Dependent Types and Fibred Computational Effects

Danel Ahman[1]

(based on joint work with Gordon Plotkin[1] and Neil Ghani[2])

[1]LFCS, University of Edinburgh

[2]MSP Group, University of Strathclyde

December 11, 2015

Outline

We will discuss language design principles for combining

- dependent types $(\Pi, \Sigma, \mathsf{Id}_A(V, W), \ldots)$
- computational effects (state, I/O, probability, recursion, ...)

In the end we want to

- have a mathematically natural story
- use established tools and methods
- cover a wide range of computational effects

We will be guided by problems with

- effectful programs in types
- assigning types to effectful programs

"Bonus" content for the 2nd half of the talk:

- adding parameters/permissions/worlds/resources/etc.

Effectful programs in types

(type-dependency in the presence of effects)

Effectful programs in types

Q: Can't we simply add dep. types to λ_c and be done with it?

A: Not quite

Let's assume that we have a dependent type $A(x)$, e.g.:

$$x : \mathrm{Nat} \vdash A(x) \quad \stackrel{\text{def}}{=} \quad \texttt{if}\ (x\ \texttt{mod}\ 2 == 0)\ \texttt{then}\ \mathrm{String}\ \texttt{else}\ \mathrm{Char}$$

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

A: Pragmatically, it depends on the nature of effects:

yes for "non-interactive" effects: M need not be restricted

- computing $A[M/x]$ only depends on static information
- (local names [Pitts et al.'15] and recursion [Casinghino et al.'14])

no for general effects: M should be restricted, otherwise

- type-checking starts to depend on interaction with runtime
- e.g., how to compute the substitution $A[\mathrm{send}_V(M)/x]$?
- (examples include I/O, non-determinism, probability, state, etc.)

Effectful programs in types

Q: Can't we simply add dep. types to λ_c and be done with it?

A: Not quite

Let's assume that we have a dependent type $A(x)$, e.g.:

$$x : \mathsf{Nat} \vdash A(x) \quad \stackrel{\text{def}}{=} \quad \mathtt{if}\ (x\ \mathtt{mod}\ 2 == 0)\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Char}$$

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

A: Pragmatically, it depends on the nature of effects:

yes for "non-interactive" effects: M need not be restricted

- computing $A[M/x]$ only depends on static information
- (local names [Pitts et al.'15] and recursion [Casinghino et al.'14])

no for general effects: M should be restricted, otherwise

- type-checking starts to depend on interaction with runtime
- e.g., how to compute the substitution $A[\mathsf{send}_V(M)/x]$?
- (examples include I/O, non-determinism, probability, state, etc.)

Effectful programs in types

Q: Can't we simply add dep. types to λ_c and be done with it?

A: Not quite

Let's assume that we have a dependent type $A(x)$, e.g.:

$$x : \mathsf{Nat} \vdash A(x) \quad \stackrel{\text{def}}{=} \quad \mathtt{if}\ (x\ \mathtt{mod}\ 2 == 0)\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Char}$$

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

A: Pragmatically, it depends on the nature of effects:

yes for "non-interactive" effects: M need not be restricted

- computing $A[M/x]$ only depends on static information
- (local names [Pitts et al.'15] and recursion [Casinghino et al.'14])

no for general effects: M should be restricted, otherwise

- type-checking starts to depend on interaction with runtime
- e.g., how to compute the substitution $A[\mathsf{send}_V(M)/x]$?
- (examples include I/O, non-determinism, probability, state, etc.)

Effectful programs in types ctd.

Aim: Types should only depend on static info about effects

- so, we need to build on something more fine-grained than λ_c

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

Then, types are allowed to depend on effects only via thunks $U\,\underline{C}$

- using only static information and by inspecting comp. trees
- e.g., `followsSession?` : Session $\to U\,\underline{C} \to$ Bool

Note: In theory, we could also let types depend on eff. comps.:

- lifting effect operations from terms to types, e.g., $\mathsf{send}_V(A)$
- similarities with ref. types and op. modalities [A.,Plotkin'15]
- type-dependency ($z : \underline{C} \vdash A(z)$) needs to be "homomorphic"

Effectful programs in types ctd.

Aim: Types should only depend on static info about effects

- so, we need to build on something more fine-grained than λ_c

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

Then, types are allowed to depend on effects only via thunks $U\,\underline{C}$

- using only static information and by inspecting comp. trees
- e.g., $\texttt{followsSession?} : \mathsf{Session} \to U\,\underline{C} \to \mathsf{Bool}$

Note: In theory, we could also let types depend on eff. comps.:

- lifting effect operations from terms to types, e.g., $\mathsf{send}_V(A)$
- similarities with ref. types and op. modalities [A.,Plotkin'15]
- type-dependency ($z : \underline{C} \vdash A(z)$) needs to be "homomorphic"

Effectful programs in types ctd.

Aim: Types should only depend on static info about effects

- so, we need to build on something more fine-grained than λ_c

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

Then, types are allowed to depend on effects only via thunks $U\,\underline{C}$

- using only static information and by inspecting comp. trees
- e.g., $\texttt{followsSession?} : \mathsf{Session} \to U\,\underline{C} \to \mathsf{Bool}$

Note: In theory, we could also let types depend on eff. comps.:

- lifting effect operations from terms to types, e.g., $\mathsf{send}_V(A)$
- similarities with ref. types and op. modalities [A.,Plotkin'15]
- type-dependency $(\, z \!:\! \underline{C} \vdash A(z)\,)$ needs to be "homomorphic"

Assigning types to effectful programs

(typing sequential composition)

Assigning types to effectful programs

How to type the sequential composition of computations?

