A fibrational view on computational effects

Danel Ahman

Prosecco Team, Inria Paris

Copenhagen, 13 November 2017

Outline

We investigate the combination of

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

We investigate the combination of

- 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 (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 $(\Pi, \Sigma, V =_A W, \ldots)$
- 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 (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 ($\Pi, \Sigma, V =_A W, \ldots$)
- 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 (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 $(\Pi, \Sigma, V =_A W, \ldots)$
- 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 (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.:

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

which could be used to type the dependent function

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

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

A1: In this talk, we say no

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

A2: But we are also looking into yes

- type-dependency needs to be "homomorphic", but not only so
- intuitively, need to lift $A(\ell)$ to $A^{\dagger}(c)$, where $c : T(\mathsf{List\ Chr})$

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 $A[\mathtt{receive}(y.\, M)/\ell\,]$?

A1: In this talk, we say no

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

A2: But we are also looking into yes

- type-dependency needs to be "homomorphic", but not only so
- intuitively, need to lift $A(\ell)$ to $A^{\dagger}(c)$, where $c : T(\mathsf{List\ Chr})$

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 $A[\mathtt{receive}(y.\, M)/\ell\,]$?

A1: In this talk, we say no

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

A2: But we are also looking into yes

- type-dependency needs to be "homomorphic", but not only so
- intuitively, need to lift $A(\ell)$ to $A^{\dagger}(c)$, where $c : T(\mathsf{List\ Chr})$

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 $A[\mathtt{receive}(y.\, M)/\ell]$?

A1: In this talk, we say no

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

A2: But we are also looking into yes

- type-dependency needs to be "homomorphic", but not only so
- intuitively, need to lift $A(\ell)$ to $A^{\dagger}(c)$, where $c : T(\mathsf{List\ Chr})$

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 $x_1 : A_1, \ldots, x_n : A_n$

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 $x_1 : A_1, \ldots, x_n : A_n$

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 $x_1 : A_1, \ldots, x_n : A_n$

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

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

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 $\underline{C}$: [Levy'04]

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

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

- if we consider

$$\mathtt{fopen}\,(\mathtt{return\ true}, \mathtt{return\ false})\ \mathtt{to}\ x\!:\!\mathsf{Bool}\ \mathtt{in}\ N$$

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

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

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

Assigning types to effectful programs

Aim: To fix the typing rule of sequential composition

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

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

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

- if we consider

$$\mathtt{fopen}\,(\mathtt{return\ true}, \mathtt{return\ false})\ \mathtt{to}\ x\!:\!\mathsf{Bool}\ \mathtt{in}\ N$$

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

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

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

Assigning types to effectful programs

Aim: To fix the typing rule of sequential composition

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

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

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

- if we consider

$$\mathtt{fopen}\left(\mathtt{return\ true}, \mathtt{return\ false}\right)\ \mathtt{to}\ x\!:\!\mathsf{Bool}\ \mathtt{in}\ N$$

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

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

(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

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

But then comp. types would be singleton-like!?!

However, something like this is probably needed for the yes case.

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

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

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

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

But then comp. types would be singleton-like!?!

However, something like this is probably needed for the yes case.

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

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

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

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

But then comp. types would be singleton-like!?!

However, something like this is probably needed for the yes case.

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

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

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

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

After tossing the coin, this program evaluates as either

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

Idea: M denotes an element of the coproduct of algebras

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

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

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

After tossing the coin, this program evaluates as either

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

Idea: M denotes an element of the coproduct of algebras

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

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

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

After tossing the coin, this program evaluates as either

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

Idea: M denotes an element of the coproduct of algebras

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

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

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

After tossing the coin, this program evaluates as either

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

Idea: M denotes an element of the coproduct of algebras

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

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

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

After tossing the coin, this program evaluates as either

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

Idea: M denotes an element of the coproduct of algebras

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

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 – 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 – 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 intensional MLTT

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

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

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

eMLTT – terms

Value terms: MLTT + thunks + ...

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

- equational theory based on intensional MLTT

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

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

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 intensional MLTT

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

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

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

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

Typing judgments:

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

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

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

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

Typing judgments:

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

eMLTT – typing sequential composition

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

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

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

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

Categorical semantics of eMLTT

(fibrations + adjunctions)

Categorical semantics – MLTT part

We define fibred adjunction models

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

given by: i) a split closed comprehension cat. p with Nat, ...

$$\begin{tikzcd} \mathcal{V} \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] \arrow[d, phantom, shift right=3ex, "\dashv"] \arrow[d, phantom, shift left=3ex, "\dashv"] \\ \mathcal{B} \arrow[u, "1"'] \end{tikzcd}$$

- 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

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

given by: i) a split closed comprehension cat. p with Nat, ...

