A fibrational view on computational effects

Danel Ahman

Prosecco Team, Inria Paris

November 13, 2017

Outline

Language design principles for combining

- dependent types $(\Pi, \Sigma, V =_A W, \ldots)$
- 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

Language design principles for combining

- dependent types $(\Pi, \Sigma, V =_A W, \ldots)$
- 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 (a clean core language)
- use established math. techniques
- cover a wide range of comp. effects
- discover smth. interesting

Outline

Language design principles for combining

- dependent types $(\Pi, \Sigma, V =_A W, \ldots)$
- 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 (a clean core language)
- use established math. techniques (fibrations and adjunctions)
- cover a wide range of comp. effects
- discover smth. interesting

Outline

Language design principles for combining

- dependent types $(\Pi, \Sigma, V =_A W, \ldots)$
- 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 (a clean core language)
- use established math. techniques (fibrations and adjunctions)
- cover a wide range of comp. effects (incl. algebraic effects)
- discover smth. interesting

Outline

Language design principles for combining

- dependent types ($\Pi, \Sigma, V =_A W, \ldots$)
- 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 (a clean core language)
- use established math. techniques (fibrations and adjunctions)
- cover a wide range of comp. effects (incl. algebraic effects)
- 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.:

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

which could be used to type the dependent function

$$\mathtt{sort} : \Pi\, \ell : (\mathsf{List}\ \mathsf{Chr}) \,.\, A(\ell)$$

Q: Should we allow $A[M/x]$ if M is an effectful program?

- e.g., if M is $\mathsf{read}(y.N)$

A1: In this work we say no

- types should only depend on static information
- e.g., how would one compute $A[\mathsf{receive}(y.M)/x]$ statically?
- we recover dependency on effectful computations via thunks

A2: We are also thinking about 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.:

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

which could be used to type the dependent function

$$\mathtt{sort} : \Pi\, \ell : (\mathsf{List\ Chr})\, .\, A(\ell)$$

Q: Should we allow $A[M/x]$ if M is an effectful program?

- e.g., if M is $\mathsf{read}(y.N)$

A1: In this work we say no

- types should only depend on static information
- e.g., how would one compute $A[\mathsf{receive}(y.M)/x]$ statically?
- we recover dependency on effectful computations via thunks

A2: We are also thinking about 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.:

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

which could be used to type the dependent function

$$\mathtt{sort} : \Pi\, \ell : (\mathsf{List\ Chr})\,.\, A(\ell)$$

Q: Should we allow $A[M/x]$ if M is an effectful program?

- e.g., if M is $\mathsf{read}(y.N)$

A1: In this work we say no

- types should only depend on static information
- e.g., how would one compute $A[\mathsf{receive}(y.M)/x]$ statically?
- we recover dependency on effectful computations via thunks

A2: We are also thinking about 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.:

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

which could be used to type the dependent function

$$\mathtt{sort} : \Pi\, \ell : (\mathsf{List}\ \mathsf{Chr})\,.\, A(\ell)$$

Q: Should we allow $A[M/x]$ if M is an effectful program?

- e.g., if M is $\mathsf{read}(y.N)$

A1: In this work we say no

- types should only depend on static information
- e.g., how would one compute $A[\mathsf{receive}(y.M)/x]$ statically?
- we recover dependency on effectful computations via thunks

A2: We are also thinking about 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. version of CBPV/EEC)
- where Γ contains **only** value variables $x_1 : A_1, \ldots, x_n : A_n$

Note: Some of the other options are 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. version of CBPV/EEC)
- where Γ contains **only** value variables $x_1 : A_1, \ldots, x_n : A_n$

Note: Some of the other options are 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. version of CBPV/EEC)
- where Γ contains **only** value variables $x_1 : A_1, \ldots, x_n : A_n$

Note: Some of the other options are 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., typing sequential composition)

Assigning types to effectful programs

The problem: The standard typing rule for seq. composition

$$\frac{\Gamma \vDash_{c} M : F\,A \qquad \Gamma, x\!:\!A \vDash_{c} N : \underline{C}}{\Gamma \vDash_{c} M \;\mathtt{to}\; x\!:\!A \;\mathtt{in}\; N : \underline{C}}$$

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

$$\underline{C}$$

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 $\underline{C}$, i.e., [Levy'04]

$$\frac{\Gamma \vdash_{\mathsf{c}} M : F\,A \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x\!:\!A \vdash_{\mathsf{c}} N : \underline{C}}{\Gamma \vdash_{\mathsf{c}} M \texttt{ to } x\!:\!A \texttt{ in } N : \underline{C}}$$

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

- if M involves opening a file and
  the return values of M model whether `fopen` succeeded,
- then it would be natural to let $\underline{C}$ to depend such values, e.g.,

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

(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 $\underline{C}$, i.e., [Levy'04]

$$\frac{\Gamma \vDash_c M : F\,A \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x\!:\!A \vDash_c N : \underline{C}}{\Gamma \vDash_c M \texttt{ to } x\!:\!A \texttt{ in } N : \underline{C}}$$

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

- if M involves opening a file and

  the return values of M model whether `fopen` succeeded,
- then it would be natural to let $\underline{C}$ to depend such values, e.g.,

$$x\!:\!\mathsf{Bool} \vdash \underline{C}(x) = \texttt{if } x \texttt{ then } \text{“allow } \texttt{fread}\text{, } \texttt{fwrite}\text{, and } \texttt{fclose}\text{”} \\ \texttt{else } \text{“allow } \texttt{fopen}\text{”}$$

(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 $\underline{C}$, i.e., [Levy'04]

$$\frac{\Gamma \vDash_{\mathsf{c}} M : F\,A \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x\!:\!A \vDash_{\mathsf{c}} N : \underline{C}}{\Gamma \vDash_{\mathsf{c}} M \ \mathtt{to}\ x\!:\!A\ \mathtt{in}\ N : \underline{C}}$$

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

- if M involves opening a file and

  the return values of M model whether `fopen` succeeded,
- then it would be natural to let $\underline{C}$ to depend such values, e.g.,

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

(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

$$\Gamma \vdash_c M \;\mathtt{to}\; x{:}A \;\mathtt{in}\; N : M \;\mathtt{to}\; x{:}A \;\mathtt{in}\; \underline{C}$$

But then all comp. types would be singleton-like:

- comp. types would contain exactly the terms we want to type!?!
- but could be useful for dependency on comps. (the yes case)

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

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

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

$$\Gamma \vDash_{c} M \ \texttt{to}\ x\!:\!A\ \texttt{in}\ N : M\ \texttt{to}\ x\!:\!A\ \texttt{in}\ \underline{C}$$

But then all comp. types would be singleton-like:

- comp. types would contain exactly the terms we want to type!?!
- but could be useful for dependency on comps. (the yes case)

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

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

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

$$\Gamma \vdash_{c} M \ \mathtt{to}\ x\!:\!A\ \mathtt{in}\ N : M\ \mathtt{to}\ x\!:\!A\ \mathtt{in}\ \underline{C}$$

But then all comp. types would be singleton-like:

- comp. types would contain exactly the terms we want to type!?!
- but could be useful for dependency on comps. (the yes case)

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

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

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 $\left(\text{for } x : \mathsf{Nat} \vdash_{c} N : \underline{C}(x)\right)$

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

After tossing the coin, this program evaluates as either

$$N[4/x] \ \text{ at type } \ \underline{C}[4/x] \qquad \text{or} \qquad N[2/x] \ \text{ at type } \ \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

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

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

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 $\left(\text{for } x : \mathsf{Nat} \vdash_c N : \underline{C}(x)\right)$

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

After tossing the coin, this program evaluates as either

$$N[4/x] \ \text{ at type } \ \underline{C}[4/x] \qquad \text{or} \qquad N[2/x] \ \text{ at type } \ \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

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

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

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 $\Big(\text{for } x : \mathsf{Nat} \vDash_{c} N : \underline{C}(x)\Big)$

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

After tossing the coin, this program evaluates as either

$$N[4/x] \ \text{ at type } \ \underline{C}[4/x] \qquad \text{or} \qquad N[2/x] \ \text{ at type } \ \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

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

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

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 $\Big(\text{for } x : \mathsf{Nat} \vDash_{\mathsf{c}} N : \underline{C}(x)\Big)$

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

After tossing the coin, this program evaluates as either

$$N[4/x] \ \text{ at type } \ \underline{C}[4/x] \qquad \text{or} \qquad N[2/x] \ \text{ at type } \ \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

$$\underline{C}[4/x] + \underline{C}[2/x] \quad \text{“}\overset{\text{def}}{=}\text{”} \quad F\Big(U\,(\underline{C}[4/x]) + U\,(\underline{C}[2/x])\Big)_{/\equiv}$$