- a key question for any system with computational effects

Our problem: The standard typing rule for seq. comp.

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

is not correct any more because here x can appear free in

$$\Gamma, x\!:\!A \vdash \underline{C}$$

in the conclusion

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 1: We could restrict the free variables in $\underline{C}$, i.e.:

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

But sometimes it is necessary for $\underline{C}$ to depend on x!

- e.g., in monadic parsing of well-typed syntax (case of functions)

$$x\!:\!\Sigma y_1.\Sigma y_2.\mathsf{LangSyntax}(\mathtt{fun}\,y_1\,y_2) \vDash_{\mathsf{c}} \mathtt{parseFunArg} : F\,(\mathsf{LangSyntax}(\mathtt{fst}\,x))$$

Option 2: We could lift seq. composition to the level of types:

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

But then comp. types contain exactly the terms we want to type!

Conclusion: Option 1 is OK, but not good enough on its own

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 1: We could restrict the free variables in $\underline{C}$, i.e.:

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

But sometimes it is necessary for $\underline{C}$ to depend on x!

- e.g., in monadic parsing of well-typed syntax (case of functions)

$$x\!:\!\Sigma y_1.\Sigma y_2.\mathsf{LangSyntax}(\texttt{fun}\; y_1\; y_2) \models_c \texttt{parseFunArg} : F\,(\mathsf{LangSyntax}(\texttt{fst}\; x))$$

Option 2: We could lift seq. composition to the level of types:

$$\Gamma \models_c M \;\texttt{to}\; x \;\texttt{in}\; N : M \;\texttt{to}\; x \;\texttt{in}\; \underline{C}$$

But then comp. types contain exactly the terms we want to type!

Conclusion: Option 1 is OK, but not good enough on its own

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 1: We could restrict the free variables in $\underline{C}$, i.e.:

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

But sometimes it is necessary for $\underline{C}$ to depend on x!

- e.g., in monadic parsing of well-typed syntax (case of functions)

$$x : \Sigma y_1 . \Sigma y_2 . \mathsf{LangSyntax}(\texttt{fun}\, y_1\, y_2) \vDash_c \texttt{parseFunArg} : F\,(\mathsf{LangSyntax}(\texttt{fst}\, x))$$

Option 2: We could lift seq. composition to the level of types:

$$\Gamma \vDash_c M \; \texttt{to} \; x \; \texttt{in} \; N : M \; \texttt{to} \; x \; \texttt{in} \; \underline{C}$$

But then comp. types contain exactly the terms we want to type!

Conclusion: Option 1 is OK, but not good enough on its own

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 1: We could restrict the free variables in $\underline{C}$, i.e.:

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

But sometimes it is necessary for $\underline{C}$ to depend on x!

- e.g., in monadic parsing of well-typed syntax (case of functions)

$$x : \Sigma y_1.\Sigma y_2.\mathsf{LangSyntax}(\texttt{fun}\, y_1\, y_2) \vdash_{c} \texttt{parseFunArg} : F\,(\mathsf{LangSyntax}(\texttt{fst}\, x))$$

Option 2: We could lift seq. composition to the level of types:

$$\Gamma \vdash_{c} M \texttt{ to } x \texttt{ in } N : M \texttt{ to } x \texttt{ in } \underline{C}$$

But then comp. types contain exactly the terms we want to type!

Conclusion: Option 1 is OK, but not good enough on its own

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 1: We could restrict the free variables in $\underline{C}$, i.e.:

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

But sometimes it is necessary for $\underline{C}$ to depend on x!

- e.g., in monadic parsing of well-typed syntax (case of functions)

$$x\!:\!\Sigma y_1.\Sigma y_2.\mathsf{LangSyntax}(\mathtt{fun}\,y_1\,y_2) \vDash_{\mathsf{c}} \mathtt{parseFunArg} : F\,(\mathsf{LangSyntax}(\mathtt{fst}\,x))$$

Option 2: We could lift seq. composition to the level of types:

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

But then comp. types contain exactly the terms we want to type!

Conclusion: Option 1 is OK, but not good enough on its own

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 3: We draw inspiration from algebraic effects

- and combine it with restricting $\underline{C}$ in seq. comp., as in Option 1

For example, consider the stateful program (for $x : \mathsf{Nat} \vdash_c N : \underline{C}$)

$$M \quad \stackrel{\text{def}}{=} \quad \mathtt{lookup}\,(\mathtt{return}\,2, \mathtt{return}\,3)\ \mathtt{to}\ x\ \mathtt{in}\ N$$

After looking up the bit, this program evaluates as either

$$N[2/x] \ \text{ at type } \ \underline{C}[2/x] \qquad \text{or} \qquad N[3/x] \ \text{ at type } \ \underline{C}[3/x]$$

Solution: M denotes an element of the coproduct of algebras

$$\underline{C}[2/x] + \underline{C}[3/x] \quad \stackrel{\text{def}}{=} \quad F\left(U\left(\underline{C}[2/x]\right) + U\left(\underline{C}[3/x]\right)\right)_{/\equiv}$$

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 3: We draw inspiration from algebraic effects