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

- 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

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

given by: i) a split closed comprehension cat. p with Nat, ...

$$\begin{tikzcd}
\mathcal{V} \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] \\
\mathcal{B} \arrow[u, "1" description] \arrow[u, phantom, shift right=4ex, "\dashv"] \arrow[u, phantom, shift left=4ex, "\dashv"]
\end{tikzcd}$$

- 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

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

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

given by: ii) a split fibration q (with ...) and a s. fib. adj. $F \dashv U$

$$\begin{tikzcd}
\mathcal{V} \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] & \mathcal{C} \arrow[l, bend left, "U"] \arrow[dl, bend left, "q"] \\
\mathcal{B} \arrow[u, "1" description] \arrow[u, phantom, shift right=4ex, "\dashv"] \arrow[u, phantom, shift left=4ex, "\dashv"] &
\end{tikzcd}$$

- 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

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

given by: ii) a split fibration q (with ...) and a s. fib. adj. $F \dashv U$

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

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

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

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

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

Digression: dep. elimination of colimits

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 colimits

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

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

Examples of fibred adjunction models

Example 1 (identity adjunctions):

$$\begin{tikzcd} \mathcal{V} \arrow[rr, bend left, "\mathsf{Id}_{\mathcal{V}}"] \arrow[rr, phantom, "\perp"] \arrow[dr, "p"'] & & \mathcal{V} \arrow[ll, bend left, "\mathsf{Id}_{\mathcal{V}}"] \arrow[dl, "p"] \\ & \mathcal{B} & \end{tikzcd}$$

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 $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is Cartesian closed (with Nat, ...), and
  - $\mathcal{E}$ and $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ 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 simple fibration $\mathsf{s}_{\mathcal{D}}$ and simpl. $\mathcal{D}$-enrich. fibration $\mathsf{s}_{\mathcal{D},\mathcal{E}}$

$$\begin{tikzcd}
\mathsf{s}(\mathcal{D}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathsf{s}_{\mathcal{D}}"'] & & \mathsf{s}(\mathcal{D},\mathcal{E}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathsf{s}_{\mathcal{D},\mathcal{E}}"] \\
& \mathcal{D} &
\end{tikzcd}$$

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

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

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

$\mathsf{s}(\mathcal{D},\mathcal{E})$: $\quad (f, h) : (X, \underline{C}) \longrightarrow (Y, \underline{D}) \quad$ 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 $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is Cartesian closed (with Nat, ...), and
  - $\mathcal{E}$ and $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ 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 simple fibration $\mathsf{s}_{\mathcal{D}}$ and simpl. $\mathcal{D}$-enrich. fibration $\mathsf{s}_{\mathcal{D},\mathcal{E}}$

$$\begin{tikzcd}
\mathsf{s}(\mathcal{D}) \arrow[rr, bend left, "F"] \arrow[dr, "\mathsf{s}_{\mathcal{D}}"'] \arrow[rr, phantom, "\perp"] & & \mathsf{s}(\mathcal{D},\mathcal{E}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathsf{s}_{\mathcal{D},\mathcal{E}}"] \\
& \mathcal{D} &
\end{tikzcd}$$

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

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

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

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

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 $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathcal{E} \longrightarrow \mathcal{D}$, such that
  - $\mathcal{D}$ is Cartesian closed (with Nat, ...), and
  - $\mathcal{E}$ and $F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}}$ 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 simple fibration $\mathsf{s}_{\mathcal{D}}$ and simpl. $\mathcal{D}$-enrich. fibration $\mathsf{s}_{\mathcal{D},\mathcal{E}}$

$$\begin{tikzcd} \mathsf{s}(\mathcal{D}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathsf{s}_{\mathcal{D}}"'] & & \mathsf{s}(\mathcal{D},\mathcal{E}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathsf{s}_{\mathcal{D},\mathcal{E}}"] \\ & \mathcal{D} & \end{tikzcd}$$

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

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

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

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

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

Examples of fibred adjunction models

Example 3 (families fibrations):

