

A fibrational view on computational effects

Danel Ahman

Prosecco Team, Inria Paris

Copenhagen, 13 November 2017

Outline

We investigate the combination of

- dependent types $(\Pi, \Sigma, V =_A W, \dots)$
- computational effects (state, I/O, probability, recursion, ...)

Two guiding problems

- effectful programs in types (e.g., read and write in types)
- types of effectful programs (e.g., of sequential composition)

Our goals

- tell a mathematically natural story
- use established math. techniques
- cover a wide range of comp. effects
- discover smth. interesting

Outline

We investigate the combination of

- dependent types $(\Pi, \Sigma, V =_A W, \dots)$
- computational effects (state, I/O, probability, recursion, ...)

Two guiding problems

- effectful programs in types (e.g., read and write in types)
- types of effectful programs (e.g., of sequential composition)

Our goals

- tell a mathematically natural story (via a clean core language)
- use established math. techniques
- cover a wide range of comp. effects
- discover smth. interesting

Outline

We investigate the combination of

- dependent types $(\Pi, \Sigma, V =_A W, \dots)$
- computational effects (state, I/O, probability, recursion, ...)

Two guiding problems

- effectful programs in types (e.g., read and write in types)
- types of effectful programs (e.g., of sequential composition)

Our goals

- tell a mathematically natural story (via a clean core language)
- use established math. techniques (fibrations and adjunctions)
- cover a wide range of comp. effects
- discover smth. interesting

Outline

We investigate the combination of

- dependent types $(\Pi, \Sigma, V =_A W, \dots)$
- computational effects (state, I/O, probability, recursion, ...)

Two guiding problems

- effectful programs in types (e.g., read and write in types)
- types of effectful programs (e.g., of sequential composition)

Our goals

- tell a mathematically natural story (via a clean core language)
- use established math. techniques (fibrations and adjunctions)
- cover a wide range of comp. effects (alg. effects, continuations)
- discover smth. interesting

Outline

We investigate the combination of

- dependent types $(\Pi, \Sigma, V =_A W, \dots)$
- computational effects (state, I/O, probability, recursion, ...)

Two guiding problems

- effectful programs in types (e.g., read and write in types)
- types of effectful programs (e.g., of sequential composition)

Our goals

- tell a mathematically natural story (via a clean core language)
- use established math. techniques (fibrations and adjunctions)
- cover a wide range of comp. effects (alg. effects, continuations)
- discover smth. interesting (using handlers to reason about effects)

Effectful programs in types

(type-dependency in the presence of effects)

Effectful programs in types

Let's assume that we have some **dependent type** A , e.g.:

$$\ell : (\text{List Chr}) \vdash A(\ell) \quad \stackrel{\text{def}}{=} \quad \Sigma \ell' : (\text{List Chr}). (\text{length } \ell =_{\text{Nat}} \text{length } \ell' \times \dots)$$

which could be used to type the **dependent function**

$$\text{sort} : \Pi \ell : (\text{List Chr}). A(\ell)$$

Q: Should we allow $A[\text{receive}(y.M)/\ell]$?

- i.e., should we be allowed to type $\text{receive}(y.M) : \text{List Chr}$

A1: In this work we say **no**

- types should only depend on **static** information about effects
- we recover dependency on effectful computations via **thunks**

A2: We are also looking into the **yes** case

- type-dependency needs to be "homomorphic" (more on this later)

Effectful programs in types

Let's assume that we have some **dependent type** A , e.g.:

$$\ell : (\text{List Chr}) \vdash A(\ell) \quad \stackrel{\text{def}}{=} \quad \Sigma \ell' : (\text{List Chr}). (\text{length } \ell =_{\text{Nat}} \text{length } \ell' \times \dots)$$

which could be used to type the **dependent function**

$$\text{sort} : \Pi \ell : (\text{List Chr}). A(\ell)$$

Q: Should we allow $A[\text{receive}(y.M)/\ell]$?

- i.e., should we be allowed to type $\text{receive}(y.M) : \text{List Chr}$

A1: In this work we say **no**

- types should only depend on **static** information about effects
- we recover dependency on effectful computations via **thunks**

A2: We are also looking into the **yes** case

- type-dependency needs to be "homomorphic" (more on this later)

Effectful programs in types

Let's assume that we have some **dependent type** A , e.g.:

$$\ell : (\text{List Chr}) \vdash A(\ell) \quad \stackrel{\text{def}}{=} \quad \Sigma \ell' : (\text{List Chr}). (\text{length } \ell =_{\text{Nat}} \text{length } \ell' \times \dots)$$

which could be used to type the **dependent function**

$$\text{sort} : \Pi \ell : (\text{List Chr}). A(\ell)$$

Q: Should we allow $A[\text{receive}(y.M)/\ell]$?

- i.e., should we be allowed to type $\text{receive}(y.M) : \text{List Chr}$

A1: In this work we say **no**

- types should only depend on **static information** about effects
- we recover dependency on effectful computations via **thunks**

A2: We are also looking into the **yes** case

- type-dependency needs to be "homomorphic" (more on this later)

Effectful programs in types

Let's assume that we have some **dependent type** A , e.g.:

$$\ell : (\text{List Chr}) \vdash A(\ell) \quad \stackrel{\text{def}}{=} \quad \Sigma \ell' : (\text{List Chr}). (\text{length } \ell =_{\text{Nat}} \text{length } \ell' \times \dots)$$

which could be used to type the **dependent function**

$$\text{sort} : \Pi \ell : (\text{List Chr}). A(\ell)$$

Q: Should we allow $A[\text{receive}(y.M)/\ell]$?

- i.e., should we be allowed to type $\text{receive}(y.M) : \text{List Chr}$

A1: In this work we say **no**

- types should only depend on **static information** about effects
- we recover dependency on effectful computations via **thunks**

A2: We are also looking into the **yes** case

- type-dependency needs to be **“homomorphic”** (more on this later)

Effectful programs in types

Aim: Types should only depend on static info about effects

Solution: CBPV/EEC style distinction between vals. and comps.

- value types $\Gamma \vdash A$ (MLTT + thunks + ...)
- computation types $\Gamma \vdash \underline{C}$ (dep. CBPV/EEC)
- where Γ contains only value variables $x_1 : A_1, \dots, x_n : A_n$

Note: Could have also considered λ_{ML} and FGCBV

- building on CBPV/EEC gives a more general story
- especially for the treatment of sequential composition
- and also for integrating dependent- and effect-typing

Effectful programs in types

Aim: Types should only depend on static info about effects

Solution: CBPV/EEC style distinction between vals. and comps.

- value types $\Gamma \vdash A$ (MLTT + thunks + ...)
- computation types $\Gamma \vdash \underline{C}$ (dep. CBPV/EEC)
- where Γ contains only value variables $x_1:A_1, \dots, x_n:A_n$

Note: Could have also considered λ_{ML} and FGCBV

- building on CBPV/EEC gives a more general story
- especially for the treatment of sequential composition
- and also for integrating dependent- and effect-typing

Effectful programs in types

Aim: Types should only depend on static info about effects

Solution: CBPV/EEC style distinction between vals. and comps.

- value types $\Gamma \vdash A$ (MLTT + thunks + ...)
- computation types $\Gamma \vdash \underline{C}$ (dep. CBPV/EEC)
- where Γ contains only value variables $x_1 : A_1, \dots, x_n : A_n$

Note: Could have also considered λ_{ML} and FGCBV

- building on CBPV/EEC gives a more general story
- especially for the treatment of sequential composition
- and also for integrating dependent- and effect-typing

Assigning types to effectful programs

(e.g., sequential composition)

Assigning types to effectful programs

The problem: The standard typing rule for seq. composition

$$\frac{\Gamma \vdash M : F A \quad \Gamma, x:A \vdash N : \underline{C}}{\Gamma \vdash M \text{ to } x:A \text{ in } N : \underline{C}}$$

is not correct any more because x can appear free in the type

C

in the conclusion

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 1: We could restrict the free variables in \underline{C} : [Levy'04]

$$\frac{\Gamma \models M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, x:A \models N : \underline{C}}{\Gamma \models M \text{ to } x:A \text{ in } N : \underline{C}}$$

But sometimes it is useful if \underline{C} can depend on x !

- if we consider

`fopen (return true, return false) to x:Bool in N`

- then it would be natural to let \underline{C} depend on x , e.g.,

$x:\text{Bool} \vdash \underline{C}(x) \stackrel{\text{def}}{=} \text{if } x \text{ then "allow fread, fwrite, and fclose"} \\ \text{else "allow fopen"}$

(needs more expressive comp. types than we consider here)

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 1: We could restrict the free variables in \underline{C} : [Levy'04]

$$\frac{\Gamma \Vdash M : FA \quad \Gamma \vdash \underline{C} \quad \Gamma, x:A \Vdash N : \underline{C}}{\Gamma \Vdash M \text{ to } x:A \text{ in } N : C}$$

Assigning types to effectful programs

Aim: To fix the typing rule of `sequential composition`

Option 1: We could `restrict the free variables` in \underline{C} : [Levy'04]

$$\frac{\Gamma \Vdash M : F A \quad \Gamma \vdash \underline{C} \quad \Gamma, x:A \Vdash N : \underline{C}}{\Gamma \Vdash M \text{ to } x:A \text{ in } N : \underline{C}}$$

But sometimes it is useful if \underline{C} can depend on x !

- if we consider

`fopen (return true, return false)` to $x:\text{Bool}$ in N

- then it would be natural to let \underline{C} depend on x , e.g.,

$x:\text{Bool} \vdash \underline{C}(x) \stackrel{\text{def}}{=} \text{if } x \text{ then "allow fread, fwrite, and fclose"} \\ \text{else "allow fopen"}$

(needs more expressive comp. types than we consider here)

Assigning types to effectful programs

Aim: To fix the typing rule of sequential composition

Option 2: One could lift sequential composition to type level

$$\Gamma \Vdash M \text{ to } x:A \text{ in } N : M \text{ to } x:A \text{ in } \underline{C}$$

But then all computation types would be singleton-like!?!

Option 3: In the monadic metalanguage λ_{ML} , one could also try

$$\frac{\Gamma \vdash M : T A \quad \Gamma, x:A \vdash N : T B(x)}{\Gamma \vdash M \text{ to } x:A \text{ in } N : T (\Sigma x:A. B)}$$

But what makes this a principled solution? Why is it correct?