- and combine it with restricting $\underline{C}$ in seq. comp., as in Option 1

For example, consider the stateful program (for $x : \mathsf{Nat} \vdash_{\mathsf{c}} N : \underline{C}$)

$$M \quad \stackrel{\text{def}}{=} \quad \mathtt{lookup}\,(\mathtt{return}\,2, \mathtt{return}\,3)\ \mathtt{to}\ x\ \mathtt{in}\ N$$

After looking up the bit, this program evaluates as either

$$N[2/x] \ \text{ at type } \ \underline{C}[2/x] \qquad \text{or} \qquad N[3/x] \ \text{ at type } \ \underline{C}[3/x]$$

Solution: M denotes an element of the coproduct of algebras

$$\underline{C}[2/x] + \underline{C}[3/x] \quad \stackrel{\text{def}}{=} \quad F\left(U\left(\underline{C}[2/x]\right) + U\left(\underline{C}[3/x]\right)\right)_{/\equiv}$$

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 3: We draw inspiration from algebraic effects

- and combine it with restricting $\underline{C}$ in seq. comp., as in Option 1

For example, consider the stateful program (for $x : \mathsf{Nat} \vdash_{\mathsf{c}} N : \underline{C}$)

$$M \quad \stackrel{\text{def}}{=} \quad \mathtt{lookup}\,(\mathtt{return}\,2, \mathtt{return}\,3)\ \mathtt{to}\ x\ \mathtt{in}\ N$$

After looking up the bit, this program evaluates as either

$$N[2/x] \;\text{ at type }\; \underline{C}[2/x] \qquad \text{or} \qquad N[3/x] \;\text{ at type }\; \underline{C}[3/x]$$

Solution: M denotes an element of the coproduct of algebras

$$\underline{C}[2/x] + \underline{C}[3/x] \quad \stackrel{\text{def}}{=} \quad F\left(U\left(\underline{C}[2/x]\right) + U\left(\underline{C}[3/x]\right)\right)_{/\equiv}$$

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 3: We draw inspiration from algebraic effects

- and combine it with restricting $\underline{C}$ in seq. comp., as in Option 1

For example, consider the stateful program (for $x\!:\!\mathsf{Nat} \vdash_{\mathsf{c}} N : \underline{C}$)

$$M \quad \stackrel{\text{def}}{=} \quad \mathtt{lookup}\,(\mathtt{return}\,2, \mathtt{return}\,3)\ \mathtt{to}\ x\ \mathtt{in}\ N$$

After looking up the bit, this program evaluates as either

$$N[2/x] \;\text{ at type }\; \underline{C}[2/x] \qquad \text{or} \qquad N[3/x] \;\text{ at type }\; \underline{C}[3/x]$$

Solution: M denotes an element of the coproduct of algebras

$$\underline{C}[2/x] + \underline{C}[3/x] \quad \stackrel{\text{def}}{=} \quad F\left(U\left(\underline{C}[2/x]\right) + U\left(\underline{C}[3/x]\right)\right)_{/\equiv}$$

Assigning types to effectful programs ctd.

Aim: Assigning a sensible type to sequential composition

Option 3: We draw inspiration from algebraic effects

- and combine it with restricting $\underline{C}$ in seq. comp., as in Option 1

For example, consider the stateful program (for $x : \mathsf{Nat} \vdash_{\mathsf{c}} N : \underline{C}$)

$$M \quad \stackrel{\text{def}}{=} \quad \texttt{lookup}\,(\texttt{return}\,2, \texttt{return}\,3)\ \texttt{to}\ x\ \texttt{in}\ N$$

After looking up the bit, this program evaluates as either

$$N[2/x] \ \text{ at type } \ \underline{C}[2/x] \quad \text{ or } \quad N[3/x] \ \text{ at type } \ \underline{C}[3/x]$$

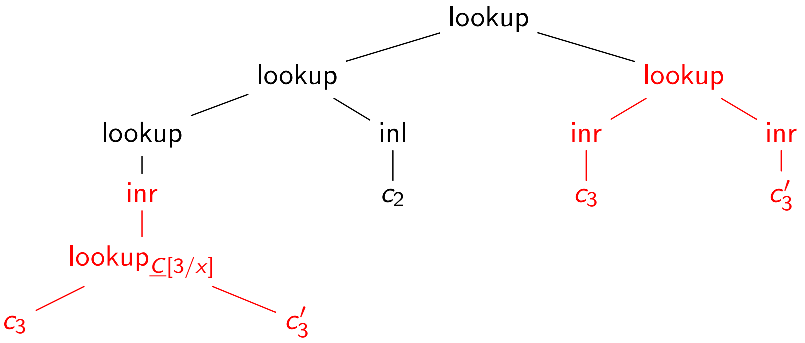
Solution: M denotes an element of the coproduct of algebras

$$\underline{C}[2/x] + \underline{C}[3/x] \quad \stackrel{\text{def}}{=} \quad F\left(U\left(\underline{C}[2/x]\right) + U\left(\underline{C}[3/x]\right)\right)_{/\equiv}$$

Assigning types to effectful programs ctd.

Sidenote: Elements of $\underline{C}[2/x] + \underline{C}[3/x]$ are not only $\mathsf{inl}\, c$ or $\mathsf{inr}\, c$!