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

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 $\left(\text{for } x : \mathsf{Nat} \vDash_{\mathsf{c}} N : \underline{C}(x)\right)$

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

After tossing the coin, this program evaluates as either

$$N[4/x] \ \text{ at type } \ \underline{C}[4/x] \qquad \text{or} \qquad N[2/x] \ \text{ at type } \ \underline{C}[2/x]$$

Idea: M denotes an element of the coproduct of algebras

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

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

Putting these ideas together

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

eMLTT

Recall: We aim to define a dependently-typed language with

- general computational effects
- a clear distinction between
  - values
  - computations
- with a principled treatment of sequential composition
  - restricting free variables in seq. composition
  - typing based on coproducts of algebras
- with a natural denotational semantics, using standard techniques
  - dep. types – split closed comprehension categories
  - comp. effects – adjunction models

eMLTT – types

Value types: MLTT + thunks + ...

$$A, B \ ::= \ \mathsf{Nat} \ | \ 1 \ | \ 0 \ | \ \Pi x{:}A.B \ | \ \Sigma x{:}A.B \ | \ V =_A W \ | \ U\,\underline{C} \ | \ \ldots$$

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

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

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

- $F\,A$ 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 $A \to \underline{C}$ 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 + ...

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

- $U\,\underline{C}$ 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 $A \to \underline{C}$ 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 + ...

$$V, W \ ::= \ x \ | \ \texttt{zero} \ | \ \texttt{succ}\, V \ | \ \ldots \ | \ \texttt{thunk}\ M \ | \ \ldots$$

- equational theory based on MLTT with intensional $V =_A W$

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

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

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

eMLTT – terms

Value terms: MLTT + thunks + ...

$$V, W \ ::= \ x \ | \ \mathtt{zero} \ | \ \mathtt{succ}\, V \ | \ \ldots \ | \ \mathtt{thunk}\ M \ | \ \ldots$$

- equational theory based on MLTT with intensional $V =_A W$

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

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

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

eMLTT – terms

Value terms: MLTT + thunks + ...

$$V, W \ ::= \ x \mid \texttt{zero} \mid \texttt{succ}\, V \mid \ \ldots \ \mid \texttt{thunk}\ M \mid \ \ldots$$

- equational theory based on MLTT with intensional $V =_A W$

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

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

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

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

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

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

Typing judgments:

- $\Gamma \vDash_{\mathsf{v}} V : A$
- $\Gamma \vDash_{\mathsf{c}} M : \underline{C}$
- $\Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D}$ (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

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

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

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

Typing judgments:

- $\Gamma \vDash_{v} V : A$
- $\Gamma \vDash_{c} M : \underline{C}$
- $\Gamma \,|\, z\!:\!\underline{C} \vDash_{h} K : \underline{D}$ (linear in z; comp. bound to z happens first)

eMLTT – typing sequential composition

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

$$\frac{\Gamma \vDash_{\mathsf{c}} M : F\,A \qquad \Gamma \vdash \Sigma y\,{:}\,A.\underline{C}(y) \qquad \dfrac{\Gamma, x\,{:}\,A \vDash_{\mathsf{c}} N : \underline{C}(x)}{\Gamma, x\,{:}\,A \vDash_{\mathsf{c}} \langle x, N\rangle : \Sigma y\,{:}\,A.\underline{C}(y)}}{\Gamma \vDash_{\mathsf{c}} M \texttt{ to } x\,{:}\,A \texttt{ in } \langle x, N\rangle : \Sigma y\,{:}\,A.\underline{C}(y)}$$

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

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

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, ...

$$\begin{tikzcd}
\mathcal{V} \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] \\
\mathcal{B} \arrow[u, "1" description]
\end{tikzcd}$$

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

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, ...

$$
\begin{tikzcd}
\mathcal{V} \arrow[d, bend right=60, "p"', "\dashv"] \arrow[d, bend left=60, "\{-\}", "\dashv"'] \\
\mathcal{B} \arrow[u, "1"']
\end{tikzcd}
$$

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

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, ...

$$
\begin{tikzcd}
\mathcal{V} \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] \\
\mathcal{B} \arrow[u, "\dashv"', "1\ \dashv"]
\end{tikzcd}
$$

- 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

$$\Sigma_{[\![\Gamma;A]\!]} \dashv \pi^*_{[\![\Gamma;A]\!]} : \mathcal{V}_{[\![\Gamma]\!]} \longrightarrow \mathcal{V}_{[\![\Gamma, x:A]\!]}$$

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

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. $F \dashv U$

$$
\begin{tikzcd}
\mathcal{V} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] & & \mathcal{C} \arrow[ll, bend left, "U"] \arrow[dll, bend left, "q"] \\
\mathcal{B} \arrow[u, "\dashv\ \ 1 \dashv"'] & &
\end{tikzcd}
$$

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

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. $F \dashv U$

$$\begin{tikzcd}
\mathcal{V} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[d, bend right, "p"'] \arrow[d, bend left, "\{-\}"] \arrow[d, phantom, "\dashv\ \ 1\ \dashv"] & & \mathcal{C} \arrow[ll, bend left, "U"] \arrow[dll, bend left, "q"] \\
\mathcal{B} \arrow[u, "1"'] & &
\end{tikzcd}$$

- the display maps $\pi_{[\![\Gamma;A]\!]} : [\![\Gamma, x : A]\!] \longrightarrow [\![\Gamma]\!]$ in $\mathcal{B}$
  induce the 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
- $\langle \{\mathsf{inl}_A\}^*, \{\mathsf{inr}_B\}^* \rangle : \mathcal{V}_{\{A +_X B\}} \longrightarrow \mathcal{V}_{\{A\}} \times \mathcal{V}_{\{B\}}$ to be fully-faith.

- allows one to interpret dependent case analysis, i.e.,

$$\mathcal{V}_{\{A\}}\Big(1_{\{A\}}, \{\mathsf{inl}_A\}^*(C)\Big) \times \mathcal{V}_{\{B\}}\Big(1_{\{B\}}, \{\mathsf{inr}_B\}^*(C)\Big)$$
$$\cong$$
$$\mathcal{V}_{\{A +_X B\}}\Big(1_{\{A +_X B\}}, C\Big)$$

provides semantics for

$$\frac{\Gamma, y_1 \!:\! A \vdash_{\mathsf{v}} W_1 : C[\mathtt{inl}_A\ y_1/x] \quad \Gamma, y_2 \!:\! B \vdash_{\mathsf{v}} W_2 : C[\mathtt{inr}_B\ y_2/x]}{\Gamma, x \!:\! A + B \vdash_{\mathsf{v}} \mathtt{case}\ x\ \mathtt{of}\ \big(\mathtt{inl}(y_1) \mapsto W_1, \mathtt{inr}(y_2) \mapsto W_2\big) : C}$$

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
- $\langle \{\mathsf{inl}_A\}^*, \{\mathsf{inr}_B\}^* \rangle : \mathcal{V}_{\{A +_X B\}} \longrightarrow \mathcal{V}_{\{A\}} \times \mathcal{V}_{\{B\}}$ to be fully-faith.

- allows one to interpret dependent case analysis, i.e.,

$$\mathcal{V}_{\{A\}}\Big(1_{\{A\}}, \{\mathsf{inl}_A\}^*(C)\Big) \times \mathcal{V}_{\{B\}}\Big(1_{\{B\}}, \{\mathsf{inr}_B\}^*(C)\Big)$$
$$\cong$$
$$\mathcal{V}_{\{A +_X B\}}\Big(1_{\{A +_X B\}}, C\Big)$$

provides semantics for

$$\frac{\Gamma, y_1 \mathbin{:} A \vdash_{\mathsf{v}} W_1 : C[\mathtt{inl}_A\ y_1/x] \quad \Gamma, y_2 \mathbin{:} B \vdash_{\mathsf{v}} W_2 : C[\mathtt{inr}_B\ y_2/x]}{\Gamma, x \mathbin{:} A + B \vdash_{\mathsf{v}} \mathtt{case}\ x\ \mathtt{of}\ \big(\mathtt{inl}(y_1) \mapsto W_1, \mathtt{inr}(y_2) \mapsto W_2\big) : C}$$

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
- $\langle \{\mathsf{inl}_A\}^*, \{\mathsf{inr}_B\}^* \rangle : \mathcal{V}_{\{A +_X B\}} \longrightarrow \mathcal{V}_{\{A\}} \times \mathcal{V}_{\{B\}}$ to be fully-faith.