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 2: One could **lift sequential composition** to type level

$$\Gamma \Vdash M \text{ to } x:A \text{ in } N : M \text{ to } x:A \text{ in } \underline{C}$$

But then all computation types would be singleton-like!?!

Option 3: In the monadic metalanguage λ_{ML} , one could also try

$$\frac{\Gamma \vdash M : T A \quad \Gamma, x:A \vdash N : T B(x)}{\Gamma \vdash M \text{ to } x:A \text{ in } N : T (\Sigma x:A. B)}$$

But what makes this a principled solution? Why is it correct?

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 2: One could **lift sequential composition** to type level

$$\Gamma \Vdash M \text{ to } x:A \text{ in } N : M \text{ to } x:A \text{ in } \underline{C}$$

But then all computation types would be singleton-like!?!

Option 3: In the **monadic metalanguage** λ_{ML} , one could also try

$$\frac{\Gamma \vdash M : T A \quad \Gamma, x:A \vdash N : T B(x)}{\Gamma \vdash M \text{ to } x:A \text{ in } N : T (\Sigma x : A. B)}$$

But what makes this a principled solution? Why is it correct?

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 4: We draw inspiration from algebraic effects

- and combine it with restricting \underline{C} in seq. comp. (**Option 1**)

E.g., consider the non-det. program $(\text{for } x:\text{Nat} \models N : \underline{C}(x))$

$$M \stackrel{\text{def}}{=} \text{choose}(\text{return } 4, \text{return } 2) \text{ to } x:\text{Nat} \text{ in } N$$

After tossing the coin, this program evaluates as either

$$N[4/x] : \underline{C}[4/x] \quad \text{or} \quad N[2/x] : \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

$$\underline{C}[4/x] + \underline{C}[2/x] \quad \stackrel{\text{def}}{=} \quad F \left(U(\underline{C}[4/x]) + U(\underline{C}[2/x]) \right) /_{\equiv}$$

and thus we would like to type M at the type $\sum x:\text{Nat}. \underline{C}$

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 4: We draw inspiration from **algebraic effects**

- and combine it with restricting \underline{C} in seq. comp. (**Option 1**)

E.g., consider the non-det. program $(\text{for } x:\text{Nat} \models N : \underline{C}(x))$

$$M \stackrel{\text{def}}{=} \text{choose}(\text{return } 4, \text{return } 2) \text{ to } x:\text{Nat} \text{ in } N$$

After tossing the coin, this program evaluates as either

$$N[4/x] : \underline{C}[4/x] \quad \text{or} \quad N[2/x] : \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

$$\underline{C}[4/x] + \underline{C}[2/x] \quad \stackrel{\text{def}}{=} \quad F \left(U(\underline{C}[4/x]) + U(\underline{C}[2/x]) \right) /_{\equiv}$$

and thus we would like to type M at the type $\Sigma x:\text{Nat}. \underline{C}$

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 4: We draw inspiration from **algebraic effects**

- and combine it with restricting \underline{C} in seq. comp. (**Option 1**)

E.g., consider the **non-det.** program $(\text{for } x:\text{Nat} \models N : \underline{C}(x))$

$$M \stackrel{\text{def}}{=} \text{choose}(\text{return } 4, \text{return } 2) \text{ to } x:\text{Nat} \text{ in } N$$

After tossing the coin, this program evaluates as either

$$N[4/x] : \underline{C}[4/x] \quad \text{or} \quad N[2/x] : \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

$$\underline{C}[4/x] + \underline{C}[2/x] \quad \stackrel{\text{def}}{=} \quad F \left(U(\underline{C}[4/x]) + U(\underline{C}[2/x]) \right) /_{\equiv}$$

and thus we would like to type M at the type $\Sigma x:\text{Nat}. \underline{C}$

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 4: We draw inspiration from **algebraic effects**

- and combine it with restricting \underline{C} in seq. comp. (**Option 1**)

E.g., consider the **non-det.** program $\left(\text{for } x:\text{Nat} \models N : \underline{C}(x)\right)$

$$M \stackrel{\text{def}}{=} \text{choose}(\text{return } 4, \text{return } 2) \text{ to } x:\text{Nat} \text{ in } N$$

After **tossing the coin**, this program evaluates as either

$$N[4/x] : \underline{C}[4/x] \quad \text{or} \quad N[2/x] : \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

$$\underline{C}[4/x] + \underline{C}[2/x] \stackrel{\text{def}}{=} F\left(U(\underline{C}[4/x]) + U(\underline{C}[2/x])\right) / \equiv$$

and thus we would like to type M at the type $\Sigma x:\text{Nat}. \underline{C}$

Assigning types to effectful programs

Aim: To fix the typing rule of **sequential composition**

Option 4: We draw inspiration from **algebraic effects**

- and combine it with restricting \underline{C} in seq. comp. (**Option 1**)

E.g., consider the **non-det.** program $\left(\text{for } x:\text{Nat} \models N : \underline{C}(x)\right)$

$$M \stackrel{\text{def}}{=} \text{choose}(\text{return } 4, \text{return } 2) \text{ to } x:\text{Nat} \text{ in } N$$

After **tossing the coin**, this program evaluates as either

$$N[4/x] : \underline{C}[4/x] \quad \text{or} \quad N[2/x] : \underline{C}[2/x]$$

Idea: M denotes an element of the **coproduct of algebras**

$$\underline{C}[4/x] + \underline{C}[2/x] \quad \stackrel{\text{def}}{=} \quad F \left(U(\underline{C}[4/x]) + U(\underline{C}[2/x]) \right) /_{\equiv}$$

and thus we would like to type M at the type $\sum x:\text{Nat}. \underline{C}$

Putting these ideas together

(eMLTT: a core dep.-typed language with comp. effects)

eMLTT – types

Value types: MLTT + **thunks** + ...

$A, B ::= \text{Nat} \mid 1 \mid 0 \mid \Pi x:A. B \mid \Sigma x:A. B \mid V =_A W \mid \underline{U} \underline{C} \mid \dots$

- $\underline{U} \underline{C}$ is the type of **thunked** (i.e., suspended) **computations**

Computation types: dep.-typed version of EEC's comp. types

$\underline{C}, \underline{D} ::= F A \mid \Pi x:A. \underline{C} \mid \Sigma x:A. \underline{C}$

- $F A$ is the type of computations returning values of type A
- $\Pi x:A. \underline{C}$ is the type of dependent effectful functions
 - generalises CBPV/EEC's comp. types $A \rightarrow \underline{C}$ and $\underline{C} \times \underline{D}$
- $\Sigma x:A. \underline{C}$ is the type of dep. pairs of values and effectful comps.
 - captures the intuition about seq. comp. and coprods. of algebras
 - generalises EEC's comp. types $!A \otimes \underline{C}$ and $\underline{C} \oplus \underline{D}$

eMLTT – types

Value types: MLTT + **thunks** + ...

$A, B ::= \text{Nat} \mid 1 \mid 0 \mid \Pi x:A. B \mid \Sigma x:A. B \mid V =_A W \mid \underline{U} \underline{C} \mid \dots$

- $\underline{U} \underline{C}$ is the type of **thunked** (i.e., suspended) **computations**

Computation types: dep.-typed version of EEC's comp. types

$\underline{C}, \underline{D} ::= F A \mid \Pi x:A. \underline{C} \mid \Sigma x:A. \underline{C}$

- $F A$ is the type of computations **returning values** of type A
- $\Pi x:A. \underline{C}$ is the type of dependent **effectful functions**
 - generalises CBPV/EEC's comp. types $A \rightarrow \underline{C}$ and $\underline{C} \times \underline{D}$
- $\Sigma x:A. \underline{C}$ is the type of **dep. pairs** of values and effectful comps.
 - captures the intuition about seq. comp. and **coprods. of algebras**
 - generalises EEC's comp. types $!A \otimes \underline{C}$ and $\underline{C} \oplus \underline{D}$

eMLTT – terms

Value terms: MLTT + *thunks* + ...

$$V, W ::= x \mid \text{zero} \mid \text{succ } V \mid \dots \mid \text{thunk } M \mid \dots$$

- equational theory based on *intensional* MLTT

Comp. terms: dep.-typed version of CBPV/EEC's comp. terms

$$\begin{array}{lcl} M, N ::= & \text{force } V & \\ & \text{return } V & \\ & M \text{ to } x:A \text{ in } N & \\ & \lambda x:A. M & \\ & MV & \\ & \langle V, M \rangle & \text{(comp. } \Sigma \text{ intro.)} \\ & M \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } K & \text{(comp. } \Sigma \text{ elim.)} \end{array}$$

But: Value and comp. terms alone do not suffice, as in EEC!

eMLTT – terms

Value terms: MLTT + *thunks* + ...

$$V, W ::= x \mid \text{zero} \mid \text{succ } V \mid \dots \mid \text{thunk } M \mid \dots$$

- equational theory based on *intensional* MLTT

Comp. terms: dep.-typed version of CBPV/EEC's comp. terms

$$\begin{array}{ll} M, N ::= & \text{force } V \\ & \mid \text{return } V \\ & \mid M \text{ to } x:A \text{ in } N \\ & \mid \lambda x:A. M \\ & \mid MV \\ & \mid \langle V, M \rangle & (\text{comp. } \Sigma \text{ intro.}) \\ & \mid M \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } K & (\text{comp. } \Sigma \text{ elim.}) \end{array}$$

But: Value and comp. terms alone do not suffice, as in EEC!

eMLTT – terms

Value terms: MLTT + *thunks* + ...

$$V, W ::= x \mid \text{zero} \mid \text{succ } V \mid \dots \mid \text{thunk } M \mid \dots$$

- equational theory based on *intensional* MLTT

Comp. terms: dep.-typed version of CBPV/EEC's comp. terms

$$\begin{array}{lcl} M, N ::= & \text{force } V & \\ & | \text{return } V & \\ & | M \text{ to } x:A \text{ in } N & \\ & | \lambda x:A. M & \\ & | MV & \\ & | \langle V, M \rangle & (\text{comp. } \Sigma \text{ intro.}) \\ & | M \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } K & (\text{comp. } \Sigma \text{ elim.}) \end{array}$$

But: Value and comp. terms alone do not suffice, as in EEC!

eMLTT – terms

Note: We need to define K in such a way that the intended left-to-right evaluation order is preserved, e.g., consider

$$\Gamma \Vdash \langle V, M \rangle \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } K = K[V/x, M/z] : \underline{D}$$