- e.g., consider another computation tree in $\underline{C}[2/x] + \underline{C}[3/x]$



- where $\underline{C}[2/x] + \underline{C}[3/x] \quad \stackrel{\text{def}}{=} \quad F\Big(U(\underline{C}[2/x]) + U(\underline{C}[3/x])\Big)_{/\equiv}$
- where $c_2 \in \underline{C}[2/x]$ and $c_3, c_3' \in \underline{C}[3/x]$, and
- where the red subtrees are made equal by $\equiv$

Putting these ideas together

(a core dependently-typed calculus with comp. effects)

A computational dep.-typed language

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

- general computational effects
- a clear distinction between values and computations
- restricting free variables in seq. composition
- using a coproducts of algebras
- a mathematically natural model theory, using standard tools
  - fibrations
  - adjunctions
  - Lawvere theories
  - monads and EM-algebras

A computational dep.-typed language

Value types: MLTT's types + thunks + ...

$$A, B \ ::= \ \mathsf{Nat} \ | \ 1 \ | \ \Pi x\!:\!A.B \ | \ \Sigma x\!:\!A.B \ | \ Id_A(V, W) \ | \ U\,\underline{C} \ | \ \ldots$$

- $U\,\underline{C}$ 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
  - it generalises CBPV's and EEC's computational function type $A \to \underline{C}$ and product type $\underline{C} \times \underline{D}$
- $\Sigma x\!:\!A.\underline{C}$ is the generalisation of coproducts of algebras
  - it generalises EEC's computational tensor type $A \otimes \underline{C}$ and sum type $\underline{C} + \underline{D}$

A computational dep.-typed language

Value types: MLTT's types + thunks + ...

$$A, B \ ::= \ \mathsf{Nat} \mid 1 \mid \Pi x{:}A.B \mid \Sigma x{:}A.B \mid Id_A(V, W) \mid U\,\underline{C} \mid \ \ldots$$

- $U\,\underline{C}$ 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
  - it generalises CBPV's and EEC's computational function type $A \to \underline{C}$ and product type $\underline{C} \times \underline{D}$
- $\Sigma x{:}A.\underline{C}$ is the generalisation of coproducts of algebras
  - it generalises EEC's computational tensor type $A \otimes \underline{C}$ and sum type $\underline{C} + \underline{D}$

A computational dep.-typed language

Value terms: MLTT's terms + thunks + ...

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

- $\mathtt{thunk}\ M$ is the thunk of computation M
- equational theory based on MLTT with intensional id.-types
- value terms are typed using judgment $\Gamma \vdash_{\mathrm{v}} V : A$

Computation terms: dep.-typed version of CBPV/EEC c. terms

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

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

A computational dep.-typed language

Value terms: MLTT's terms + thunks + ...

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

- $\texttt{thunk}\ M$ is the thunk of computation M
- equational theory based on MLTT with intensional id.-types
- value terms are typed using judgment $\Gamma \vdash_{\mathsf{v}} V : A$

Computation terms: dep.-typed version of CBPV/EEC c. terms

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

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

A computational dep.-typed language

Value terms: MLTT's terms + thunks + ...

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

- $\texttt{thunk}\ M$ is the thunk of computation M
- equational theory based on MLTT with intensional id.-types
- value terms are typed using judgment $\Gamma \vdash_{\mathrm{v}} V : A$

Computation terms: dep.-typed version of CBPV/EEC c. terms

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

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

A computational dep.-typed language

Note: We are lead to introduce the terms K so that we:

- can first define comp. Σ elimination., and
- also to preserve intended eval. order in it, i.e., in

$$\Gamma \vDash_{\mathsf{c}} \langle V, M \rangle \ \mathtt{to}\ \langle x, z \rangle \ \mathtt{in}\ K = K[V/x, M/z] : \underline{C}_2$$

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

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

Computation and homomorphic terms are typed using judgments

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

A computational dep.-typed language

Note: We are lead to introduce the terms K so that we:

- can first define comp. Σ elimination., and
- also to preserve intended eval. order in it, i.e., in

$$\Gamma \vDash_{c} \langle V, M\rangle \; \mathtt{to} \; \langle x, z\rangle \; \mathtt{in} \; K = K[V/x, M/z] : \underline{C}_2$$

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

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

Computation and homomorphic terms are typed using judgments

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

A computational dep.-typed language

Typing rules: Dep.-typed versions of CBPV and EEC, e.g.:

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

$$\ldots$$

$$\frac{\Gamma \vdash \underline{C}}{\Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} z : \underline{C}}$$

$$\ldots$$

$$\frac{\Gamma \vDash_{\mathsf{v}} V : A \qquad \Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D}[V/x]}{\Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} \langle V, K \rangle : \Sigma x : A . \underline{D}}$$

$$\frac{\Gamma \,|\, z_1 : \underline{C} \vDash_{\mathsf{h}} K : \Sigma x : A . \underline{D}_1 \qquad \Gamma \vdash \underline{D}_2 \qquad \Gamma, x : A \,|\, z_2 : \underline{D}_1 \vDash_{\mathsf{h}} L : \underline{D}_2}{\Gamma \,|\, z_1 : \underline{C} \vDash_{\mathsf{h}} K\ \mathtt{to}\ \langle x, z_2 \rangle\ \mathtt{in}\ L : \underline{D}_2}$$

Some similarities and some differences with the recent work on linear dependent types by [Krishnaswami et al.'15] and [Vákár'15]