- allows one to interpret dependent case analysis, i.e.,

$$\mathcal{V}_{\{A\}}\Big(1_{\{A\}}, \{\mathsf{inl}_A\}^*(C)\Big) \times \mathcal{V}_{\{B\}}\Big(1_{\{B\}}, \{\mathsf{inr}_B\}^*(C)\Big)$$
$$\cong$$
$$\mathcal{V}_{\{A +_X B\}}\Big(1_{\{A +_X B\}}, C\Big)$$

provides semantics for

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

Digression: dep. elimination of 0 and $+$

A generalisation: [Ahman'17]

- **Idea:** fully-faith. for cocones $A \longrightarrow A \circledast_X B \longleftarrow B$ 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 $\underline{\mathsf{in}}^J : J \longrightarrow \Delta(\underline{\mathsf{colim}}(J))$ in $\mathcal{V}_X$,
  - such that $f^*(\underline{\mathsf{in}}^J_D) = \underline{\mathsf{in}}^{f^* \circ J}_D$, for any $f : X \longrightarrow Y$, and such that the unique mediating functor
  $$\langle \{\underline{\mathsf{in}}^J_D\}^*_{D \in \mathcal{D}} \rangle : \mathcal{V}_{\{\underline{\mathsf{colim}}(J)\}} \longrightarrow \lim(\widehat{J})$$
  is fully-faithful (for $\widehat{J} : \mathcal{D}^{op} \longrightarrow \mathsf{Cat}$, where $\widehat{J}(D) = \mathcal{V}_{\{J(D)\}}$),
  - then p has split fibred colimits of shape $\mathcal{D}$, and p supports dependent elimination for them (analogously to $+_X$)

Digression: dep. elimination of 0 and $+$

A generalisation: [Ahman'17]

- **Idea:** fully-faith. for cocones $A \longrightarrow A \circledast_X B \longleftarrow B$ 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 $\underline{\mathsf{in}}^J : J \longrightarrow \Delta(\underline{\mathsf{colim}}(J))$ in $\mathcal{V}_X$,

  - such that $f^*(\underline{\mathsf{in}}^J_D) = \underline{\mathsf{in}}^{f^* \circ J}_D$, for any $f : X \longrightarrow Y$, and

    such that the unique mediating functor

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

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

  - 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

$$
\begin{tikzcd}
\mathcal{V} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[d, bend right=50, "p"'] \arrow[d, bend left=50, "\{-\}"] \arrow[d, phantom, shift right=3ex, "\dashv"] \arrow[d, phantom, shift left=3ex, "\dashv"] & & \mathcal{C} \arrow[ll, bend left, "U"] \arrow[dll, bend left, "q"] \\
\mathcal{B} \arrow[u, "1"'] & &
\end{tikzcd}
$$

Examples of fibred adjunction models

Example 1 (identity adjunctions): $\mathsf{Id}_{\mathcal{V}} \dashv \mathsf{Id}_{\mathcal{V}} : \mathcal{V} \longrightarrow \mathcal{V}$

- $\mathcal{C} \stackrel{\text{def}}{=} \mathcal{V}$
- $q \stackrel{\text{def}}{=} p$
- $F \stackrel{\text{def}}{=} \mathsf{Id}_{\mathcal{V}}$
- $U \stackrel{\text{def}}{=} \mathsf{Id}_{\mathcal{V}}$

$$
\begin{tikzcd}
\mathcal{V} \arrow[rr, bend left, "\mathsf{Id}_{\mathcal{V}}"] \arrow[rr, phantom, "\perp"] \arrow[d, bend right=50, "p"'] \arrow[d, bend left=50, "\{-\}"] & & \mathcal{V} \arrow[ll, bend left, "\mathsf{Id}_{\mathcal{V}}"] \arrow[dll, bend left, "p"] \\
\mathcal{B} \arrow[u, "\dashv"', "1 \dashv"] & &
\end{tikzcd}
$$

- sound as long as we haven't included any effects

Examples of fibred adjunction models

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

- given an adjunction $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is a CCC (with 0, ...), and
  - $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ and $\mathcal{E}$ are $\mathcal{D}$-enriched, and
  - $\mathcal{E}$ has all $\mathcal{D}$-tensors ($A \otimes \underline{C}$) and $\mathcal{D}$-cotensors ($A \Rightarrow \underline{C}$)
- we use the simple fibration $p : \mathsf{s}(\mathcal{D}) \longrightarrow \mathcal{D}$, where

$$p(f : X \longrightarrow Y, g : X \times A \longrightarrow B) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we define the simpl. $\mathcal{D}$-enrich. fib. $q : \mathsf{s}(\mathcal{D}, \mathcal{E}) \longrightarrow \mathcal{D}$, where

$$q(f : X \longrightarrow Y, g : X \otimes \underline{C} \longrightarrow \underline{D}) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ as follows:

$$F(A, B) \stackrel{\text{def}}{=} (A, F_{\mathsf{EEC}}(B)) \qquad U(A, \underline{C}) \stackrel{\text{def}}{=} (A, U_{\mathsf{EEC}}(\underline{C}))$$

- this model does not support any real type-dependency

Examples of fibred adjunction models

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

- given an adjunction $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is a CCC (with 0, ...), and
  - $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ and $\mathcal{E}$ are $\mathcal{D}$-enriched, and
  - $\mathcal{E}$ has all $\mathcal{D}$-tensors ($A \otimes \underline{C}$) and $\mathcal{D}$-cotensors ($A \Rightarrow \underline{C}$)
- we use the simple fibration $p : \mathsf{s}(\mathcal{D}) \longrightarrow \mathcal{D}$, where

$$p\big(f : X \longrightarrow Y, g : X \times A \longrightarrow B\big) \stackrel{\mathsf{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we define the simpl. $\mathcal{D}$-enrich. fib. $q : \mathsf{s}(\mathcal{D}, \mathcal{E}) \longrightarrow \mathcal{D}$, where

$$q\big(f : X \longrightarrow Y, g : X \otimes \underline{C} \longrightarrow \underline{D}\big) \stackrel{\mathsf{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ as follows:

$$F(A, B) \stackrel{\mathsf{def}}{=} (A, F_{\mathsf{EEC}}(B)) \qquad U(A, \underline{C}) \stackrel{\mathsf{def}}{=} (A, U_{\mathsf{EEC}}(\underline{C}))$$

- this model does not support any real type-dependency

Examples of fibred adjunction models

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

- given an adjunction $F_{\mathrm{EEC}} \dashv U_{\mathrm{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is a CCC (with $0, \ldots$), and
  - $F_{\mathrm{EEC}} \dashv U_{\mathrm{EEC}}$ and $\mathcal{E}$ are $\mathcal{D}$-enriched, and
  - $\mathcal{E}$ has all $\mathcal{D}$-tensors ($A \otimes \underline{C}$) and $\mathcal{D}$-cotensors ($A \Rightarrow \underline{C}$)
- we use the simple fibration $p : \mathrm{s}(\mathcal{D}) \longrightarrow \mathcal{D}$, where

$$p\big(f : X \longrightarrow Y, g : X \times A \longrightarrow B\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we define the simpl. $\mathcal{D}$-enrich. fib. $q : \mathrm{s}(\mathcal{D}, \mathcal{E}) \longrightarrow \mathcal{D}$, where

$$q\big(f : X \longrightarrow Y, g : X \otimes \underline{C} \longrightarrow \underline{D}\big) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathrm{EEC}} \dashv U_{\mathrm{EEC}}$ as follows:

$$F(A, B) \stackrel{\text{def}}{=} (A, F_{\mathrm{EEC}}(B)) \qquad U(A, \underline{C}) \stackrel{\text{def}}{=} (A, U_{\mathrm{EEC}}(\underline{C}))$$