Homomorphism terms: dep.-typed version of EEC's linear terms

$$\begin{array}{ll} K, L ::= & z \quad \text{(linear comp. vars.)} \\ & K \text{ to } x:A \text{ in } M \\ & \lambda x:A. K \\ & KV \\ & \langle V, K \rangle \quad \text{(comp. } \Sigma \text{ intro.)} \\ & K \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } L \quad \text{(comp. } \Sigma \text{ elim.)} \end{array}$$

Typing judgments:

- $\Gamma \Vdash V : A$
- $\Gamma \Vdash M : \underline{C}$
- $\Gamma \mid z:\underline{C} \Vdash K : \underline{D}$ (linear in z ; comp. bound to z happens first)

eMLTT – terms

Note: We need to define K in such a way that the intended left-to-right evaluation order is preserved, e.g., consider

$$\Gamma \Vdash \langle V, M \rangle \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } K = K[V/x, M/z] : \underline{D}$$

Homomorphism terms: dep.-typed version of EEC's linear terms

$$\begin{array}{ll} K, L ::= & z \quad (\text{linear comp. vars.}) \\ & | \quad K \text{ to } x:A \text{ in } M \\ & | \quad \lambda x:A. K \\ & | \quad K V \\ & | \quad \langle V, K \rangle \quad (\text{comp. } \Sigma \text{ intro.}) \\ & | \quad K \text{ to } \langle x:A, z:\underline{C} \rangle \text{ in } L \quad (\text{comp. } \Sigma \text{ elim.}) \end{array}$$

Typing judgments:

- $\Gamma \Vdash V : A$
- $\Gamma \Vdash M : \underline{C}$
- $\Gamma \mid z:\underline{C} \Vdash K : \underline{D}$ (linear in z ; comp. bound to z happens first)

eMLTT – typing sequential composition

We can then account for **type-dependency in seq. comp.** as

$$\frac{\Gamma \Vdash M : F A \quad \Gamma \vdash \Sigma y:A. \underline{C}(y) \quad \frac{\Gamma, x:A \Vdash N : \underline{C}(x)}{\Gamma, x:A \Vdash \langle x, N \rangle : \Sigma y:A. \underline{C}(y)}}{\Gamma \Vdash M \text{ to } x:A \text{ in } \langle x, N \rangle : \Sigma y:A. \underline{C}(y)}$$

The **seq. comp. rule for λ_{ML}** is justified by the type isomorphism

$$\frac{\Gamma \vdash A \quad \Gamma, x:A \vdash B(x)}{\Gamma \vdash U(\Sigma x:A. F(B)) \cong UF(\Sigma x:A. B) = T(\Sigma x:A. B)}$$

Categorical semantics of eMLTT

(fibrations + adjunctions)

Categorical semantics – MLTT part

We define **fibred adjunction models** $(\mathcal{B}, \mathcal{V}, p, q, F \dashv U)$

- **Theorem:** a sound and complete class of models for eMLTT

given by: i) a split closed comprehension cat. p (with s. fib. 0, ...)



- we define a partial interpretation fun. $\llbracket - \rrbracket$, that (if defined) maps:
 - a context Γ to an object $\llbracket \Gamma \rrbracket$ in \mathcal{B} , with $\llbracket \Gamma, x:A \rrbracket \stackrel{\text{def}}{=} \{\llbracket \Gamma; A \rrbracket\}$
 - a context Γ and a value type A to an object $\llbracket \Gamma; A \rrbracket$ in $\mathcal{V}_{\llbracket \Gamma \rrbracket}$
 - a context Γ and a value term V to $\llbracket \Gamma; V \rrbracket : 1_{\llbracket \Gamma \rrbracket} \longrightarrow A$ in $\mathcal{V}_{\llbracket \Gamma \rrbracket}$

Categorical semantics – MLTT part

We define **fibred adjunction models** $(\mathcal{B}, \mathcal{V}, p, q, F \dashv U)$

- **Theorem:** a sound and complete class of models for eMLTT given by: i) a **split closed comprehension cat.** \mathcal{P} (with s. fib. 0, ...)

$$\begin{array}{c} \mathcal{V} \\ \left(\begin{array}{c} \dashv \uparrow 1 \dashv \\ \downarrow \\ \mathcal{B} \end{array} \right) \{-\} \end{array}$$

\mathcal{P}

- we define a **partial interpretation fun.** $\llbracket - \rrbracket$, that (if defined) maps:
 - a **context** Γ to an object $\llbracket \Gamma \rrbracket$ in \mathcal{B} , with $\llbracket \Gamma, x:A \rrbracket \stackrel{\text{def}}{=} \{\llbracket \Gamma; A \rrbracket\}$
 - a context Γ and a **value type** A to an object $\llbracket \Gamma; A \rrbracket$ in $\mathcal{V}_{\llbracket \Gamma \rrbracket}$
 - a context Γ and a **value term** V to $\llbracket \Gamma; V \rrbracket : 1_{\llbracket \Gamma \rrbracket} \longrightarrow \llbracket \Gamma; A \rrbracket$ in $\mathcal{V}_{\llbracket \Gamma \rrbracket}$

Categorical semantics – MLTT part

We define **fibred adjunction models** $(\mathcal{B}, \mathcal{V}, p, q, F \dashv U)$

- **Theorem:** a sound and complete class of models for eMLTT given by: i) a **split closed comprehension cat.** \mathcal{P} (with s. fib. 0, ...)

$$\begin{array}{c} \mathcal{V} \\ \left(\begin{array}{c} \lrcorner \uparrow 1 \lrcorner \\ \downarrow \end{array} \right) \{-\} \\ \mathcal{B} \end{array}$$

- the **display maps** $\pi_{[\Gamma; A]} : [\Gamma, x:A] \longrightarrow [\Gamma]$ in \mathcal{B} induce the **weakening functors** $\pi_{[\Gamma; A]}^* : \mathcal{V}_{[\Gamma]} \longrightarrow \mathcal{V}_{[\Gamma, x:A]}$, and
- the value Σ - and Π -types are interpreted as **adjoints**

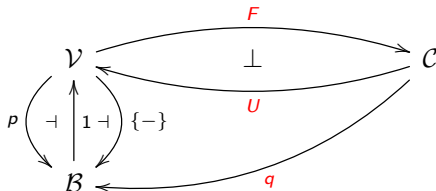
$$\Sigma_{[\Gamma; A]} \dashv \pi_{[\Gamma; A]}^* : \mathcal{V}_{[\Gamma]} \longrightarrow \mathcal{V}_{[\Gamma, x:A]} \quad (\text{such that } \Sigma \text{ is strong})$$

$$\pi_{[\Gamma; A]}^* \dashv \Pi_{[\Gamma; A]} : \mathcal{V}_{[\Gamma, x:A]} \longrightarrow \mathcal{V}_{[\Gamma]}$$

Categorical semantics – effects part

We define **fibred adjunction models** $(\mathcal{B}, \mathcal{V}, p, q, F \dashv U)$

- **Theorem:** a sound and complete class of models for eMLTT given by: ii) a **split fibration** q (with ...) and a **s. fib. adj.** $F \dashv U$

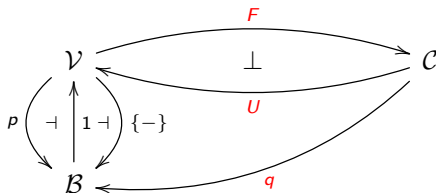


- we **extend** $\llbracket - \rrbracket$ so that (if defined) it maps:
 - a ctx. Γ and a **comp. type** \underline{C} to an object $\llbracket \Gamma; \underline{C} \rrbracket$ in $\mathcal{C}_{\llbracket \Gamma \rrbracket}$
 - a ctx. Γ and a **comp. term** M to $\llbracket \Gamma; M \rrbracket : 1_{\llbracket \Gamma \rrbracket} \longrightarrow U(\underline{C})$ in $\mathcal{V}_{\llbracket \Gamma \rrbracket}$
 - a ctx. Γ , a c. var. z , a c. type \underline{C} , and a **hom. term** K to $\llbracket \Gamma; z : \underline{C}; K \rrbracket : \llbracket \Gamma; \underline{C} \rrbracket \longrightarrow \underline{D}$ in $\mathcal{C}_{\llbracket \Gamma \rrbracket}$

Categorical semantics – effects part

We define **fibred adjunction models** $(\mathcal{B}, \mathcal{V}, p, q, F \dashv U)$

- **Theorem:** a sound and complete class of models for eMLTT given by: ii) a **split fibration** q (with ...) and a **s. fib. adj.** $F \dashv U$



- we again have **weakening functors** $\pi_{[\Gamma; A]}^*: \mathcal{C}_{[\Gamma]} \longrightarrow \mathcal{C}_{[\Gamma, x:A]}$, and
- the comp. Σ - and Π -types are interpreted again as **adjoints**

$$\Sigma_{[\Gamma; A]} \dashv \pi_{[\Gamma; A]}^*: \mathcal{C}_{[\Gamma]} \longrightarrow \mathcal{C}_{[\Gamma, x:A]}$$

$$\pi_{[\Gamma; A]}^* \dashv \Pi_{[\Gamma; A]}: \mathcal{C}_{[\Gamma, x:A]} \longrightarrow \mathcal{C}_{[\Gamma]}$$

Digression: dep. elimination of 0 and +

The coproduct type $A + B$:

[Jacobs'99]

- require $p : \mathcal{V} \longrightarrow \mathcal{B}$ to have split fibred coproducts $A +_X B$, and
- $\langle \{\text{inl}_A\}^*, \{\text{inr}_B\}^* \rangle : \mathcal{V}_{\{A +_X B\}} \longrightarrow \mathcal{V}_{\{A\}} \times \mathcal{V}_{\{B\}}$ to be fully-faith.
- allows one to interpret dependent case analysis, i.e.,

$$\begin{aligned} \mathcal{V}_{\{A\}} \left(1_{\{A\}}, \{\text{inl}_A\}^*(C) \right) \times \mathcal{V}_{\{B\}} \left(1_{\{B\}}, \{\text{inr}_B\}^*(C) \right) \\ \cong \\ \mathcal{V}_{\{A +_X B\}} \left(1_{\{A +_X B\}}, C \right) \end{aligned}$$

provides semantics for

$$\frac{\Gamma, y_1 : A \Vdash W_1 : C[\text{inl}_A y_1/x] \quad \Gamma, y_2 : B \Vdash W_2 : C[\text{inr}_B y_2/x]}{\Gamma, x : A + B \Vdash \text{case } x \text{ of } (\text{inl}(y_1) \mapsto W_1, \text{inr}(y_2) \mapsto W_2) : C}$$