Algebraic effects

(primitives for programming with side-effects)

Algebraic effects

Effect theories:

- we consider signatures of typed operation symbols

$$\frac{\cdot \vdash I \qquad x_i : I \vdash O \qquad I, O \text{ do not contain } U}{\mathsf{op} : (x_i : I) \longrightarrow O}$$

- equipped with equations on derivable effect terms

Example: Global store with two locations (modeled as booleans)

$$\mathsf{lookup} : (x_i : \mathsf{Bool}) \longrightarrow (\mathtt{if}\ x_i\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})$$

$$\mathsf{update} : (x_i : \Sigma x : \mathsf{Bool}.(\mathtt{if}\ x\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})) \longrightarrow 1$$

Algebraic operations:

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

Generic effects:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I}{\Gamma \vDash_{\mathsf{c}} \mathsf{genop}_V : F\,(O[V/x_i])}$$

Homomorphism eq.: (either by postulating or making derivable)

$$\Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D} \quad \Longrightarrow \quad \Gamma \vDash_{\mathsf{c}} K[\mathsf{op}^{\underline{C}}_V(x.M)/z] = \mathsf{op}^{\underline{D}}_V(x.K[M/z]) : \underline{D}$$

Algebraic effects

Effect theories:

- we consider signatures of typed operation symbols

$$\frac{\cdot \vdash I \qquad x_i : I \vdash O \qquad I, O \text{ do not contain } U}{\mathsf{op} : (x_i : I) \longrightarrow O}$$

- equipped with equations on derivable effect terms

Example: Global store with two locations (modeled as booleans)

$$\mathsf{lookup} : (x_i : \mathsf{Bool}) \longrightarrow (\mathtt{if}\ x_i\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})$$

$$\mathsf{update} : \big(x_i : \Sigma x : \mathsf{Bool}.(\mathtt{if}\ x\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})\big) \longrightarrow 1$$

Algebraic operations:

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

Generic effects:

$$\frac{\Gamma \vdash_{\!v} V : I}{\Gamma \vdash_{\!c} \mathsf{genop}_V : F\,(O[V/x_i])}$$

Homomorphism eq.: (either by postulating or making derivable)

$$\Gamma \,|\, z : \underline{C} \vdash_{\!h} K : \underline{D} \quad \Longrightarrow \quad \Gamma \vdash_{\!c} K[\mathsf{op}^{\underline{C}}_{V}(x.M)/z] = \mathsf{op}^{\underline{D}}_{V}(x.K[M/z]) : \underline{D}$$

Algebraic effects

Effect theories:

- we consider signatures of typed operation symbols

$$\frac{\cdot \vdash I \qquad x_i : I \vdash O \qquad I, O \text{ do not contain } U}{\mathsf{op} : (x_i : I) \longrightarrow O}$$

- equipped with equations on derivable effect terms

Example: Global store with two locations (modeled as booleans)

$$\mathsf{lookup} : (x_i : \mathsf{Bool}) \longrightarrow (\mathtt{if}\ x_i\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})$$

$$\mathsf{update} : \big(x_i : \Sigma x : \mathsf{Bool}.(\mathtt{if}\ x\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})\big) \longrightarrow 1$$

Algebraic operations:

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

Generic effects:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I}{\Gamma \vDash_{\mathsf{c}} \mathsf{genop}_V : F\,(O[V/x_i])}$$

Homomorphism eq.: (either by postulating or making derivable)

$$\Gamma \,|\, z : \underline{C} \vDash_{\mathsf{h}} K : \underline{D} \quad \Longrightarrow \quad \Gamma \vDash_{\mathsf{c}} K[\mathsf{op}^{\underline{C}}_{V}(x.M)/z] = \mathsf{op}^{\underline{D}}_{V}(x.K[M/z]) : \underline{D}$$

Algebraic effects

Effect theories:

- we consider signatures of typed operation symbols

$$\frac{\cdot \vdash I \qquad x_i : I \vdash O \qquad I, O \text{ do not contain } U}{\mathsf{op} : (x_i : I) \longrightarrow O}$$

- equipped with equations on derivable effect terms

Example: Global store with two locations (modeled as booleans)

$$\mathsf{lookup} : (x_i : \mathsf{Bool}) \longrightarrow (\mathtt{if}\ x_i\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})$$

$$\mathsf{update} : \big(x_i : \Sigma x : \mathsf{Bool}.(\mathtt{if}\ x\ \mathtt{then}\ \mathsf{String}\ \mathtt{else}\ \mathsf{Nat})\big) \longrightarrow 1$$

Algebraic operations:

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

Generic effects:

$$\frac{\Gamma \vDash_{\mathsf{v}} V : I}{\Gamma \vDash_{\mathsf{c}} \mathsf{genop}_{V} : F\,(O[V/x_i])}$$

Homomorphism eq.: (either by postulating or making derivable)

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

Categorical semantics

(fibrations and adjunctions)

Categorical semantics

Using fibred cat. theory, we define fibred adjunction models

- a sound and complete class of models

given by: i) a standard fibrational model of dependent types

$$\begin{tikzcd}
\mathcal{B}^{\to} \arrow[dr, bend right, "\mathsf{cod}"'] & \mathcal{V} \arrow[l, "\mathcal{P}"'] \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] \\
& \mathcal{B} \arrow[u, "1" description] \arrow[u, phantom, shift right=2.5ex, "\dashv"] \arrow[u, phantom, shift left=2.5ex, "\dashv"]
\end{tikzcd}$$