- this model does not support any real type-dependency

Examples of fibred adjunction models

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

- given an adjunction $F_{\mathrm{EEC}} \dashv U_{\mathrm{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is a CCC (with 0, ...), and
  - $F_{\mathrm{EEC}} \dashv U_{\mathrm{EEC}}$ and $\mathcal{E}$ are $\mathcal{D}$-enriched, and
  - $\mathcal{E}$ has all $\mathcal{D}$-tensors ($A \otimes \underline{C}$) and $\mathcal{D}$-cotensors ($A \Rightarrow \underline{C}$)
- we use the simple fibration $p : \mathrm{s}(\mathcal{D}) \longrightarrow \mathcal{D}$, where

$$p\big(f : X \longrightarrow Y, g : X \times A \longrightarrow B\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we define the simpl. $\mathcal{D}$-enrich. fib. $q : \mathrm{s}(\mathcal{D}, \mathcal{E}) \longrightarrow \mathcal{D}$, where

$$q\big(f : X \longrightarrow Y, g : X \otimes \underline{C} \longrightarrow \underline{D}\big) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathrm{EEC}} \dashv U_{\mathrm{EEC}}$ as follows:

$$F(A, B) \stackrel{\text{def}}{=} (A, F_{\mathrm{EEC}}(B)) \qquad U(A, \underline{C}) \stackrel{\text{def}}{=} (A, U_{\mathrm{EEC}}(\underline{C}))$$

- this model does not support any real type-dependency

Examples of fibred adjunction models

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

- given an adjunction $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is a CCC (with 0, ...), and
  - $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ and $\mathcal{E}$ are $\mathcal{D}$-enriched, and
  - $\mathcal{E}$ has all $\mathcal{D}$-tensors ($A \otimes \underline{C}$) and $\mathcal{D}$-cotensors ($A \Rightarrow \underline{C}$)
- we use the simple fibration $p : \mathsf{s}(\mathcal{D}) \longrightarrow \mathcal{D}$, where

$$p\big(f : X \longrightarrow Y, g : X \times A \longrightarrow B\big) \overset{\mathsf{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we define the simpl. $\mathcal{D}$-enrich. fib. $q : \mathsf{s}(\mathcal{D}, \mathcal{E}) \longrightarrow \mathcal{D}$, where

$$q\big(f : X \longrightarrow Y, g : X \otimes \underline{C} \longrightarrow \underline{D}\big) \overset{\mathsf{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ as follows:

$$F(A, B) \overset{\mathsf{def}}{=} (A, F_{\mathsf{EEC}}(B)) \qquad U(A, \underline{C}) \overset{\mathsf{def}}{=} (A, U_{\mathsf{EEC}}(\underline{C}))$$

- this model does not support any real type-dependency

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathsf{Set}$, such that
  - $\mathcal{D}$ has set-indexed products and coproducts
- we use the families of sets fibration $p : \mathrm{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}$, where

$$p\big(f : X \longrightarrow Y, \{g_x : A(x) \longrightarrow (B \circ f)(x)\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we use the families fibration $q : \mathrm{Fam}(\mathcal{D}) \longrightarrow \mathsf{Set}$, where

$$q\big(f : X \longrightarrow Y, \{g_x : \underline{C}(x) \longrightarrow (\underline{D} \circ f)(x)\}_{x \in X}\big) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathcal{D}} \dashv U_{\mathcal{D}}$ as follows

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

- **ex:** EM-cats. ($\mathcal{D} \stackrel{\text{def}}{=} \mathsf{Set}^{\mathbf{T}}$) and Lawere ths. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Mod}(\mathcal{L}, \mathsf{Set})$)

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathrm{Set}$, such that
  - $\mathcal{D}$ has set-indexed products and coproducts
- we use the families of sets fibration $p : \mathrm{Fam}(\mathrm{Set}) \longrightarrow \mathrm{Set}$, where

$$p\big(f : X \longrightarrow Y, \big\{g_x : A(x) \longrightarrow (B \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we use the families fibration $q : \mathrm{Fam}(\mathcal{D}) \longrightarrow \mathrm{Set}$, where

$$q\big(f : X \longrightarrow Y, \big\{g_x : \underline{C}(x) \longrightarrow (\underline{D} \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathcal{D}} \dashv U_{\mathcal{D}}$ as follows

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

- **ex:** EM-cats. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Set}^{\mathbf{T}}$) and Lawere ths. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Mod}(\mathcal{L}, \mathrm{Set})$)

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathsf{Set}$, such that
  - $\mathcal{D}$ has set-indexed products and coproducts
- we use the families of sets fibration $p : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}$, where

$$p\big(f : X \longrightarrow Y, \{g_x : A(x) \longrightarrow (B \circ f)(x)\}_{x \in X}\big) \overset{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we use the families fibration $q : \mathsf{Fam}(\mathcal{D}) \longrightarrow \mathsf{Set}$, where

$$q\big(f : X \longrightarrow Y, \{g_x : \underline{C}(x) \longrightarrow (\underline{D} \circ f)(x)\}_{x \in X}\big) \overset{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathcal{D}} \dashv U_{\mathcal{D}}$ as follows

$$F(X, A) \overset{\text{def}}{=} (X, F_{\mathcal{D}} \circ A) \qquad U(X, \underline{C}) \overset{\text{def}}{=} (X, U_{\mathcal{D}} \circ \underline{C})$$

- **ex:** EM-cats. ($\mathcal{D} \overset{\text{def}}{=} \mathsf{Set}^{\mathbf{T}}$) and Lawere ths. ($\mathcal{D} \overset{\text{def}}{=} \mathsf{Mod}(\mathcal{L}, \mathsf{Set})$)

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathrm{Set}$, such that
  - $\mathcal{D}$ has set-indexed products and coproducts
- we use the families of sets fibration $p : \mathrm{Fam}(\mathrm{Set}) \longrightarrow \mathrm{Set}$, where

$$p\big(f : X \longrightarrow Y, \big\{g_x : A(x) \longrightarrow (B \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we use the families fibration $q : \mathrm{Fam}(\mathcal{D}) \longrightarrow \mathrm{Set}$, where

$$q\big(f : X \longrightarrow Y, \big\{g_x : \underline{C}(x) \longrightarrow (\underline{D} \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathcal{D}} \dashv U_{\mathcal{D}}$ as follows

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

- **ex:** EM-cats. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Set}^{\mathbf{T}}$) and Lawere ths. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Mod}(\mathcal{L}, \mathrm{Set})$)

Examples of fibred adjunction models

Example 3 (families fibrations):

- given an adjunction $F_{\mathcal{D}} \dashv U_{\mathcal{D}} : \mathcal{D} \longrightarrow \mathrm{Set}$, such that
  - $\mathcal{D}$ has set-indexed products and coproducts
- we use the families of sets fibration $p : \mathrm{Fam}(\mathrm{Set}) \longrightarrow \mathrm{Set}$, where

$$p\big(f : X \longrightarrow Y, \big\{g_x : A(x) \longrightarrow (B \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(Y, B)$$

- then, we use the families fibration $q : \mathrm{Fam}(\mathcal{D}) \longrightarrow \mathrm{Set}$, where

$$q\big(f : X \longrightarrow Y, \big\{g_x : \underline{C}(x) \longrightarrow (\underline{D} \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : q(X, \underline{C}) \longrightarrow q(Y, \underline{D})$$

- finally, we define $F \dashv U$ as the lifting of $F_{\mathcal{D}} \dashv U_{\mathcal{D}}$ as follows

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

- **ex:** EM-cats. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Set}^{\mathbf{T}}$) and Lawere ths. ($\mathcal{D} \stackrel{\text{def}}{=} \mathrm{Mod}(\mathcal{L}, \mathrm{Set})$)

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. ($\perp_A : 1 \longrightarrow TA$), and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- we use the continuous families fibration $p : \mathrm{CFam}(\mathrm{CPO}) \longrightarrow \mathrm{CPO}$

$$p\big(f : X \longrightarrow Y, \{g_x : A(x) \longrightarrow (B \circ f)(x)\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(X, B)$$

where

$$X \in \mathrm{CPO} \qquad A : X \longrightarrow \mathrm{CPO}^{\mathrm{EP}} \text{ an } \omega\text{-continuous functor}$$

- analogously, we use the cont. fam. fib. $q : \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \longrightarrow \mathrm{CPO}$
- finally, we define $F \dashv U$ as the lifting of $F^{\mathbf{T}} \dashv U^{\mathbf{T}}$
- **Ex.:** monads from discrete CPO-enriched countable Lawere ths.

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. ($\perp_A : 1 \longrightarrow TA$), and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- we use the continuous families fibration $p : \mathrm{CFam}(\mathrm{CPO}) \longrightarrow \mathrm{CPO}$

$$p\big(f : X \longrightarrow Y, \big\{g_x : A(x) \longrightarrow (B \circ f)(x)\big\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(X, B)$$

where

$$X \in \mathrm{CPO} \qquad A : X \longrightarrow \mathrm{CPO}^{\mathrm{EP}} \text{ an } \omega\text{-continuous functor}$$

- analogously, we use the cont. fam. fib. $q : \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \longrightarrow \mathrm{CPO}$
- finally, we define $F \dashv U$ as the lifting of $F^{\mathbf{T}} \dashv U^{\mathbf{T}}$
- **Ex.:** monads from discrete CPO-enriched countable Lawvere ths.