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

$$
\begin{tikzcd}
\mathsf{Fam}(\mathsf{Set}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathsf{fam}_{\mathsf{Set}}"'] & & \mathsf{Fam}(\mathcal{D}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathsf{fam}_{\mathcal{D}}"] \\
& \mathsf{Set} &
\end{tikzcd}
$$

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

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

$\mathsf{Fam}(\mathsf{Set})$: (X, A) where $X \in \mathsf{Set}$ $A : X \longrightarrow \mathsf{Set}$

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

Examples of fibred adjunction models

Example 3 (families fibrations):

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

$$\begin{tikzcd}
\mathrm{Fam}(\mathsf{Set}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathsf{fam}_{\mathsf{Set}}"'] & & \mathrm{Fam}(\mathcal{D}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathsf{fam}_{\mathcal{D}}"] \\
& \mathsf{Set} &
\end{tikzcd}$$

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

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

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

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

Examples of fibred adjunction models

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

- 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
- such $\mathbf{T}$ arise from discrete CPO-enriched countable Law. ths.

- we use continuous families fibrations $\mathrm{cfam}_{\mathrm{CPO}}$ and $\mathrm{cfam}_{\mathrm{CPO}^{\mathbf{T}}}$

$$\begin{tikzcd}
\mathrm{CFam}(\mathrm{CPO}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathrm{cfam}_{\mathrm{CPO}}"'] & & \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathrm{cfam}_{\mathrm{CPO}^{\mathbf{T}}}"] \\
& \mathrm{CPO} &
\end{tikzcd}$$

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

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

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

Thm.: we don't use $\mathrm{cod} : \mathrm{CPO}^{\rightarrow} \to \mathrm{CPO}$ 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. ($\perp_A : 1 \longrightarrow TA$), and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- such $\mathbf{T}$ arise from discrete CPO-enriched countable Law. ths.
- we use continuous families fibrations $\mathrm{cfam}_{\mathrm{CPO}}$ and $\mathrm{cfam}_{\mathrm{CPO}^{\mathbf{T}}}$

$$\begin{tikzcd}
\mathrm{CFam}(\mathrm{CPO}) \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\mathrm{cfam}_{\mathrm{CPO}}"'] & & \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathrm{cfam}_{\mathrm{CPO}^{\mathbf{T}}}"] \\
& \mathrm{CPO} &
\end{tikzcd}$$

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

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

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

Thm.: we don't use $\mathrm{cod} : \mathrm{CPO}^{\to} \to \mathrm{CPO}$ 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. ($\bot_A : 1 \longrightarrow TA$), and
  - $\mathrm{CPO}^{\mathbf{T}}$ has reflexive coequalizers
- such $\mathbf{T}$ arise from discrete CPO-enriched countable Law. ths.
- we use continuous families fibrations $\mathrm{cfam}_{\mathrm{CPO}}$ and $\mathrm{cfam}_{\mathrm{CPO}^{\mathbf{T}}}$

$$\begin{tikzcd} \mathrm{CFam}(\mathrm{CPO}) \arrow[rr, bend left, "F"] \arrow[dr, "\mathrm{cfam}_{\mathrm{CPO}}"'] \arrow[rr, phantom, "\perp"] & & \mathrm{CFam}(\mathrm{CPO}^{\mathbf{T}}) \arrow[ll, bend left, "U"] \arrow[dl, "\mathrm{cfam}_{\mathrm{CPO}^{\mathbf{T}}}"] \\ & \mathrm{CPO} & \end{tikzcd}$$

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

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

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

Thm.: we don't use $\mathrm{cod} : \mathrm{CPO}^{\rightarrow} \rightarrow \mathrm{CPO}$ 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.,

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

- 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

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

- **Prop.:** every $\mathbf{T}$ on a split closed comp. cat. has a dep. strength

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

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

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

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

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

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

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

- **Prop.:** every $\mathbf{T}$ on a split closed comp. cat. has a dep. strength

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

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

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

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

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

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

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

- **Prop.:** every $\mathbf{T}$ on a split closed comp. cat. has a dep. strength

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

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

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

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

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

Algebraic effects

Algebraic effects – ops. and eqs.

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

- 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

In eMLTT:

$$M ::= \ldots \mid \mathsf{op}^{\underline{C}}_{V}(x.M)$$

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

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

Sound semantics: based on

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

Algebraic effects – ops. and eqs.