- a split closed comprehension category $\mathcal{P}$
- where for all $A \in \mathcal{A}$ the display maps in $\mathcal{B}$ are $\pi_A : \{A\} \longrightarrow p(A)$
- inducing the weakening functors $\pi_A^* : \mathcal{V}_{p(A)} \longrightarrow \mathcal{V}_{\{A\}}$
- value Σ- and Π-types are defined as adjoints in $\Sigma_A \dashv \pi_A^* \dashv \Pi_A$

Categorical semantics

Using fibred cat. theory, we define fibred adjunction models

- a sound and complete class of models

given by: i) a standard fibrational model of dependent types

$$
\begin{tikzcd}
\mathcal{B}^{\to} \arrow[dr, bend right, "\mathrm{cod}"'] & \mathcal{V} \arrow[l, "\mathcal{P}"'] \arrow[d, bend right=60, "p"'] \arrow[d, bend left=60, "\{-\}"] \\
& \mathcal{B} \arrow[u, "1"'] \arrow[u, phantom, shift left=4, "\dashv"] \arrow[u, phantom, shift right=4, "\dashv"]
\end{tikzcd}
$$

- a split closed comprehension category $\mathcal{P}$
- where for all $A \in \mathcal{A}$ the display maps in $\mathcal{B}$ are $\pi_A : \{A\} \longrightarrow p(A)$
- inducing the weakening functors $\pi_A^* : \mathcal{V}_{p(A)} \longrightarrow \mathcal{V}_{\{A\}}$
- value Σ- and Π-types are defined as adjoints in $\Sigma_A \dashv \pi_A^* \dashv \Pi_A$

Categorical semantics

Using fibred cat. theory, we define fibred adjunction models

- a sound and complete class of models

given by: ii) an adjunction to model computational effects

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\mathcal{B}^{\to} \arrow[dr, bend right, "\mathrm{cod}"'] & \mathcal{V} \arrow[l, "\mathcal{P}"'] \arrow[r, bend left, "F"] \arrow[r, phantom, "\perp"] \arrow[d, bend right=50, "p"'] \arrow[d, bend left=50, "\{-\}"] & \mathcal{C} \arrow[l, bend left, "U"] \arrow[dl, bend left, "q"] \\
& \mathcal{B} \arrow[u, "\dashv\ 1\ \dashv" description] &
\end{tikzcd}
$$

- a split fibration q and a split fibred adjunction $F \dashv U$
- where for all $A \in \mathcal{A}$ the display maps in $\mathcal{B}$ are $\pi_A : \{A\} \longrightarrow p(A)$
- inducing the weakening functors $\pi_A^* : \mathcal{C}_{p(A)} \longrightarrow \mathcal{C}_{\{A\}}$
- comp. Σ- and Π-types are also defined as adjoints $\Sigma_A \dashv \pi_A^* \dashv \Pi_A$

Categorical semantics

Some sources of examples (writing fib. adj. with total cats. only):

- for a split closed comprehension cat. $\mathcal{P} : \mathcal{V} \longrightarrow \mathcal{B}^{\rightarrow}$, we have

$$\mathsf{Id}_{\mathcal{V}} \dashv \mathsf{Id}_{\mathcal{V}} : \mathcal{V} \longrightarrow \mathcal{V}$$

- for a model of EEC ($\mathcal{V}$ is CCC, $\mathcal{C}$ is $\mathcal{V}$-enriched, $\mathcal{V}$-enr. adj., etc.)

$$F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathsf{s}(\mathcal{V}, \mathcal{C}) \longrightarrow \mathsf{s}(\mathcal{V})$$

Categorical semantics

Some sources of examples (writing fib. adj. with total cats. only):

- for a split closed comprehension cat. $\mathcal{P} : \mathcal{V} \longrightarrow \mathcal{B}^{\rightarrow}$, we have

$$\mathrm{Id}_{\mathcal{V}} \dashv \mathrm{Id}_{\mathcal{V}} : \mathcal{V} \longrightarrow \mathcal{V}$$

- for a model of EEC ($\mathcal{V}$ is CCC, $\mathcal{C}$ is $\mathcal{V}$-enriched, $\mathcal{V}$-enr. adj., etc.)

$$F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathrm{s}(\mathcal{V}, \mathcal{C}) \longrightarrow \mathrm{s}(\mathcal{V})$$

- for a countable Lawvere theory $\mathcal{L}$ and $\mathcal{P}_{\mathsf{fam}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}^{\rightarrow}$

$$\widehat{F_{\mathcal{L}}} \dashv \widehat{U_{\mathcal{L}}} : \mathsf{Fam}(\mathsf{Mod}(\mathcal{L}, \mathsf{Set})) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

- for a monad $T : \mathsf{Set} \longrightarrow \mathsf{Set}$ and $\mathcal{P}_{\mathsf{fam}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}^{\rightarrow}$

$$\widehat{F^{T}} \dashv \widehat{U^{T}} : \mathsf{Fam}(\mathsf{Set}^{T}) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

- for the continuations monad $R^{R^{(-)}} : \mathsf{Set} \longrightarrow \mathsf{Set}$, we have

$$\widehat{R^{(-)}} \dashv \widehat{R^{(-)}} : \mathsf{Fam}(\mathsf{Set}^{op}) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