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. ($\perp_A : 1 \longrightarrow TA$), and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- we use the continuous families fibration $p : \mathrm{CFam}(\mathrm{CPO}) \longrightarrow \mathrm{CPO}$

$$p\big(f : X \longrightarrow Y, \{g_x : A(x) \longrightarrow (B \circ f)(x)\}_{x \in X}\big) \overset{\mathrm{def}}{=} f : p(X, A) \longrightarrow p(X, B)$$

where

$$X \in \mathrm{CPO} \qquad A : X \longrightarrow \mathrm{CPO}^{\mathrm{EP}} \text{ an } \omega\text{-continuous functor}$$

- analogously, we use the cont. fam. fib. $q : \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \longrightarrow \mathrm{CPO}$
- finally, we define $F \dashv U$ as the lifting of $F^{\mathbf{T}} \dashv U^{\mathbf{T}}$
- **Ex.:** monads from discrete CPO-enriched countable Lawvere ths.

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. $(\bot_A : 1 \longrightarrow TA)$, and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- we use the continuous families fibration $p : \mathrm{CFam}(\mathrm{CPO}) \longrightarrow \mathrm{CPO}$

$$p\big(f : X \longrightarrow Y, \{g_x : A(x) \longrightarrow (B \circ f)(x)\}_{x \in X}\big) \stackrel{\text{def}}{=} f : p(X, A) \longrightarrow p(X, B)$$

where

$$X \in \mathrm{CPO} \qquad A : X \longrightarrow \mathrm{CPO}^{\mathrm{EP}} \text{ an } \omega\text{-continuous functor}$$

- analogously, we use the cont. fam. fib. $q : \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \longrightarrow \mathrm{CPO}$
- finally, we define $F \dashv U$ as the lifting of $F^{\mathbf{T}} \dashv U^{\mathbf{T}}$
- **Ex.:** monads from discrete CPO-enriched countable Lawvere ths.

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. ($\bot_A : 1 \longrightarrow TA$), and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- we use the continuous families fibration $p : \mathrm{CFam}(\mathrm{CPO}) \longrightarrow \mathrm{CPO}$

$$p\big(f : X \longrightarrow Y, \big\{g_x : A(x) \longrightarrow (B \circ f)(x)\big\}_{x \in X}\big) \stackrel{\mathrm{def}}{=} f : p(X, A) \longrightarrow p(X, B)$$

where

$$X \in \mathrm{CPO} \qquad A : X \longrightarrow \mathrm{CPO}^{\mathrm{EP}} \text{ an } \omega\text{-continuous functor}$$

- analogously, we use the cont. fam. fib. $q : \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \longrightarrow \mathrm{CPO}$
- finally, we define $F \dashv U$ as the lifting of $F^{\mathbf{T}} \dashv U^{\mathbf{T}}$
- **Ex.:** monads from discrete CPO-enriched countable Lawere ths.

Examples of fibred adjunction models

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

- Why not use $p : \mathrm{CPO}^{\rightarrow} \longrightarrow \mathrm{CPO}$?
- **Theorem:** CPO is not locally cartesian closed!
  - **Idea:** Not all functors $f^* : \mathrm{CPO}/Y \to \mathrm{CPO}/X$ are left adjoints
  - consider the epimorphism $e \stackrel{\mathrm{def}}{=} n \mapsto n : \mathbb{N}_{=} \longrightarrow \mathbb{N}_{\omega}$ in CPO, and
  - assume given a non-empty cpo X, and
  - consider the constant function $f_{\omega} \stackrel{\mathrm{def}}{=} x \mapsto \omega : X \longrightarrow \mathbb{N}_{\omega}$,
  - then we have the following situation

$$\begin{tikzcd}
(\emptyset, =) \arrow[r, "f^*_{\omega}(e)"] \arrow[dr, bend right] & (\{\langle x, \omega\rangle \mid x \in |X|\}, \leq) \arrow[r, shift left, "g_1"] \arrow[r, shift right, "g_2"'] \arrow[d] & Y \arrow[dl, bend left, "h"] \\
& X &
\end{tikzcd}$$

  - thus f^*_{ω} does not preserve epis., and thus we don't have $f^*_{\omega} \dashv \Pi_{f_{\omega}}$

Examples of fibred adjunction models

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

- Why not use $p : \mathrm{CPO}^{\rightarrow} \longrightarrow \mathrm{CPO}$?
- **Theorem:** CPO is not locally cartesian closed!
  - **Idea:** Not all functors $f^* : \mathrm{CPO}/Y \to \mathrm{CPO}/X$ are left adjoints
  - consider the epimorphism $e \stackrel{\text{def}}{=} n \mapsto n : \mathbb{N}_{=} \longrightarrow \mathbb{N}_{\omega}$ in CPO, and
  - assume given a non-empty cpo X, and
  - consider the constant function $f_\omega \stackrel{\text{def}}{=} x \mapsto \omega : X \longrightarrow \mathbb{N}_\omega$,
  - then we have the following situation

$$\begin{tikzcd}
(\emptyset, =) \arrow[r, "f_\omega^*(e)"] \arrow[dr, bend right] & (\{\langle x, \omega\rangle \mid x \in |X|\}, \leq) \arrow[r, shift left, "g_1"] \arrow[r, shift right, "g_2"'] \arrow[d] & Y \arrow[dl, bend left, "h"] \\
 & X &
\end{tikzcd}$$

  - thus f_ω^* does not preserve epis., and thus we don't have $f_\omega^* \dashv \Pi_{f_\omega}$

Examples of fibred adjunction models

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

- Why not use $p : \mathrm{CPO}^{\to} \longrightarrow \mathrm{CPO}$?
- **Theorem:** CPO is not locally cartesian closed!
  - **Idea:** Not all functors $f^*\colon \mathrm{CPO}/Y \to \mathrm{CPO}/X$ are left adjoints
  - consider the epimorphism $e \stackrel{\text{def}}{=} n \mapsto n : \mathbb{N}_= \longrightarrow \mathbb{N}_\omega$ in CPO, and
  - assume given a non-empty cpo X, and
  - consider the constant function $f_\omega \stackrel{\text{def}}{=} x \mapsto \omega : X \longrightarrow \mathbb{N}_\omega$,
  - then we have the following situation

$$\begin{tikzcd}
(\emptyset, =) \arrow[r, "f^*_\omega(e)"] \arrow[dr, bend right] & (\{\langle x, \omega\rangle \mid x \in |X|\}, \leq) \arrow[r, shift left, "g_1"] \arrow[r, shift right, "g_2"'] \arrow[d] & Y \arrow[dl, bend left, "h"] \\
& X &
\end{tikzcd}$$

  - thus f^*_ω does not preserve epis., and thus we don't have $f^*_\omega \dashv \Pi_{f_\omega}$

Examples of fibred adjunction models

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

- Why not use $p : \mathrm{CPO}^{\rightarrow} \longrightarrow \mathrm{CPO}$?
- **Theorem:** CPO is not locally cartesian closed!
  - **Idea:** Not all functors $f^* : \mathrm{CPO}/Y \to \mathrm{CPO}/X$ are left adjoints
  - consider the epimorphism $e \stackrel{\mathrm{def}}{=} n \mapsto n : \mathbb{N}_{=} \longrightarrow \mathbb{N}_{\omega}$ in CPO, and
  - assume given a non-empty cpo X, and
  - consider the constant function $f_\omega \stackrel{\mathrm{def}}{=} x \mapsto \omega : X \longrightarrow \mathbb{N}_\omega$,
  - then we have the following situation

$$\begin{tikzcd}
(\emptyset, =) \arrow[r, "f_\omega^*(e)"] \arrow[dr, bend right] & (\{\langle x, \omega\rangle \mid x \in |X|\}, \leq) \arrow[r, shift left, "g_1"] \arrow[r, shift right, "g_2"'] \arrow[d] & Y \arrow[dl, bend left, "h"] \\
& X &
\end{tikzcd}$$

  - thus f_ω^* does not preserve epis., and thus we don't have $f_\omega^* \dashv \Pi_{f_\omega}$