Digression: dep. elimination of 0 and +

The coproduct type $A + B$:

[Jacobs'99]

- require $p : \mathcal{V} \longrightarrow \mathcal{B}$ to have **split fibred coproducts** $A +_X B$, and
- $\langle \{\text{inl}_A\}^*, \{\text{inr}_B\}^* \rangle : \mathcal{V}_{\{A+_X B\}} \longrightarrow \mathcal{V}_{\{A\}} \times \mathcal{V}_{\{B\}}$ to be **fully-faith.**
- allows one to interpret dependent case analysis, i.e.,

$$\begin{aligned} \mathcal{V}_{\{A\}} \left(1_{\{A\}}, \{\text{inl}_A\}^*(C) \right) \times \mathcal{V}_{\{B\}} \left(1_{\{B\}}, \{\text{inr}_B\}^*(C) \right) \\ \cong \\ \mathcal{V}_{\{A+_X B\}} \left(1_{\{A+_X B\}}, C \right) \end{aligned}$$

provides semantics for

$$\frac{\Gamma, y_1 : A \Vdash W_1 : C[\text{inl}_A y_1/x] \quad \Gamma, y_2 : B \Vdash W_2 : C[\text{inr}_B y_2/x]}{\Gamma, x : A + B \Vdash \text{case } x \text{ of } (\text{inl}(y_1) \mapsto W_1, \text{inr}(y_2) \mapsto W_2) : C}$$

Digression: dep. elimination of 0 and +

The coproduct type $A + B$:

[Jacobs'99]

- require $p : \mathcal{V} \longrightarrow \mathcal{B}$ to have **split fibred coproducts** $A +_X B$, and
- $\langle \{\text{inl}_A\}^*, \{\text{inr}_B\}^* \rangle : \mathcal{V}_{\{A+_X B\}} \longrightarrow \mathcal{V}_{\{A\}} \times \mathcal{V}_{\{B\}}$ to be **fully-faith.**
- allows one to interpret **dependent case analysis**, i.e.,

$$\begin{aligned} \mathcal{V}_{\{A\}} \left(1_{\{A\}}, \{\text{inl}_A\}^*(C) \right) \times \mathcal{V}_{\{B\}} \left(1_{\{B\}}, \{\text{inr}_B\}^*(C) \right) \\ \cong \\ \mathcal{V}_{\{A+_X B\}} \left(1_{\{A+_X B\}}, C \right) \end{aligned}$$

provides semantics for

$$\frac{\Gamma, y_1 : A \Vdash W_1 : C[\text{inl}_A y_1/x] \quad \Gamma, y_2 : B \Vdash W_2 : C[\text{inr}_B y_2/x]}{\Gamma, x : A + B \Vdash \text{case } x \text{ of } (\text{inl}(y_1) \mapsto W_1, \text{inr}(y_2) \mapsto W_2) : C}$$

Digression: dep. elimination of colimits

A generalisation:

[Ahman'17]

- **Idea:** **fully-faith.** for cocones $A \longrightarrow A \otimes_X B \longleftarrow B$ is enough, and we can generalise this to all **split fibred colimits**

- **Theorem:**

- if for every object $X \in \mathcal{B}$ and diagram $J : \mathcal{D} \longrightarrow \mathcal{V}_X$ there exists a cocone $\underline{\text{in}}^J : J \longrightarrow \Delta(\underline{\text{colim}}(J))$ in \mathcal{V}_X ,
- such that $f^*(\underline{\text{in}}_D^J) = \underline{\text{in}}_D^{f^* \circ J}$, for any $f : X \longrightarrow Y$, and such that the unique mediating functor

$$\langle \{\underline{\text{in}}_D^J\}_{D \in \mathcal{D}}^* \rangle : \mathcal{V}_{\{\underline{\text{colim}}(J)\}} \longrightarrow \text{lim}(\hat{J})$$

is fully-faithful (for $\hat{J} : \mathcal{D}^{op} \longrightarrow \text{Cat}$, where $\hat{J}(D) = \mathcal{V}_{\{J(D)\}}$),

- then p has split fibred colimits of shape \mathcal{D} , and p supports dependent elimination for them (analogously to $+_X$)

Digression: dep. elimination of colimits

A generalisation:

[Ahman'17]

- **Idea:** **fully-faith.** for cocones $A \longrightarrow A \otimes_X B \longleftarrow B$ is enough, and we can generalise this to all **split fibred colimits**

- **Theorem:**

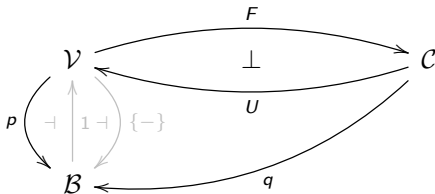
- if for every object $X \in \mathcal{B}$ and diagram $J : \mathcal{D} \longrightarrow \mathcal{V}_X$ there exists a **cocone** $\underline{\text{in}}^J : J \longrightarrow \Delta(\underline{\text{colim}}(J))$ in \mathcal{V}_X ,
- such that $f^*(\underline{\text{in}}^J_D) = \underline{\text{in}}^{f^* \circ J}_D$, for any $f : X \longrightarrow Y$, and such that the **unique mediating functor**

$$\langle \{\underline{\text{in}}^J_D\}_{D \in \mathcal{D}}^* \rangle : \mathcal{V}_{\{\underline{\text{colim}}(J)\}} \longrightarrow \lim(\hat{J})$$

is **fully-faithful** (for $\hat{J} : \mathcal{D}^{op} \longrightarrow \text{Cat}$, where $\hat{J}(D) = \mathcal{V}_{\{J(D)\}}$),

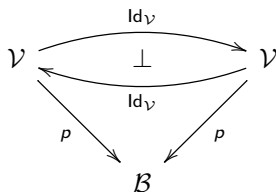
- then p has **split fibred colimits** of shape \mathcal{D} , and p supports **dependent elimination** for them (analogously to $+_X$)

Examples of fibred adjunction models



Examples of fibred adjunction models

Example 1 (identity adjunctions):

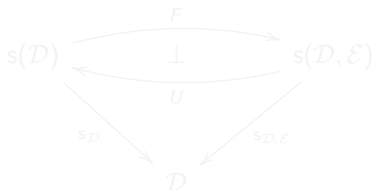


- **Note:** sound model as long as we haven't included any effects

Examples of fibred adjunction models

Example 2 (simple models from Egger et al.'s EEC):

- given an adjunction $F_{\text{EEC}} \dashv U_{\text{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
 - \mathcal{D} is Cartesian closed (with $0, \dots$), and
 - $F_{\text{EEC}} \dashv U_{\text{EEC}}$ and \mathcal{E} are \mathcal{D} -enriched, and
 - \mathcal{E} has all \mathcal{D} -tensors ($A \otimes \underline{C}$) and \mathcal{D} -cotensors ($A \rightrightarrows \underline{C}$)
- we use simple fibration $s_{\mathcal{D}}$ and simpl. \mathcal{D} -enrich. fibration $s_{\mathcal{D}, \mathcal{E}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F_{\text{EEC}}(A))$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U_{\text{EEC}}(\underline{C}))$$

$$s(\mathcal{D}): \quad (f, g) : (X, A) \longrightarrow (Y, B) \quad \text{where} \quad f : X \longrightarrow Y \quad g : X \times A \longrightarrow B$$

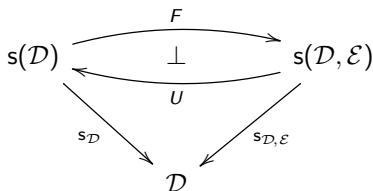
$$s(\mathcal{D}, \mathcal{E}): \quad (f, h) : (X, \underline{C}) \longrightarrow (Y, \underline{D}) \quad \text{where} \quad f : X \longrightarrow Y \quad h : X \otimes \underline{C} \longrightarrow \underline{D}$$

- Note:** this model doesn't support any real type-dependency

Examples of fibred adjunction models

Example 2 (simple models from Egger et al.'s EEC):

- given an **adjunction** $F_{\text{EEC}} \dashv U_{\text{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
 - \mathcal{D} is **Cartesian closed** (with $0, \dots$), and
 - $F_{\text{EEC}} \dashv U_{\text{EEC}}$ and \mathcal{E} are **\mathcal{D} -enriched**, and
 - \mathcal{E} has all **\mathcal{D} -tensors** ($A \otimes \underline{C}$) and **\mathcal{D} -cotensors** ($A \rightrightarrows \underline{C}$)
- we use **simple fibration** $s_{\mathcal{D}}$ and **simpl. \mathcal{D} -enrich. fibration** $s_{\mathcal{D}, \mathcal{E}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F_{\text{EEC}}(A))$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U_{\text{EEC}}(\underline{C}))$$

$$s(\mathcal{D}): \quad (f, g) : (X, A) \longrightarrow (Y, B) \quad \text{where} \quad f : X \longrightarrow Y \quad g : X \times A \longrightarrow B$$

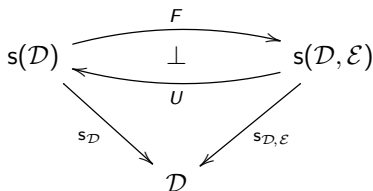
$$s(\mathcal{D}, \mathcal{E}): \quad (f, h) : (X, \underline{C}) \longrightarrow (Y, \underline{D}) \quad \text{where} \quad f : X \longrightarrow Y \quad h : X \otimes \underline{C} \longrightarrow \underline{D}$$

- Note:** this model doesn't support any real type-dependency

Examples of fibred adjunction models

Example 2 (simple models from Egger et al.'s EEC):

