times B_2 \\
& \quad \gamma_2 \cdot \Gamma, x_1 : q \ U B_1, x_2 : q \ U B_2 \vdash \text{coeff } N : B \\
& \quad \gamma \equiv q \cdot \gamma_1 + \gamma_2 \\
& \quad \gamma \cdot \Gamma \vdash \text{coeff } \text{case}_q M \text{ of } (x_1, x_2) \rightarrow N : B
\end{align*}
\]