Examples of fibred adjunction models

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

- Why not use $p : \mathrm{CPO}^{\to} \longrightarrow \mathrm{CPO}$?
- **Theorem:** CPO is not locally cartesian closed!
  - **Idea:** Not all functors $f^*\!:\!\mathrm{CPO}/Y \to \mathrm{CPO}/X$ are left adjoints
  - consider the epimorphism $e \stackrel{\text{def}}{=} n \mapsto n : \mathbb{N}_{=} \longrightarrow \mathbb{N}_\omega$ in CPO, and
  - assume given a non-empty cpo X, and
  - consider the constant function $f_\omega \stackrel{\text{def}}{=} x \mapsto \omega : X \longrightarrow \mathbb{N}_\omega$,
  - then we have the following situation

$$\begin{tikzcd}
(\emptyset, =) \arrow[r, "f_\omega^*(e)"] \arrow[dr, bend right] & (\{\langle x, \omega\rangle \mid x \in |X|\}, \leq) \arrow[r, shift left, "g_1"] \arrow[r, shift right, "g_2"'] \arrow[d] & Y \arrow[dl, bend left, "h"] \\
& X &
\end{tikzcd}$$

  - thus f_ω^* does not preserve epis., and thus we don't have $f_\omega^* \dashv \Pi_{f_\omega}$

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.,

$$
\begin{tikzcd}
\mathcal{V} \arrow[rr, "T"] \arrow[dr, "p"'] & & \mathcal{V} \arrow[dl, "p"] \\
& \mathcal{B} &
\end{tikzcd}
\qquad \text{and} \qquad p(\eta_A) = \mathrm{id}_{p(A)} \qquad p(\mu_A) = \mathrm{id}_{p(A)}
$$

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

$$
\begin{tikzcd}
\mathcal{V} \arrow[rr, bend left, "F^{\mathbf{T}}"] \arrow[rr, phantom, "\perp"] \arrow[dr, "p"'] & & \mathcal{V}^{\mathbf{T}} \arrow[ll, bend left, "U^{\mathbf{T}}"] \arrow[dl, "p^{\mathbf{T}}"] \\
& \mathcal{B} &
\end{tikzcd}
$$

- 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

- **Theorem 2:** if p supports Σ-types and the dependent strength

$$\sigma_A : \Sigma_A \circ T \longrightarrow T \circ \Sigma_A$$

  is a natural isomorphism, then $p^{\mathbf{T}}$ also supports Σ-types

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

  coequalizers, then $p^{\mathbf{T}}$ also supports Σ-types

(for corresponding simply typed results, see [Borceux'94] and [Linton'69])

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

- **Theorem 2:** if p supports Σ-types and the dependent strength

$$\sigma_A : \Sigma_A \circ T \longrightarrow T \circ \Sigma_A$$

  is a natural isomorphism, then $p^{\mathbf{T}}$ also supports Σ-types

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

  coequalizers, then $p^{\mathbf{T}}$ also supports Σ-types

(for corresponding simply typed results, see [Borceux'94] and [Linton'69])

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

- **Theorem 2:** if p supports Σ-types and the dependent strength

  $$\sigma_A : \Sigma_A \circ T \longrightarrow T \circ \Sigma_A$$

  is a natural isomorphism, then $p^{\mathbf{T}}$ also supports Σ-types

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

  coequalizers, then $p^{\mathbf{T}}$ also supports Σ-types

(for corresponding simply typed results, see [Borceux'94] and [Linton'69])

Algebraic effects

Algebraic effects

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

- we consider signatures of dep. typed operation symbols

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

- equipped with equations on derivable effect terms

Typing rule:

$$\frac{\Gamma \vdash_{\mathsf{v}} V : I \quad \Gamma \vdash \underline{C} \quad \Gamma, x : O[V/x_i] \vdash_{\mathsf{c}} M : \underline{C}}{\Gamma \vdash_{\mathsf{c}} \mathsf{op}^{\underline{C}}_V(x.M) : \underline{C}} \; (\mathsf{op} : (x_i : I) \longrightarrow O)$$

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

$$\frac{\Gamma \vdash_{\mathsf{v}} V : I \quad \Gamma, x : O[V/x_i] \vdash_{\mathsf{c}} M : \underline{C} \quad \Gamma \,|\, z : \underline{C} \vdash_{\mathsf{h}} K : \underline{D}}{\Gamma \vdash_{\mathsf{c}} K[\mathsf{op}^{\underline{C}}_V(x.M)/z] = \mathsf{op}^{\underline{D}}_V(x.K[M/z]) : \underline{D}} \; (\mathsf{op} : (x_i : I) \longrightarrow O)$$

Semantics: based on $q : \mathrm{Fam}(\mathrm{Mod}(\mathcal{L}_{\mathcal{T}_{\mathsf{eff}}}, \mathrm{Set})) \longrightarrow \mathrm{Set}$

Algebraic effects

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

- we consider signatures of dep. typed operation symbols

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

- equipped with equations on derivable effect terms

Typing rule:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I \quad \Gamma \vdash \underline{C} \quad \Gamma, x : O[V/x_i] \vDash_{\mathsf{c}} M : \underline{C}}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}^{\underline{C}}_{V}(x.M) : \underline{C}} \; (\mathsf{op} : (x_i : I) \longrightarrow O)$$

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

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I \quad \Gamma, x : O[V/x_i] \vDash_{\mathsf{c}} M : \underline{C} \quad \Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D}}{\Gamma \vDash_{\mathsf{c}} K[\mathsf{op}^{\underline{C}}_{V}(x.M)/z] = \mathsf{op}^{\underline{D}}_{V}(x.K[M/z]) : \underline{D}} \; (\mathsf{op} : (x_i : I) \longrightarrow O)$$

Semantics: based on $q : \mathsf{Fam}(\mathsf{Mod}(\mathcal{L}_{\mathcal{T}_{\mathsf{eff}}}, \mathsf{Set})) \longrightarrow \mathsf{Set}$

Algebraic effects

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

- we consider signatures of dep. typed operation symbols

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

- equipped with equations on derivable effect terms

Typing rule:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I \quad \Gamma \vdash \underline{C} \quad \Gamma, x : O[V/x_i] \vDash_{\mathsf{c}} M : \underline{C}}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}^{\underline{C}}_{V}(x.M) : \underline{C}} \ (\mathsf{op} : (x_i : I) \longrightarrow O)$$

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

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I \quad \Gamma, x : O[V/x_i] \vDash_{\mathsf{c}} M : \underline{C} \quad \Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D}}{\Gamma \vDash_{\mathsf{c}} K[\mathsf{op}^{\underline{C}}_{V}(x.M)/z] = \mathsf{op}^{\underline{D}}_{V}\big(x.K[M/z]\big) : \underline{D}} \ (\mathsf{op} : (x_i : I) \longrightarrow O)$$

Semantics: based on $q : \mathrm{Fam}(\mathrm{Mod}(\mathcal{L}_{\mathcal{T}_{\mathrm{eff}}}, \mathrm{Set})) \longrightarrow \mathrm{Set}$

Algebraic effects

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

- we consider signatures of dep. typed operation symbols

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

- equipped with equations on derivable effect terms

Typing rule:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I \quad \Gamma \vdash \underline{C} \quad \Gamma, x : O[V/x_i] \vDash_{\mathsf{c}} M : \underline{C}}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}^{\underline{C}}_{V}(x.M) : \underline{C}} \; (\mathsf{op} : (x_i : I) \longrightarrow O)$$

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

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I \quad \Gamma, x : O[V/x_i] \vDash_{\mathsf{c}} M : \underline{C} \quad \Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D}}{\Gamma \vDash_{\mathsf{c}} K[\mathsf{op}^{\underline{C}}_{V}(x.M)/z] = \mathsf{op}^{\underline{D}}_{V}(x.K[M/z]) : \underline{D}} \; (\mathsf{op} : (x_i : I) \longrightarrow O)$$

Semantics: based on $q : \mathrm{Fam}(\mathrm{Mod}(\mathcal{L}_{\mathcal{T}_{\text{eff}}}, \mathrm{Set})) \longrightarrow \mathrm{Set}$

Algebraic effects

Example 1 (interactive I/O):

- $\mathsf{read} : 1 \longrightarrow \mathsf{Chr}$ $(\mathsf{Chr} \stackrel{\mathsf{def}}{=} 1 + \ldots + 1)$

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

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

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

  $x : \mathsf{Loc} \vdash \mathsf{Val}$

  $\diamond \models_{\mathsf{v}} \mathsf{isDec}_{\mathsf{Loc}} : \Pi x : \mathsf{Loc} \,.\, \Pi y : \mathsf{Loc} \,.\, (x =_{\mathsf{Loc}} y) + (x =_{\mathsf{Loc}} y \to 0)$
- $\mathsf{get} : (x : \mathsf{Loc}) \longrightarrow \mathsf{Val}$