- given an **adjunction** $F_{\text{EEC}} \dashv U_{\text{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
 - \mathcal{D} is **Cartesian closed** (with $0, \dots$), and
 - $F_{\text{EEC}} \dashv U_{\text{EEC}}$ and \mathcal{E} are **\mathcal{D} -enriched**, and
 - \mathcal{E} has all **\mathcal{D} -tensors** ($A \otimes \underline{C}$) and **\mathcal{D} -cotensors** ($A \rightrightarrows \underline{C}$)
- we use **simple fibration** $s_{\mathcal{D}}$ and **simpl. \mathcal{D} -enrich. fibration** $s_{\mathcal{D}, \mathcal{E}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F_{\text{EEC}}(A))$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U_{\text{EEC}}(\underline{C}))$$

$$s(\mathcal{D}): \quad (f, g) : (X, A) \longrightarrow (Y, B) \quad \text{where} \quad f : X \longrightarrow Y \quad g : X \times A \longrightarrow B$$

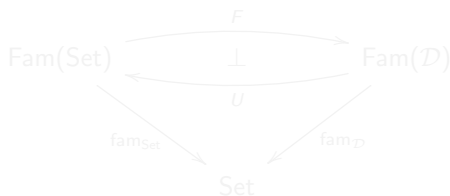
$$s(\mathcal{D}, \mathcal{E}): \quad (f, h) : (X, \underline{C}) \longrightarrow (Y, \underline{D}) \quad \text{where} \quad f : X \longrightarrow Y \quad h : X \otimes \underline{C} \longrightarrow \underline{D}$$

- Note:** this model doesn't support any real type-dependency

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathbf{Set}$, such that
 - \mathcal{D} has set-indexed products and set-indexed coproducts
- such adjunctions arise from
 - EM-cats. ($\mathcal{D} \stackrel{\text{def}}{=} \mathbf{Set}^{\mathbf{T}}$) and Law. ths. ($\mathcal{D} \stackrel{\text{def}}{=} \mathbf{Mod}(\mathcal{L}, \mathbf{Set})$)
 - resolutions of $S \Rightarrow (-) \times S$ and $((-) \Rightarrow R) \Rightarrow R$
- we use families fibrations $\text{fam}_{\mathbf{Set}}$ and $\text{fam}_{\mathcal{D}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F_{\mathcal{D}} \circ A)$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U_{\mathcal{D}} \circ \underline{C})$$

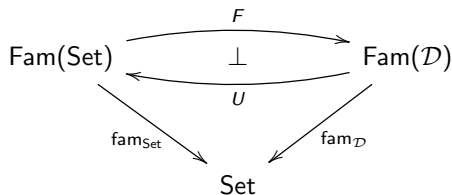
$\text{Fam}(\mathbf{Set}) : (X, A) \quad \text{where} \quad X \in \mathbf{Set} \quad A : X \longrightarrow \mathbf{Set}$

$(f, \{g_x\}_{x \in X}) : (X, A) \longrightarrow (Y, B) \quad \text{where} \quad g_x : A(x) \longrightarrow (B \circ f)(x)$

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathbf{Set}$, such that
 - \mathcal{D} has set-indexed products and set-indexed coproducts
- such adjunctions arise from
 - EM-cats. $(\mathcal{D} \stackrel{\text{def}}{=} \mathbf{Set}^{\mathbf{T}})$ and Law. ths. $(\mathcal{D} \stackrel{\text{def}}{=} \mathbf{Mod}(\mathcal{L}, \mathbf{Set}))$
 - resolutions of $S \Rightarrow (-) \times S$ and $((-) \Rightarrow R) \Rightarrow R$
- we use families fibrations $\mathbf{fam}_{\mathbf{Set}}$ and $\mathbf{fam}_{\mathcal{D}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F_{\mathcal{D}} \circ A)$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U_{\mathcal{D}} \circ \underline{C})$$

$$\mathbf{Fam}(\mathbf{Set}): (X, A) \quad \text{where} \quad X \in \mathbf{Set} \quad A : X \longrightarrow \mathbf{Set}$$

$$(f, \{g_x\}_{x \in X}) : (X, A) \longrightarrow (Y, B) \quad \text{where} \quad g_x : A(x) \longrightarrow (B \circ f)(x)$$

Examples of fibred adjunction models

Example 4 (continuous families for $\mu x: U\underline{C}. M$):

- given a **CPO-enriched monad** \mathbf{T} on CPO, such that
 - \mathbf{T} supports least zero-ary alg. op. ($\perp_A : 1 \rightarrow TA$), and
 - $\text{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- such \mathbf{T} arise from **discrete CPO-enriched countable Law. ths.**
- we use continuous families fibrations cfam_{CPO} and $\text{cfam}_{\text{CPO}^{\mathbf{T}}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F^{\mathbf{T}} \circ A)$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U^{\mathbf{T}} \circ \underline{C})$$

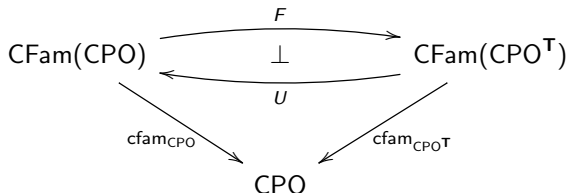
$\text{CFam}(\text{CPO})$: (X, A) where $X \in \text{CPO}$ $A : X \rightarrow \text{CPO}^{\text{EP}}$ an ω -cont. fun.

- Thm.:** we don't use $\text{cod} : \text{CPO}^{\rightarrow} \rightarrow \text{CPO}$ because CPO isn't LCCC

Examples of fibred adjunction models

Example 4 (continuous families for $\mu x: U\underline{C}. M$):

- given a **CPO-enriched monad** \mathbf{T} on CPO, such that
 - \mathbf{T} supports least zero-ary alg. op. ($\perp_A : 1 \rightarrow TA$), and
 - $\text{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- such \mathbf{T} arise from **discrete CPO-enriched countable Law. ths.**
- we use **continuous families fibrations** cfam_{CPO} and $\text{cfam}_{\text{CPO}^{\mathbf{T}}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F^{\mathbf{T}} \circ A)$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U^{\mathbf{T}} \circ \underline{C})$$

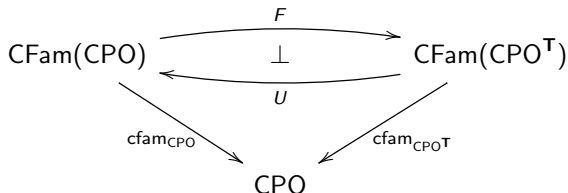
$\text{CFam}(\text{CPO}): (X, A)$ where $X \in \text{CPO}$ $A : X \rightarrow \text{CPO}^{\text{red}}$ an ω -cont. fun.

- Thm.:** we don't use $\text{cod} : \text{CPO}^{\rightarrow} \rightarrow \text{CPO}$ because CPO isn't LCCC

Examples of fibred adjunction models

Example 4 (continuous families for $\mu x: U\underline{C}. M$):

- given a **CPO-enriched monad** \mathbf{T} on CPO, such that
 - \mathbf{T} supports least zero-ary alg. op. ($\perp_A : 1 \rightarrow TA$), and
 - $\text{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- such \mathbf{T} arise from **discrete CPO-enriched countable Law. ths.**
- we use **continuous families fibrations** cfam_{CPO} and $\text{cfam}_{\text{CPO}^{\mathbf{T}}}$



$$F(X, A) \stackrel{\text{def}}{=} (X, F^{\mathbf{T}} \circ A)$$

$$U(X, \underline{C}) \stackrel{\text{def}}{=} (X, U^{\mathbf{T}} \circ \underline{C})$$

$\text{CFam}(\text{CPO})$: (X, A) where $X \in \text{CPO}$ $A : X \rightarrow \text{CPO}^{\text{red}}$ an ω -cont. fun.

- Thm.:** we don't use $\text{cod} : \text{CPO}^{\rightarrow} \rightarrow \text{CPO}$ because CPO isn't LCCC

Examples of fibred adjunction models

Example 5 (EM-resolutions of split fibred monads):

- given a **split fibred monad** $\mathbf{T} = (T, \eta, \mu)$ on p , i.e.,

$$\begin{array}{ccc} \mathcal{V} & \xrightarrow{T} & \mathcal{V} \\ & \searrow p & \swarrow p \\ & \mathcal{B} & \end{array} \quad \text{and} \quad p(\eta_A) = \text{id}_{p(A)} \quad p(\mu_A) = \text{id}_{p(A)}$$

- we consider models based on the **EM-resolution** of \mathbf{T}

$$\begin{array}{ccc} \mathcal{V} & \begin{array}{c} \xrightarrow{F^T} \\ \perp \\ \xleftarrow{U^T} \end{array} & \mathcal{V}^T \\ & \searrow p & \swarrow p^T \\ & \mathcal{B} & \end{array}$$

- and show that **three familiar results** hold for this situation

Examples of fibred adjunction models

Example 5 (EM-resolutions of split fibred monads):

- **Theorem 1:** if p supports Π -types, then $p^{\mathbf{T}}$ also supports Π -types

$$\Pi_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} (\Pi_A(B), \beta_{\Pi_A^{\mathbf{T}}})$$

- **Proposition:** every \mathbf{T} on a s. closed comp. cat. has a dep. str.

$$\sigma_A : \Sigma_A \circ T \longrightarrow T \circ \Sigma_A \quad (A \in \mathcal{V})$$

- **Theorem 2:** if p supports Σ -types and σ_A are natural isos., then $p^{\mathbf{T}}$ also supports Σ -types

$$\Sigma_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} (\Sigma_A(B), \beta_{\Sigma_A^{\mathbf{T}}})$$

- **Theorem 3:** if p supports Σ -types and $p^{\mathbf{T}}$ has split fibred reflexive coequalizers, then $p^{\mathbf{T}}$ also supports Σ -types

$$\Sigma_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} f^{\mathbf{T}}(\Sigma_A(B)) /_{\equiv}$$

Examples of fibred adjunction models

Example 5 (EM-resolutions of split fibred monads):

- **Theorem 1:** if p supports Π -types, then $p^{\mathbf{T}}$ also supports Π -types

$$\Pi_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} (\Pi_A(B), \beta_{\Pi_A^{\mathbf{T}}})$$

- **Proposition:** every \mathbf{T} on a s. closed comp. cat. has a dep. str.

$$\sigma_A : \Sigma_A \circ T \longrightarrow T \circ \Sigma_A \quad (A \in \mathcal{V})$$

- **Theorem 2:** if p supports Σ -types and σ_A are natural isos., then $p^{\mathbf{T}}$ also supports Σ -types

$$\Sigma_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} (\Sigma_A(B), \beta_{\Sigma_A^{\mathbf{T}}})$$

- **Theorem 3:** if p supports Σ -types and $p^{\mathbf{T}}$ has split fibred reflexive coequalizers, then $p^{\mathbf{T}}$ also supports Σ -types

$$\Sigma_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} \ulcorner^{\mathbf{T}}(\Sigma_A(B)) /_{\equiv}$$

Examples of fibred adjunction models

Example 5 (EM-resolutions of split fibred monads):

- **Theorem 1:** if p supports Π -types, then $p^{\mathbf{T}}$ also supports Π -types

$$\Pi_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} (\Pi_A(B), \beta_{\Pi_A^{\mathbf{T}}})$$

- **Proposition:** every \mathbf{T} on a s. closed comp. cat. has a dep. str.

$$\sigma_A : \Sigma_A \circ T \longrightarrow T \circ \Sigma_A \quad (A \in \mathcal{V})$$

- **Theorem 2:** if p supports Σ -types and σ_A are natural isos., then $p^{\mathbf{T}}$ also supports Σ -types

$$\Sigma_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} (\Sigma_A(B), \beta_{\Sigma_A^{\mathbf{T}}})$$

- **Theorem 3:** if p supports Σ -types and $p^{\mathbf{T}}$ has split fibred reflexive coequalizers, then $p^{\mathbf{T}}$ also supports Σ -types

$$\Sigma_A^{\mathbf{T}}(B, \beta) \stackrel{\text{def}}{=} F^{\mathbf{T}}(\Sigma_A(B)) /_{\equiv}$$

Algebraic effects

Algebraic effects – ops. and eqs.