Categorical semantics

Some sources of examples (writing fib. adj. with total cats. only):

- for a split closed comprehension cat. $\mathcal{P} : \mathcal{V} \longrightarrow \mathcal{B}^{\rightarrow}$, we have

$$\mathsf{Id}_{\mathcal{V}} \dashv \mathsf{Id}_{\mathcal{V}} : \mathcal{V} \longrightarrow \mathcal{V}$$

- for a model of EEC ($\mathcal{V}$ is CCC, $\mathcal{C}$ is $\mathcal{V}$-enriched, $\mathcal{V}$-enr. adj., etc.)

$$F_{\mathsf{EEC}} \dashv U_{\mathsf{EEC}} : \mathsf{s}(\mathcal{V}, \mathcal{C}) \longrightarrow \mathsf{s}(\mathcal{V})$$

- for a countable Lawvere theory $\mathcal{L}$ and $\mathcal{P}_{\mathsf{fam}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}^{\rightarrow}$

$$\widehat{F_{\mathcal{L}}} \dashv \widehat{U_{\mathcal{L}}} : \mathsf{Fam}(\mathsf{Mod}(\mathcal{L}, \mathsf{Set})) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

- for a monad $T : \mathsf{Set} \longrightarrow \mathsf{Set}$ and $\mathcal{P}_{\mathsf{fam}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}^{\rightarrow}$

$$\widehat{F^T} \dashv \widehat{U^T} : \mathsf{Fam}(\mathsf{Set}^T) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

- for the continuations monad $R^{R^{(-)}} : \mathsf{Set} \longrightarrow \mathsf{Set}$, we have

$$\widehat{R^{(-)}} \dashv \widehat{R^{(-)}} : \mathsf{Fam}(\mathsf{Set}^{\mathsf{op}}) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

Categorical semantics

More sources of examples (writing fib. adj. with total cats. only):

- these last three examples are instances of a more general result:

  for $\mathcal{P}_{\mathsf{fam}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}^{\rightarrow}$ and $F \dashv U : \mathcal{C} \longrightarrow \mathsf{Set}$, when $\mathcal{C}$ has set-indexed products and set-indexed coproducts, we have

  $$\widehat{F} \dashv \widehat{U} : \mathsf{Fam}(\mathcal{C}) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

Categorical semantics

More sources of examples (writing fib. adj. with total cats. only):

- these last three examples are instances of a more general result:

  for $\mathcal{P}_{\mathsf{fam}} : \mathsf{Fam}(\mathsf{Set}) \longrightarrow \mathsf{Set}^{\rightarrow}$ and $F \dashv U : \mathcal{C} \longrightarrow \mathsf{Set}$, when $\mathcal{C}$ has set-indexed products and set-indexed coproducts, we have

  $$\widehat{F} \dashv \widehat{U} : \mathsf{Fam}(\mathcal{C}) \longrightarrow \mathsf{Fam}(\mathsf{Set})$$

- for a $\mathcal{CPO}$-enriched monad $T : \mathcal{CPO} \longrightarrow \mathcal{CPO}$ with a least algebraic operation $\Omega : 0$ and reflexive coequalizers in $\mathcal{CPO}^T$

  $$\widehat{F^T} \dashv \widehat{U^T} : \mathsf{CFam}(\mathcal{CPO}^T) \longrightarrow \mathsf{CFam}(\mathcal{CPO})$$

  allows us to treat general recursion as a computational effect

  $$\frac{\Gamma, x : U\underline{C} \vdash_{\mathsf{c}} M : \underline{C}}{\Gamma \vdash_{\mathsf{c}} \mu x : U\underline{C}.M : \underline{C}}$$

  (we get such monads from $\mathcal{CPO}$-enriched Law. theories with Ω)

Combining effect- and dependent-typing

(adding parameters/worlds/permissions/etc.)

Fibred parametrised comp. effects

Aim: To make our comp. types more expressive

- we extend our language with an effect-and-type system
- we build on [Atkey'09]'s parametrised notions of computation
- we take par. adjunctions as a primitive construction
- we make the effect annotations internal to our language
- we want a semantics for [Brady'13,'14]'s `Effects` DSL for Idris

We omit: Details of the accompanying denotational semantics

- based on fibred analogues of parametrised adjunctions, e.g.,

$$
\begin{tikzcd}
\mathcal{W} \arrow[d, "r"'] & \mathcal{V} \arrow[d, "p"'] & \int\big(\lambda X.\, \mathcal{W}_X \times \mathcal{V}_X\big) \arrow[rr, "F"] \arrow[dr, "\mathsf{fst}"'] & & \mathcal{C} \arrow[dl, "q"] \\
\mathcal{B} & \mathcal{B} & & \mathcal{B} &
\end{tikzcd}
$$

- in particular, we take $\mathcal{W} \overset{\text{def}}{=} \int\Big(\lambda X.\mathcal{V}_X(1_X, !^*_X(\llbracket S \rrbracket))\Big)$

Fibred parametrised comp. effects

Aim: To make our comp. types more expressive

- we extend our language with an effect-and-type system
- we build on [Atkey'09]'s parametrised notions of computation
- we take par. adjunctions as a primitive construction
- we make the effect annotations internal to our language
- we want a semantics for [Brady'13,'14]'s `Effects` DSL for Idris