  $\mathsf{put} : (\Sigma x : \mathsf{Loc}. \mathsf{Val}) \longrightarrow 1$
- five equations (two of them conditional on $\mathsf{isDec}_{\mathsf{Loc}}$)

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

Algebraic effects

Example 1 (interactive I/O):

- $\mathsf{read} : 1 \longrightarrow \mathsf{Chr}$ $\qquad$ $(\mathsf{Chr} \stackrel{\text{def}}{=} 1 + \ldots + 1)$

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

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

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

  $x : \mathsf{Loc} \vdash \mathsf{Val}$

  $\diamond \models_{\mathrm{v}} \mathsf{isDec}_{\mathsf{Loc}} : \Pi x : \mathsf{Loc} \,.\, \Pi y : \mathsf{Loc} \,.\, (x =_{\mathsf{Loc}} y) + (x =_{\mathsf{Loc}} y \to 0)$
- $\mathsf{get} : (x : \mathsf{Loc}) \longrightarrow \mathsf{Val}$

  $\mathsf{put} : (\Sigma x : \mathsf{Loc}.\mathsf{Val}) \longrightarrow 1$
- five equations (two of them conditional on $\mathsf{isDec}_{\mathsf{Loc}}$)

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

Algebraic effects

Example 1 (interactive I/O):

- read : $1 \longrightarrow \mathsf{Chr}$ $\quad$ $(\mathsf{Chr} \stackrel{\text{def}}{=} 1 + \ldots + 1)$

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

- no equations

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

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

  $x\,{:}\,\mathsf{Loc} \vdash \mathsf{Val}$

  $\diamond \vDash_{\mathsf{v}} \mathsf{isDec}_{\mathsf{Loc}} : \Pi x\,{:}\,\mathsf{Loc}\,.\,\Pi y\,{:}\,\mathsf{Loc}\,.\,(x =_{\mathsf{Loc}} y) + (x =_{\mathsf{Loc}} y \to 0)$

- get : $(x\,{:}\,\mathsf{Loc}) \longrightarrow \mathsf{Val}$

  put : $(\Sigma x\,{:}\,\mathsf{Loc}.\mathsf{Val}) \longrightarrow 1$

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

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

Handlers of algebraic effects

(for programming and reasoning)

Handlers of alg. effects (for programming)

Idea: Generalisation of exception handlers to all algebraic effects

[Plotkin,Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

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

satisfying

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

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

Typical use case:

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

Handlers of alg. effects (for programming)

Idea: Generalisation of exception handlers to all algebraic effects

[Plotkin,Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

$$\frac{\Gamma \vDash_{\mathsf{c}} M : FA \quad \Gamma, x_v : I, x_k : O[x_v/x_i] \to U\underline{C} \vDash_{\mathsf{c}} N_{\mathrm{op}} : \underline{C} \quad \Gamma, y : A \vDash_{\mathsf{c}} N_{\mathrm{ret}} : \underline{C}}{\Gamma \vDash_{\mathsf{c}} M \;\mathtt{handled\ with}\; \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{T}_{\mathrm{eff}}} \;\mathtt{to}\; y : A \;\mathtt{in}_{\underline{C}}\; N_{\mathrm{ret}} : \underline{C}}$$

satisfying

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

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

Typical use case:

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

Handlers of alg. effects (for programming)

Idea: Generalisation of exception handlers to all algebraic effects

[Plotkin,Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

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

satisfying

$$(\texttt{return}\, V) \;\texttt{handled with}\; \{\ldots\}_{\mathrm{op} \in \mathcal{T}_{\mathrm{eff}}} \;\texttt{to}\; y : A \;\texttt{in}\; N_{\mathrm{ret}} = N_{\mathrm{ret}}[V/x]$$

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

Typical use case:

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

Handlers of alg. effects (for reasoning)

Idea: Assuming we were able to handle into values

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

we could define predicates (value terms)

$$\Gamma \vDash_{\!\text{v}} P : UFA \to \mathcal{U}$$

by

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

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

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

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

Handlers of alg. effects (for reasoning)

Idea: Assuming we were able to handle into values

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

we could define predicates (value terms)

$$\Gamma \vdash_{\mathrm{v}} P : UFA \to \mathcal{U}$$

by

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

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

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

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

Handlers of alg. effects (for reasoning)

Idea: Assuming we were able to handle into values

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

we could define predicates (value terms)

$$\Gamma \vDash_{\mathsf{v}} P : UFA \to \mathcal{U}$$

by

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

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

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

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

Handlers of alg. effects (for reasoning)

Idea: Assuming we were able to handle into values

$$M \;\texttt{handled with}\; \{\mathtt{op}_{x_v}(x_k) \mapsto V_{\mathtt{op}}\}_{\mathtt{op} \in \mathcal{T}_{\mathsf{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{B}\; V_{\mathsf{ret}}$$

we could define predicates (value terms)

$$\Gamma \vDash_{\mathsf{v}} P : UFA \to \mathcal{U}$$

by

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

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

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

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

Handlers of alg. effects (for reasoning)

Example 1 (Evaluation logic style modalities):

- Given a predicate $P : A \to \mathcal{U}$ on return values,

  we define a predicate $\Diamond P : UFA \to \mathcal{U}$ on I/O-computations as

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

  using the handler given by

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

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

- $\Diamond P$ is the possibility modality

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

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

Handlers of alg. effects (for reasoning)

Example 1 (Evaluation logic style modalities):

- Given a predicate $P : A \to \mathcal{U}$ on return values,

  we define a predicate $\Diamond P : UFA \to \mathcal{U}$ on I/O-computations as

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

  using the handler given by

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

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

- $\Diamond P$ is the possibility modality

$$\Diamond P\ (\mathtt{thunk}\,(\mathtt{read}(x \,.\, \mathtt{write}_{e'}(\mathtt{return}\, V)))) = \widehat{\Sigma}\, x \!:\! \mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, P\ V$$

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

Handlers of alg. effects (for reasoning)

Example 1 (Evaluation logic style modalities):

- Given a predicate $P : A \to \mathcal{U}$ on return values,

  we define a predicate $\Diamond P : UFA \to \mathcal{U}$ on I/O-computations as

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

  using the handler given by

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

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

- $\Diamond P$ is the possibility modality

$$\Diamond P\,\big(\mathtt{thunk}\,(\mathtt{read}(x\,.\,\mathtt{write}_{\prime e^{\prime}}(\mathtt{return}\; V)))\big) = \widehat{\Sigma}\, x \,{:}\, \mathsf{El}(\widehat{\mathsf{Chr}})\,.\,P\;\; V$$

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

Handlers of alg. effects (for reasoning)

Example 1 (Evaluation logic style modalities):

- Given a predicate $P : A \to \mathcal{U}$ on return values,

  we define a predicate $\Diamond P : UFA \to \mathcal{U}$ on I/O-computations as

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

  using the handler given by

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

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

- $\Diamond P$ is the possibility modality

$$\Diamond P\, \big(\mathtt{thunk}\, (\mathtt{read}(x \,.\, \mathtt{write}_{'e'}(\mathtt{return}\, V)))\big) = \widehat{\Sigma}\, x \!:\! \mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, P\ V$$

- To get the necessity modality, use $\widehat{\Pi}\, x \!:\! \mathsf{El}(\widehat{\mathsf{Chr}})$ 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

  $$Q : A \to S \to \mathcal{U} \qquad\qquad (S \stackrel{\text{def}}{=} \Pi x{:}\mathrm{Loc}\,.\,\mathrm{Val})$$

  we define a precondition for stateful comps. on initial states

  $$\mathrm{wp}_Q : UFA \to S \to \mathcal{U}$$

  by

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

  $$V_{\mathrm{get}}, V_{\mathrm{put}} \quad \text{on} \quad S \to (\mathcal{U} \times S) \qquad \text{and} \qquad V_{\mathrm{ret}} \text{ ``}=\text{'' } Q$$

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

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

  $$\mathrm{wp}_Q\,(\mathtt{thunk}\,(\mathtt{return}\,V)) \quad = \quad \lambda\, x_S {:} S\,.\,Q\ V\ x_S$$

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

Handlers of alg. effects (for reasoning)

Example 2 (Dijkstra's weakest precondition semantics):