Fibred effect theories \mathcal{T}_{eff} :

- signatures of **dep. typed operation symbols**

$$\frac{\cdot \vdash I \quad x_i : I \vdash O \quad I \text{ and } O \text{ are pure value types}}{\text{op} : (x_i : I) \longrightarrow O}$$

- equipped with **equations** on derivable effect terms

In eMLTT:

$$M ::= \dots \mid \text{op}_V^C(x.M)$$

General algebraicity equations (in addition to eff. th. eqs.):

$$\frac{\Gamma \Vdash V : I \quad \Gamma, x : O[V/x_i] \Vdash M : \underline{C} \quad \Gamma \mid z : \underline{C} \Vdash K : \underline{D}}{\Gamma \Vdash K[\text{op}_V^C(x.M)/z] = \text{op}_V^D(x.K[M/z]) : \underline{D}} \quad (\text{op} : (x_i : I) \longrightarrow O)$$

Sound semantics: based on

- $p : \text{Fam}(\text{Set}) \longrightarrow \text{Set}$ and $q : \text{Fam}(\text{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \text{Set})) \longrightarrow \text{Set}$

Algebraic effects – ops. and eqs.

Fibred effect theories \mathcal{T}_{eff} :

- signatures of **dep. typed operation symbols**

$$\frac{\cdot \vdash I \quad x_i : I \vdash O \quad I \text{ and } O \text{ are pure value types}}{\text{op} : (x_i : I) \longrightarrow O}$$

- equipped with **equations** on derivable effect terms

In eMLTT:

$$M ::= \dots \mid \text{op}_V^C(x.M)$$

General algebraicity equations (in addition to eff. th. eqs.):

$$\frac{\Gamma \Vdash V : I \quad \Gamma, x : O[V/x_i] \Vdash M : \underline{C} \quad \Gamma \mid z : \underline{C} \Vdash K : \underline{D}}{\Gamma \Vdash K[\text{op}_V^C(x.M)/z] = \text{op}_V^D(x.K[M/z]) : \underline{D}} \quad (\text{op} : (x_i : I) \longrightarrow O)$$

Sound semantics: based on

- $p : \text{Fam}(\text{Set}) \longrightarrow \text{Set}$ and $q : \text{Fam}(\text{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \text{Set})) \longrightarrow \text{Set}$

Algebraic effects – ops. and eqs.

Fibred effect theories \mathcal{T}_{eff} :

- signatures of **dep. typed operation symbols**

$$\frac{\cdot \vdash I \quad x_i : I \vdash O \quad I \text{ and } O \text{ are pure value types}}{\text{op} : (x_i : I) \longrightarrow O}$$

- equipped with **equations** on derivable effect terms

In eMLTT:

$$M ::= \dots \mid \text{op}_V^C(x.M)$$

General algebraicity equations (in addition to **eff. th. eqs.**):

$$\frac{\Gamma \Vdash V : I \quad \Gamma, x : O[V/x_i] \Vdash M : \underline{C} \quad \Gamma \mid \underline{z} : \underline{C} \Vdash \underline{K} : \underline{D}}{\Gamma \Vdash \underline{K}[\text{op}_V^C(x.M)/\underline{z}] = \text{op}_V^D(x.\underline{K}[M/\underline{z}]) : \underline{D}} \quad (\text{op} : (x_i : I) \longrightarrow O)$$

Sound semantics: based on

- $p : \text{Fam}(\text{Set}) \longrightarrow \text{Set}$ and $q : \text{Fam}(\text{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \text{Set})) \longrightarrow \text{Set}$

Algebraic effects – ops. and eqs.

Fibred effect theories \mathcal{T}_{eff} :

- signatures of **dep. typed operation symbols**

$$\frac{\cdot \vdash I \quad x_i : I \vdash O \quad I \text{ and } O \text{ are pure value types}}{\text{op} : (x_i : I) \longrightarrow O}$$

- equipped with **equations** on derivable effect terms

In eMLTT:

$$M ::= \dots \mid \text{op}_V^C(x.M)$$

General algebraicity equations (in addition to **eff. th. eqs.**):

$$\frac{\Gamma \Vdash V : I \quad \Gamma, x : O[V/x_i] \Vdash M : \underline{C} \quad \Gamma \mid \underline{z} : \underline{C} \Vdash \underline{K} : \underline{D}}{\Gamma \Vdash \underline{K}[\text{op}_V^C(x.M)/\underline{z}] = \text{op}_V^D(x.\underline{K}[M/\underline{z}]) : \underline{D}} \quad (\text{op} : (x_i : I) \longrightarrow O)$$

Sound semantics: based on

- $p : \text{Fam}(\text{Set}) \longrightarrow \text{Set}$ and $q : \text{Fam}(\text{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \text{Set})) \longrightarrow \text{Set}$

Algebraic effects – examples

Example 1 (interactive I/O):

- $\text{read} : 1 \longrightarrow \text{Chr}$
 $\text{write} : \text{Chr} \longrightarrow 1$
- no equations

$$(\text{Chr} \stackrel{\text{def}}{=} 1 + \dots + 1)$$

Example 2 (global state with location-dependent store type):

- $\diamond \vdash \text{Loc}$
 $\ell : \text{Loc} \vdash \text{Val}$
 $\diamond \Vdash \text{isDec}_{\text{Loc}} : \prod \ell : \text{Loc} . \prod \ell' : \text{Loc} . (\ell =_{\text{Loc}} \ell') + (\ell =_{\text{Loc}} \ell' \rightarrow 0)$
- $\text{get} : (\ell : \text{Loc}) \longrightarrow \text{Val}$
 $\text{put} : (\sum \ell : \text{Loc} . \text{Val}) \longrightarrow 1$
- five equations (two of them branching on $\text{isDec}_{\text{Loc}}$)

Example 3 (dep. typed update monads $T X \stackrel{\text{def}}{=} \prod_{s:S} . P s \times X$)

Algebraic effects – examples

Example 1 (interactive I/O):

- $\text{read} : 1 \longrightarrow \text{Chr}$

$$(\text{Chr} \stackrel{\text{def}}{=} 1 + \dots + 1)$$

$\text{write} : \text{Chr} \longrightarrow 1$

- no equations

Example 2 (global state with location-dependent store type):

- $\diamond \vdash \text{Loc}$

$\ell : \text{Loc} \vdash \text{Val}$

$$\diamond \Vdash \text{isDec}_{\text{Loc}} : \prod \ell : \text{Loc} . \prod \ell' : \text{Loc} . (\ell =_{\text{Loc}} \ell') + (\ell =_{\text{Loc}} \ell' \rightarrow 0)$$

- $\text{get} : (\ell : \text{Loc}) \longrightarrow \text{Val}$

$\text{put} : (\sum \ell : \text{Loc} . \text{Val}) \longrightarrow 1$

- five equations (two of them branching on $\text{isDec}_{\text{Loc}}$)

Example 3 (dep. typed update monads $T X \stackrel{\text{def}}{=} \prod_{s:S} . P s \times X$)

Algebraic effects – examples

Example 1 (interactive I/O):

- $\text{read} : 1 \longrightarrow \text{Chr}$

$$(\text{Chr} \stackrel{\text{def}}{=} 1 + \dots + 1)$$

$\text{write} : \text{Chr} \longrightarrow 1$

- no equations

Example 2 (global state with location-dependent store type):

- $\diamond \vdash \text{Loc}$

$\ell : \text{Loc} \vdash \text{Val}$

$$\diamond \Vdash \text{isDec}_{\text{Loc}} : \prod \ell : \text{Loc} . \prod \ell' : \text{Loc} . (\ell =_{\text{Loc}} \ell') + (\ell =_{\text{Loc}} \ell' \rightarrow 0)$$

- $\text{get} : (\ell : \text{Loc}) \longrightarrow \text{Val}$

$\text{put} : (\sum \ell : \text{Loc} . \text{Val}) \longrightarrow 1$

- five equations (two of them branching on $\text{isDec}_{\text{Loc}}$)

Example 3 (dep. typed update monads $T X \stackrel{\text{def}}{=} \prod_{s:S} . P s \times X$)

Handlers of algebraic effects

(for programming and extrinsic reasoning)

Handlers of alg. effects – for programming

Idea: Generalisation of exception handlers [Plotkin, Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

$$\frac{\Gamma \Vdash M : FA \quad \Gamma, x_v : I, x_k : O[x_v/x_i] \rightarrow U_{\underline{C}} \Vdash N_{\text{op}} : \underline{C} \quad \Gamma, y : A \Vdash N_{\text{ret}} : \underline{C}}{\Gamma \Vdash M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}} \text{ to } y : A \text{ in } \underline{C} \Vdash N_{\text{ret}} : \underline{C}}$$

satisfying

$(\text{return } V) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{T}_{\text{eff}}} \text{ to } y : A \text{ in } N_{\text{ret}} = N_{\text{ret}}[V/x]$