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

- 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

In eMLTT:

$$M ::= \ldots \mid \mathsf{op}^{\underline{C}}_V(x.M)$$

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

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

Sound semantics: based on

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

Algebraic effects – ops. and eqs.

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

- 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

In eMLTT:

$$M ::= \ldots \mid \mathsf{op}^{\underline{C}}_V(x.M)$$

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

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

Sound semantics: based on

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

Algebraic effects – ops. and eqs.

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

- 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

In eMLTT:

$$M ::= \ldots \mid \mathsf{op}^{\underline{C}}_{V}(x.M)$$

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

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

Sound semantics: based on

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

Algebraic effects – examples

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):

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

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

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

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

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

- five equations (two of them branching 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 – examples

Example 1 (interactive I/O):

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

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

- no equations

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

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

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

  $\diamond \models_{\mathsf{v}} \mathsf{isDec}_{\mathsf{Loc}} : \Pi \ell : \mathsf{Loc} \,.\, \Pi \ell' : \mathsf{Loc} \,.\, (\ell =_{\mathsf{Loc}} \ell') + (\ell =_{\mathsf{Loc}} \ell' \to 0)$

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

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

- five equations (two of them branching 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 – examples

Example 1 (interactive I/O):

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

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

- no equations

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

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

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

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

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

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

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

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

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:

$$\Gamma \vdash_{\mathsf{c}} M \ \mathtt{handled\ with}\ \{\mathsf{op}_{x_v}(x_k) \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{T}_{\mathsf{eff}}}\ \mathtt{to}\ y\!:\!A\ \mathtt{in}_{\underline{C}}\ N_{\mathsf{ret}} : \underline{C}$$

satisfying

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

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

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., $S \to X \times S$)

Handlers of alg. effects – for programming

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

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

$$\Gamma \vdash_{\mathsf{c}} M \;\texttt{handled with}\; \{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{T}_{\mathrm{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\mathrm{ret}} : \underline{C}$$

satisfying

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

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

Typical use case for programming:

- 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 [Plotkin,Pretnar'09]

Handler = Algebra and Handling = Homomorphism

Usual term-level presentation:

$$\Gamma \vdash_{\mathsf{c}} M \ \mathtt{handled\ with}\ \{\mathsf{op}_{x_v}(x_k) \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{T}_{\mathsf{eff}}}\ \mathtt{to}\ y\!:\!A\ \mathtt{in}_{\underline{C}}\ N_{\mathsf{ret}} : \underline{C}$$

satisfying

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

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

Typical use case for programming:

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

Handlers of alg. effects – for reasoning

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

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

we can define natural predicates (essentially, dependent types)

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

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 Nat, 1, 0, $+$, Σ, and Π
- 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, it's a bit more extensional)

Handlers of alg. effects – for reasoning

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

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

we can define natural predicates (essentially, dependent types)

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

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 Nat, 1, 0, $+$, Σ, and Π
- 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, it's a bit more extensional)

Handlers of alg. effects – for reasoning

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

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

we can define natural predicates (essentially, dependent types)

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

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

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

- a universe $\mathcal{U}$ closed under Nat, 1, 0, +, Σ, and Π
- 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, it's a bit more extensional)

Handlers of alg. effects – for reasoning

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

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

we can define natural predicates (essentially, dependent types)

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

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 Nat, 1, 0, $+$, Σ, and Π
- 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, 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

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

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

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

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

- 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

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

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

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

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

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

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

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

- 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

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

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

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

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

  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

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

  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

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

  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

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

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

- Assuming an inductive type Protocol, given by

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

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

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

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

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

  by handling the given comp. using

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

  where

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

Handlers of alg. effects – for reasoning

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

- Assuming an inductive type Protocol, given by

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

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

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

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

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

  by handling the given comp. using

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

  where

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

Handlers of alg. effects – for reasoning

Example 3 (Patterns of allowed I/O-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

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

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
- alg. effects and handlers, in particular, for reasoning using
  - Evaluation Logic style modalities
  - Dijkstra's weakest precondition semantics of state
  - patterns of allowed (I/O)-effects

Things to look at:

- type-dependency on computations (e.g., in seq. composition)
- more expressive comp. types (par. adjunctions, Dijkstra monads)

Other work: directed containers, F* and monotonic state, ...

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 of state
  - patterns of allowed (I/O)-effects

Things to look at:

- type-dependency on computations (e.g., in seq. composition)
- more expressive comp. types (par. adjunctions, Dijkstra monads)

Other work: 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)