- Given a postcondition on return values and final states

$$Q : A \to S \to \mathcal{U} \qquad\qquad (S \stackrel{\text{def}}{=} \Pi x : \mathsf{Loc} \,.\, \mathsf{Val})$$

  we define a precondition for stateful comps. on initial states

$$\mathsf{wp}_Q : UFA \to S \to \mathcal{U}$$

  by

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

$$V_{\mathrm{get}}, V_{\mathrm{put}} \quad \text{on} \quad S \to (\mathcal{U} \times S) \qquad \text{and} \qquad V_{\mathrm{ret}} \text{ ``}={}\text{''}\ Q$$

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

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

$$\mathsf{wp}_Q\,(\mathtt{thunk}\,(\mathtt{return}\,V)) \quad = \quad \lambda\, x_S : S \,.\, Q\ V\ x_S$$

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

Handlers of alg. effects (for reasoning)

Example 2 (Dijkstra's weakest precondition semantics):

- Given a postcondition on return values and final states

$$Q : A \to S \to \mathcal{U} \qquad\qquad (S \stackrel{\text{def}}{=} \Pi x : \mathsf{Loc} \,.\, \mathsf{Val})$$

  we define a precondition for stateful comps. on initial states

$$\mathsf{wp}_Q : UFA \to S \to \mathcal{U}$$

  by

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

$$V_{\mathrm{get}}, V_{\mathrm{put}} \quad \text{on} \quad S \to (\mathcal{U} \times S) \qquad \text{and} \qquad V_{\mathrm{ret}} \text{ ``} = \text{''}\ Q$$

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

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

$$\mathsf{wp}_Q\left(\mathtt{thunk}\left(\mathtt{return}\,V\right)\right) \;=\; \lambda\, x_S : S \,.\, Q\ V\ x_S$$

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

Handlers of alg. effects (for reasoning)

Example 3 (Patterns of allowed effects):

- Assuming an inductive type Protocol, given by

$$\mathtt{e} : \mathsf{Protocol} \qquad \mathtt{r} : (\mathsf{Chr} \to \mathsf{Protocol}) \to \mathsf{Protocol}$$

$$\mathtt{w} : (\mathsf{Chr} \to \mathcal{U}) \to \mathsf{Protocol} \to \mathsf{Protocol}$$

  and potentially also by $\wedge$, $\vee$, ...

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

$$\underline{\mathsf{Pr}} : UFA \to \mathcal{U}$$

  by handling the given comp. using

$$V_{\mathsf{read}}\,,\, V_{\mathsf{write}} \qquad \text{on} \qquad \mathsf{Protocol} \to \mathcal{U}$$

  where

$$\begin{aligned} V_{\mathsf{read}}\ \langle \_\,, V_{\mathsf{rk}} \rangle\ (\mathtt{r}\ \mathsf{Pr}') &\stackrel{\text{def}}{=} \widehat{\Pi}\, x \mathbin{:} \mathsf{El}(\widehat{\mathsf{Chr}})\,.\,(V_{\mathsf{rk}}\ x)\,(\mathsf{Pr}'\ x) \\ V_{\mathsf{write}}\ \langle V\,, V_{\mathsf{wk}} \rangle\ (\mathtt{w}\ P\ \mathsf{Pr}') &\stackrel{\text{def}}{=} \widehat{\Sigma}\, x \mathbin{:} \mathsf{El}(P\ V)\,.\,V_{\mathsf{wk}}\ \star\ \mathsf{Pr}' \\ \_ &\stackrel{\text{def}}{=} \widehat{0} \end{aligned}$$

Handlers of alg. effects (for reasoning)

Example 3 (Patterns of allowed effects):

- Assuming an inductive type Protocol, given by

$$\mathtt{e} : \mathsf{Protocol} \qquad \mathtt{r} : (\mathsf{Chr} \to \mathsf{Protocol}) \to \mathsf{Protocol}$$

$$\mathtt{w} : (\mathsf{Chr} \to \mathcal{U}) \to \mathsf{Protocol} \to \mathsf{Protocol}$$

  and potentially also by $\wedge$, $\vee$, ...

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

$$\underline{\mathsf{Pr}} : \mathit{UFA} \to \mathcal{U}$$

  by handling the given comp. using

$$V_{\mathsf{read}}, V_{\mathsf{write}} \qquad \text{on} \qquad \mathsf{Protocol} \to \mathcal{U}$$

  where

$$\begin{aligned} V_{\mathsf{read}}\ \langle \_ , V_{\mathsf{rk}} \rangle\ (\mathtt{r}\ \mathsf{Pr}') &\stackrel{\mathsf{def}}{=} \widehat{\Pi}\, x{:}\mathsf{El}(\widehat{\mathsf{Chr}}) . (V_{\mathsf{rk}}\ x)\,(\mathsf{Pr}'\ x) \\ V_{\mathsf{write}}\ \langle V , V_{\mathsf{wk}} \rangle\ (\mathtt{w}\ P\ \mathsf{Pr}') &\stackrel{\mathsf{def}}{=} \widehat{\Sigma}\, x{:}\mathsf{El}(P\ V) . V_{\mathsf{wk}} \star \mathsf{Pr}' \\ \_ &\stackrel{\mathsf{def}}{=} \widehat{0} \end{aligned}$$

Handlers of alg. effects (for reasoning)

Example 3 (Patterns of allowed effects):

- Assuming an inductive type Protocol, given by

$$\mathtt{e} : \mathsf{Protocol} \qquad \mathtt{r} : (\mathsf{Chr} \to \mathsf{Protocol}) \to \mathsf{Protocol}$$

$$\mathtt{w} : (\mathsf{Chr} \to \mathcal{U}) \to \mathsf{Protocol} \to \mathsf{Protocol}$$

  and potentially also by $\wedge$, $\vee$, ...

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

$$\underline{\mathsf{Pr}} : UFA \to \mathcal{U}$$

  by handling the given comp. using

$$V_{\mathsf{read}}, V_{\mathsf{write}} \qquad \text{on} \qquad \mathsf{Protocol} \to \mathcal{U}$$

  where

$$\begin{array}{lcl} V_{\mathsf{read}} \ \langle \_ , V_{\mathsf{rk}} \rangle \ \ (\mathtt{r}\ \mathsf{Pr}') & \stackrel{\mathsf{def}}{=} & \widehat{\Pi}\, x : \mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, (V_{\mathsf{rk}}\ x)\,(\mathsf{Pr}'\ x) \\ V_{\mathsf{write}} \ \langle V , V_{\mathsf{wk}} \rangle \ (\mathtt{w}\ P\ \mathsf{Pr}') & \stackrel{\mathsf{def}}{=} & \widehat{\Sigma}\, x : \mathsf{El}(P\ \ V) \,.\, V_{\mathsf{wk}}\ \star\ \mathsf{Pr}' \\ \_ & \stackrel{\mathsf{def}}{=} & \widehat{0} \end{array}$$

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
- algebraic effects and their handlers (also for reasoning)

Future work: type-dep. on computations, i.e., $\underline{D}(\mathtt{read}(x.M))$

- how to avoid restricting the typing of seq. comp.?
- $M \mathrel{\mathtt{to}} x{:}A \mathrel{\mathtt{in}} N : \underline{C}[\mathtt{thunk}\, M/y]$ (where $y : UFA$) [Vákár'17]
- $\alpha : T(\mathcal{U}) \longrightarrow \mathcal{U}$ [Pédrot,Tabareau'17]
- for eMLTT, two obvious ways forward
  - **i)** Vákár's approach but with eq. hom. proof obligations; or
  - **ii)** $\underline{D}[M \mathrel{\mathtt{to}} x{:}A \mathrel{\mathtt{in}} N/z] = M \mathrel{\mathtt{to}} x{:}A \mathrel{\mathtt{in}} \underline{D}[N/z]$

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
- algebraic effects and their handlers (also for reasoning)

Future work: type-dep. on computations, i.e., $\underline{D}(\mathtt{read}(x.M))$

- how to avoid restricting the typing of seq. comp.?
- $M \ \mathtt{to}\ x\!:\!A \ \mathtt{in}\ N : \underline{C}[\mathtt{thunk}\, M/y]$ (where $y\!:\!UFA$) [Vákár'17]
- $\alpha : T(\mathcal{U}) \longrightarrow \mathcal{U}$ [Pédrot,Tabareau'17]
- for eMLTT, two obvious ways forward
  - **i)** Vákár's approach but with eq. hom. proof obligations; or
  - **ii)** $\underline{D}[M \ \mathtt{to}\ x\!:\!A \ \mathtt{in}\ N/z] = M \ \mathtt{to}\ x\!:\!A \ \mathtt{in}\ \underline{D}[N/z]$

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)