$(\text{op}_V^{\underline{C}}(x.M)) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{T}_{\text{eff}}} \text{ to } y : A \text{ in } N_{\text{ret}} = N_{\text{op}}[V/x_v][\dots/x_k]$

Typical use case for programming:

- write your programs using alg. ops. (e.g., get and put)
- use handlers to provide fit-for-purpose impl. (e.g., $S \rightarrow X \times S$)

Handlers of alg. effects – for programming

Idea: Generalisation of exception handlers [Plotkin, Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

$$\frac{\Gamma \vdash M : FA \quad \Gamma, x_v : I, x_k : O[x_v/x_i] \rightarrow U\mathcal{C} \vdash N_{\text{op}} : \mathcal{C} \quad \Gamma, y : A \vdash N_{\text{ret}} : \mathcal{C}}{\Gamma \vdash M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}} \text{ to } y : A \text{ in } \mathcal{C} \quad N_{\text{ret}} : \mathcal{C}}$$

satisfying

(return V) handled with $\{\dots\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y : A$ in $N_{\text{ret}} = N_{\text{ret}}[V/x]$

($\text{op}_{\mathcal{C}}^{\mathcal{C}}(x.M)$) handled with $\{\dots\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y : A$ in $N_{\text{ret}} = N_{\text{op}}[V/x_v][\dots/x_k]$

Typical use case for programming:

- write your programs using alg. ops. (e.g., get and put)
- use handlers to provide fit-for-purpose impl. (e.g., $S \rightarrow X \times S$)

Handlers of alg. effects – for programming

Idea: Generalisation of exception handlers [Plotkin, Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

$$\frac{\Gamma \vdash M : FA \quad \Gamma, x_v : I, x_k : O[x_v/x_i] \rightarrow U\mathcal{C} \vdash N_{\text{op}} : \mathcal{C} \quad \Gamma, y : A \vdash N_{\text{ret}} : \mathcal{C}}{\Gamma \vdash M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}} \text{ to } y : A \text{ in } \mathcal{C} \quad N_{\text{ret}} : \mathcal{C}}$$

satisfying

(return V) handled with $\{\dots\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y : A$ in $N_{\text{ret}} = N_{\text{ret}}[V/x]$

($\text{op}_{\mathcal{C}}^{\mathcal{C}}(x.M)$) handled with $\{\dots\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y : A$ in $N_{\text{ret}} = N_{\text{op}}[V/x_v][\dots/x_k]$

Typical use case for programming:

- write your programs using alg. ops. (e.g., get and put)
- use handlers to provide fit-for-purpose impl. (e.g., $S \rightarrow X \times S$)

Handlers of alg. effects – for reasoning

Idea: Using a derived handle-into-values handling construct

M handled with $\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y:A \text{ in}_B V_{\text{ret}}$

we can define predicates (essentially, dependent types)

$$\Gamma \Vdash P : UFA \rightarrow \mathcal{U}$$

by

- equipping a universe \mathcal{U} with an algebra for \mathcal{T}_{eff} , and
- using the above handle-into-values construct to define P

Note 1: $P(\text{thunk } M)$ computes a proof obligation for M

Note 2: Formally, we work in an extension of eMLTT with

- a universe \mathcal{U} closed under $\text{Nat}, 1, 0, +, \Sigma, \Pi$
- a type-based treatment of handlers $\underline{C} ::= \dots \mid \langle A; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle$
- function extensionality (actually, it's a bit more extensional)

Handlers of alg. effects – for reasoning

Idea: Using a derived **handle-into-values** handling construct

M handled with $\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y:A$ in $\textcolor{red}{B}$ V_{ret}

we can define **predicates** (essentially, dependent types)

$$\Gamma \vdash P : UFA \rightarrow \mathcal{U}$$

by

- equipping a universe \mathcal{U} with an **algebra** for \mathcal{T}_{eff} , and
- using the above **handle-into-values** construct to define P

Note 1: $P(\text{thunk } M)$ computes a proof obligation for M

Note 2: Formally, we work in an extension of eMLTT with

- a universe \mathcal{U} closed under $\text{Nat}, 1, 0, +, \Sigma, \Pi$
- a type-based treatment of handlers $\underline{C} ::= \dots \mid \langle A; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle$
- function extensionality (actually, it's a bit more extensional)

Handlers of alg. effects – for reasoning

Idea: Using a derived **handle-into-values** handling construct

M handled with $\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y:A$ in $\textcolor{red}{B}$ V_{ret}

we can define **predicates** (essentially, dependent types)

$$\Gamma \vdash P : UFA \rightarrow \mathcal{U}$$

by

- equipping a universe \mathcal{U} with an **algebra** for \mathcal{T}_{eff} , and
- using the above **handle-into-values** construct to define P

Note 1: $P(\text{thunk } M)$ computes a **proof obligation** for M

Note 2: Formally, we work in an extension of eMLTT with

- a universe \mathcal{U} closed under $\text{Nat}, 1, 0, +, \Sigma, \Pi$
- a type-based treatment of handlers $\underline{C} ::= \dots \mid \langle A; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle$
- function extensionality (actually, it's a bit more extensional)

Handlers of alg. effects – for reasoning

Idea: Using a derived **handle-into-values** handling construct

M handled with $\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op} \in \mathcal{T}_{\text{eff}}}$ to $y:A$ in $\textcolor{red}{B}$ V_{ret}

we can define **predicates** (essentially, dependent types)

$$\Gamma \vdash P : UFA \rightarrow \mathcal{U}$$

by

- equipping a universe \mathcal{U} with an **algebra** for \mathcal{T}_{eff} , and
- using the above **handle-into-values** construct to define P

Note 1: $P(\text{thunk } M)$ computes a **proof obligation** for M

Note 2: Formally, we work in an extension of eMLTT with

- a universe \mathcal{U} closed under $\text{Nat}, 1, 0, +, \Sigma, \Pi$
- a **type-based treatment of handlers** $\underline{C} ::= \dots \mid \langle A; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle$
- function extensionality (actually, it's a bit more extensional)

Handlers of alg. effects – for reasoning

Example 1 (Evaluation Logic style modalities):

- Given a predicate $P : A \rightarrow \mathcal{U}$ on return values,
we define a predicate $\Diamond P : UFA \rightarrow \mathcal{U}$ on I/O-computations as

$$\Diamond P \stackrel{\text{def}}{=} \lambda x : UFA. (\text{force } x) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{T}_{\text{IO}}} \text{ to } y : A \text{ in } P y$$

using the handler given by

$$V_{\text{read}} \stackrel{\text{def}}{=} \lambda x : (\Sigma x_v : 1. \text{Chr} \rightarrow \mathcal{U}). \widehat{\Sigma} y : \text{El}(\widehat{\text{Chr}}). (\text{snd } x) y$$

$$V_{\text{write}} \stackrel{\text{def}}{=} \lambda x : (\Sigma x_v : \text{Chr}. 1 \rightarrow \mathcal{U}). (\text{snd } x) \star$$

- $\Diamond P$ corresponds to Evaluation Logic's possibility modality

$$\Diamond P (\text{think}(\text{read}(x.\text{write}_{e'}(\text{return } V)))) = \widehat{\Sigma} x : \text{El}(\widehat{\text{Chr}}). P V$$

- To get the necessity modality, use $\widehat{\Pi} x : \text{El}(\widehat{\text{Chr}})$ in V_{read}

Handlers of alg. effects – for reasoning

Example 1 (Evaluation Logic style modalities):

- Given a predicate $P : A \rightarrow \mathcal{U}$ on return values,
we define a predicate $\Diamond P : UFA \rightarrow \mathcal{U}$ on I/O-computations as

$$\Diamond P \stackrel{\text{def}}{=} \lambda x : UFA. (\text{force } x) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{T}_{\text{IO}}} \text{ to } y : A \text{ in } P y$$

using the handler given by

$$V_{\text{read}} \stackrel{\text{def}}{=} \lambda x : (\sum x_v : 1. \text{Chr} \rightarrow \mathcal{U}). \widehat{\Sigma} y : \text{El}(\widehat{\text{Chr}}). (\text{snd } x) y$$

$$V_{\text{write}} \stackrel{\text{def}}{=} \lambda x : (\sum x_v : \text{Chr} . 1 \rightarrow \mathcal{U}). (\text{snd } x) \star$$

- $\Diamond P$ corresponds to Evaluation Logic's possibility modality

$$\Diamond P (\text{think}(\text{read}(x.\text{write}_{e'}(\text{return } V)))) = \widehat{\Sigma} x : \text{El}(\widehat{\text{Chr}}). P V$$

- To get the necessity modality, use $\widehat{\Pi} x : \text{El}(\widehat{\text{Chr}})$ in V_{read}

Handlers of alg. effects – for reasoning

Example 1 (Evaluation Logic style modalities):

- Given a predicate $P : A \rightarrow \mathcal{U}$ on return values,
we define a predicate $\Diamond P : UFA \rightarrow \mathcal{U}$ on I/O-computations as

$$\Diamond P \stackrel{\text{def}}{=} \lambda x : UFA. (\text{force } x) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{T}_{\text{IO}}} \text{ to } y : A \text{ in } P y$$

using the handler given by

$$V_{\text{read}} \stackrel{\text{def}}{=} \lambda x : (\sum x_v : 1. \text{Chr} \rightarrow \mathcal{U}). \hat{\Sigma} y : \text{El}(\widehat{\text{Chr}}). (\text{snd } x) y$$

$$V_{\text{write}} \stackrel{\text{def}}{=} \lambda x : (\sum x_v : \text{Chr} . 1 \rightarrow \mathcal{U}). (\text{snd } x) \star$$

- $\Diamond P$ corresponds to Evaluation Logic's **possibility modality**

$$\Diamond P (\text{think}(\text{read}(x. \text{write}_{e'}(\text{return } V)))) = \hat{\Sigma} x : \text{El}(\widehat{\text{Chr}}). P V$$

- To get the necessity modality, use $\hat{\Pi} x : \text{El}(\widehat{\text{Chr}})$ in V_{read}

Handlers of alg. effects – for reasoning

Example 1 (Evaluation Logic style modalities):

- Given a predicate $P : A \rightarrow \mathcal{U}$ on return values,
we define a predicate $\Diamond P : UFA \rightarrow \mathcal{U}$ on I/O-computations as

$$\Diamond P \stackrel{\text{def}}{=} \lambda x : UFA. (\text{force } x) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{T}_{\text{IO}}} \text{ to } y : A \text{ in } P y$$

using the handler given by

$$V_{\text{read}} \stackrel{\text{def}}{=} \lambda x : (\sum x_v : 1. \text{Chr} \rightarrow \mathcal{U}). \widehat{\Sigma} y : \text{El}(\widehat{\text{Chr}}). (\text{snd } x) y$$

$$V_{\text{write}} \stackrel{\text{def}}{=} \lambda x : (\sum x_v : \text{Chr} . 1 \rightarrow \mathcal{U}). (\text{snd } x) \star$$

- $\Diamond P$ corresponds to Evaluation Logic's **possibility modality**

$$\Diamond P (\text{think}(\text{read}(x.\text{write}_{e'}(\text{return } V)))) = \widehat{\Sigma} x : \text{El}(\widehat{\text{Chr}}). P V$$

- To get the **necessity modality**, use $\widehat{\Pi} x : \text{El}(\widehat{\text{Chr}})$ in V_{read}

Handlers of alg. effects – for reasoning

Example 2 (Dijkstra's weakest precondition semantics):

- Given a postcondition on return values and final states

$$Q : A \rightarrow S \rightarrow \mathcal{U} \quad (S \stackrel{\text{def}}{=} \prod x:\text{Loc}. \text{Val})$$

we define a precondition for stateful comps. on initial states

$$\text{wp}_Q : \text{UFA} \rightarrow S \rightarrow \mathcal{U}$$

by

- i) handling the given comp. into a state-passing function using

$$V_{\text{get}}, V_{\text{put}} \text{ on } S \rightarrow (\mathcal{U} \times S) \quad \text{and} \quad V_{\text{ret}} \text{ " = " } Q$$

- ii) feeding in the initial state; and iii) projecting out \mathcal{U}

- Theorem:** wp_Q satisfies expected properties of WPs, e.g.,

$$\text{wp}_Q (\text{thunk}(\text{return } V)) = \lambda x_S : S. Q \ V \ x_S$$

$$\text{wp}_Q (\text{thunk}(\text{put}_{\langle \ell, V \rangle}(M))) = \lambda x_S : S. \text{wp}_Q (\text{thunk } M) (x_S[\ell \mapsto V])$$

Handlers of alg. effects – for reasoning

Example 2 (Dijkstra's weakest precondition semantics):

- Given a postcondition on **return values** and **final states**

$$Q : A \rightarrow S \rightarrow \mathcal{U} \quad (S \stackrel{\text{def}}{=} \prod x : \text{Loc} . \text{Val})$$

we define a precondition for **stateful comps.** on **initial states**

$$\text{wp}_Q : \text{UFA} \rightarrow S \rightarrow \mathcal{U}$$

by

- i)* handling the given comp. into a **state-passing function** using

$$V_{\text{get}}, V_{\text{put}} \quad \text{on} \quad S \rightarrow (\mathcal{U} \times S) \quad \text{and} \quad V_{\text{ret}} \text{ “=” } Q$$

- ii)* feeding in the **initial state**; and *iii)* **projecting** out \mathcal{U}

- Theorem:** wp_Q satisfies expected properties of WPs, e.g.,

$$\text{wp}_Q (\text{thunk}(\text{return } V)) = \lambda x_S : S . Q \ V \ x_S$$

$$\text{wp}_Q (\text{thunk}(\text{put}_{\langle \ell, V \rangle}(M))) = \lambda x_S : S . \text{wp}_Q (\text{thunk } M) (x_S[\ell \mapsto V])$$

Handlers of alg. effects – for reasoning

Example 2 (Dijkstra's weakest precondition semantics):

- Given a postcondition on **return values** and **final states**

$$Q : A \rightarrow S \rightarrow \mathcal{U} \quad (S \stackrel{\text{def}}{=} \prod x : \text{Loc} . \text{Val})$$

we define a precondition for **stateful comps.** on **initial states**

$$\text{wp}_Q : \text{UFA} \rightarrow S \rightarrow \mathcal{U}$$

by

- i)* handling the given comp. into a **state-passing function** using

$$V_{\text{get}}, V_{\text{put}} \quad \text{on} \quad S \rightarrow (\mathcal{U} \times S) \quad \text{and} \quad V_{\text{ret}} \text{ “=” } Q$$

- ii)* feeding in the **initial state**; and *iii)* **projecting** out \mathcal{U}

- Theorem:** wp_Q satisfies expected properties of WPs, e.g.,

$$\text{wp}_Q (\text{thunk}(\text{return } V)) = \lambda x_S : S . Q \text{ } V \text{ } x_S$$

$$\text{wp}_Q (\text{thunk}(\text{put}_{\langle \ell, V \rangle}(M))) = \lambda x_S : S . \text{wp}_Q (\text{thunk } M) (x_S[\ell \mapsto V])$$

Handlers of alg. effects – for reasoning

Example 3 (Patterns of allowed I/O-effects):

- Assuming an inductive type Protocol, given by

$$e : \text{Protocol} \quad r : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol}$$

$$w : (\text{Chr} \rightarrow \mathcal{U}) \rightarrow \text{Protocol} \rightarrow \text{Protocol}$$

and potentially also by \wedge, \vee, \dots

- Then, given a protocol $Pr : \text{Protocol}$, we define

$$\underline{Pr} : UFA \rightarrow \mathcal{U}$$

by handling the given comp. using

$$V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \rightarrow \mathcal{U}$$

where

$$V_{\text{read}} \langle -, V_{rk} \rangle (r \text{ Pr}') \stackrel{\text{def}}{=} \hat{\Pi} x : \text{El}(\widehat{\text{Chr}}) . (V_{rk} \ x) (Pr' \ x)$$

$$V_{\text{write}} \langle V, V_{wk} \rangle (w \ P \ Pr') \stackrel{\text{def}}{=} \hat{\Sigma} x : \text{El}(P \ V) . V_{wk} \star Pr'$$

$$\stackrel{\text{def}}{=} \hat{0}$$

Handlers of alg. effects – for reasoning

Example 3 (Patterns of allowed I/O-effects):

- Assuming an **inductive type** Protocol, given by

$$\mathbf{e} : \text{Protocol} \quad \mathbf{r} : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol}$$

$$\mathbf{w} : (\text{Chr} \rightarrow \mathcal{U}) \rightarrow \text{Protocol} \rightarrow \text{Protocol}$$

and potentially also by \wedge, \vee, \dots

- Then, given a protocol $\text{Pr} : \text{Protocol}$, we define

$$\underline{\text{Pr}} : \text{UFA} \rightarrow \mathcal{U}$$

by handling the given comp. using

$$V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \rightarrow \mathcal{U}$$

where

$$V_{\text{read}} \langle -, V_{rk} \rangle (\mathbf{r} \text{ Pr}') \stackrel{\text{def}}{=} \hat{\Pi} x : \text{El}(\widehat{\text{Chr}}) . (V_{rk} x) (\text{Pr}' x)$$

$$V_{\text{write}} \langle V, V_{wk} \rangle (\mathbf{w} P \text{ Pr}') \stackrel{\text{def}}{=} \hat{\Sigma} x : \text{El}(P V) . V_{wk} \star \text{Pr}'$$

$$\stackrel{\text{def}}{=} \hat{0}$$

Handlers of alg. effects – for reasoning

Example 3 (Patterns of allowed I/O-effects):

- Assuming an **inductive type** Protocol, given by

$$\mathbf{e} : \text{Protocol} \quad \mathbf{r} : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol}$$

$$\mathbf{w} : (\text{Chr} \rightarrow \mathcal{U}) \rightarrow \text{Protocol} \rightarrow \text{Protocol}$$

and potentially also by \wedge, \vee, \dots

- Then, given a protocol $\text{Pr} : \text{Protocol}$, we define

$$\underline{\text{Pr}} : \text{UFA} \rightarrow \mathcal{U}$$

by handling the given comp. using

$$V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \rightarrow \mathcal{U}$$

where

$$V_{\text{read}} \langle -, V_{\text{rk}} \rangle (\mathbf{r} \text{ Pr}') \stackrel{\text{def}}{=} \widehat{\Pi} x : \text{El}(\widehat{\text{Chr}}) . (V_{\text{rk}} x) (\text{Pr}' x)$$

$$V_{\text{write}} \langle V, V_{\text{wk}} \rangle (\mathbf{w} \text{ } P \text{ Pr}') \stackrel{\text{def}}{=} \widehat{\Sigma} x : \text{El}(P V) . V_{\text{wk}} \star \text{Pr}'$$

$$\text{—} \stackrel{\text{def}}{=} \widehat{0}$$

Conclusion

In work we told a mathematically natural story of combining

- dependent types and computational effects

In particular, we saw

- a clean core language of dependent types and comp. effects
- a natural category-theoretic semantics
- alg. effects and handlers, in particular, for reasoning using
 - Evaluation Logic style modalities
 - Dijkstra's weakest precondition semantics
 - patterns of allowed (I/O)-effects

Future work:

- type-dependency on computations
- more expressive computation types

Other works: directed containers, F^* and monotonic state, ...

Thank you!

D. Ahman.

Fibred Computational Effects. (PhD Thesis, 2017)

D. Ahman, N. Ghani, G. Plotkin.

Dependent Types and Fibred Computational Effects. (FoSSaCS'16)

D. Ahman.

Handling Fibred Computational Effects. (POPL'18)

Future work (type-dependency on comps.)

- How to accommodate $\underline{D}(\text{read}(x.M))$
- That is, how to avoid restricting the typing of seq. comp.?
- $M \text{ to } x:A \text{ in } N : \underline{C}[\text{thunk } M/y]$ (where $y:UFA$) [Vákár'17]
- $\alpha : \hat{T}(\mathcal{U}_{\text{comp}}) \longrightarrow \mathcal{U}_{\text{comp}}$ [Pédrot, Tabareau'17]
- for eMLTT, one possible way forward
 - i) build on Vákár's proposal
 - ii) but force type-dep. to be homomorphic
 - $\underline{D}[\text{thunk}(M \text{ to } x:A \text{ in } N)/y] = M \text{ to } x:A \text{ in } \underline{D}[\text{thunk } N/y]$
 - $\underline{D}[M \text{ to } x:A \text{ in } N/z] = M \text{ to } x:A \text{ in } \underline{D}[N/z]$