We omit: Details of the accompanying denotational semantics

- based on fibred analogues of parametrised adjunctions, e.g.,

$$\begin{array}{c} \mathcal{W} \\ {\scriptstyle r}\big\downarrow \\ \mathcal{B} \end{array} \qquad \begin{array}{c} \mathcal{V} \\ {\scriptstyle p}\big\downarrow \\ \mathcal{B} \end{array} \qquad \begin{tikzcd} \int\left(\lambda X.\, \mathcal{W}_X \times \mathcal{V}_X\right) \arrow[rr, "F"] \arrow[dr, "\mathsf{fst}"'] & & \mathcal{C} \arrow[dl, "q"] \\ & \mathcal{B} & \end{tikzcd}$$

- in particular, we take $\mathcal{W} \stackrel{\text{def}}{=} \int\left(\lambda X. \mathcal{V}_X\left(1_X, !_X^*(\llbracket S \rrbracket)\right)\right)$

Fibred parametrised comp. effects

Aim: To extend our language with an effect-and-type system

Our solution: Use fibred version of S-parametrised adjunctions

$$\frac{\Gamma \vdash A \qquad \Gamma \vDash_{\mathsf{v}} W : S}{\Gamma \vdash F_W\, A} \qquad \frac{\Gamma \vdash \underline{C} \qquad \Gamma \vDash_{\mathsf{v}} W : S}{\Gamma \vdash U_W\, \underline{C}}$$

with the resulting S-parametrised monad (`EffM` in Idris) given by

$$\Gamma \vdash T_{W_1,W_2}\, A \stackrel{\text{def}}{=} U_{W_1}\,(F_{W_2}\, A)$$

The main changes we make to our language:

- typing judgment for comp. terms: $\Gamma \,|\, W \vDash_{\mathsf{c}} M : \underline{C}$
- returning values: $\Gamma \,|\, W \vDash_{\mathsf{c}} \mathtt{return}_W\, V : F_W\, A$
- thunking computations: $\Gamma \vDash_{\mathsf{v}} \mathtt{thunk}^{\underline{C}}_W\, M : U_W\, \underline{C}$
- forcing of thunks: $\Gamma \,|\, W \vDash_{\mathsf{c}} \mathtt{force}^{\underline{C}}_W\, V : \underline{C}$

Fibred parametrised comp. effects

Aim: To extend our language with an effect-and-type system

Our solution: Use fibred version of S-parametrised adjunctions

$$\frac{\Gamma \vdash A \qquad \Gamma \vDash_{\mathrm{v}} W : S}{\Gamma \vdash F_W\, A} \qquad \frac{\Gamma \vdash \underline{C} \qquad \Gamma \vDash_{\mathrm{v}} W : S}{\Gamma \vdash U_W\, \underline{C}}$$

with the resulting S-parametrised monad (`EffM` in Idris) given by

$$\Gamma \vdash T_{W_1, W_2}\, A \stackrel{\mathrm{def}}{=} U_{W_1}\, (F_{W_2}\, A)$$

The main changes we make to our language:

- typing judgment for comp. terms: $\Gamma \,|\, W \vDash_{\mathrm{c}} M : \underline{C}$
- returning values: $\Gamma \,|\, W \vDash_{\mathrm{c}} \mathtt{return}_W\, V : F_W\, A$
- thunking computations: $\Gamma \vDash_{\mathrm{v}} \mathtt{thunk}^{\underline{C}}_{W}\, M : U_W\, \underline{C}$
- forcing of thunks: $\Gamma \,|\, W \vDash_{\mathrm{c}} \mathtt{force}^{\underline{C}}_{W}\, V : \underline{C}$

Fibred parametrised comp. effects

Aim: To extend our language with an effect-and-type system

Our solution: Use fibred version of S-parametrised adjunctions

$$\frac{\Gamma \vdash A \qquad \Gamma \models_{\mathsf{v}} W : S}{\Gamma \vdash F_W\, A} \qquad \frac{\Gamma \vdash \underline{C} \qquad \Gamma \models_{\mathsf{v}} W : S}{\Gamma \vdash U_W\, \underline{C}}$$

with the resulting S-parametrised monad (`EffM` in Idris) given by

$$\Gamma \vdash T_{W_1, W_2}\, A \stackrel{\text{def}}{=} U_{W_1}\, (F_{W_2}\, A)$$

The main changes we make to our language:

- typing judgment for comp. terms: $\Gamma \,|\, W \models_{\mathsf{c}} M : \underline{C}$
- returning values: $\Gamma \,|\, W \models_{\mathsf{c}} \mathtt{return}_W\, V : F_W\, A$
- thunking computations: $\Gamma \models_{\mathsf{v}} \mathtt{thunk}^{\underline{C}}_{W}\, M : U_W\, \underline{C}$
- forcing of thunks: $\Gamma \,|\, W \models_{\mathsf{c}} \mathtt{force}^{\underline{C}}_{W}\, V : \underline{C}$

Fibred parametrised comp. effects

Aim: We can explain [Brady'14]'s resource-dependent effects

Example: We will look at the prototypical example of:

- locking-unlocking / opening-closing / authenticating / etc.

As usual, the non-failing operations are easy to specify, e.g.,

$$\Gamma \,|\, \textbf{acquired} \vDash_{c} \texttt{lookup} : F_{\textbf{acquired}}\, \mathsf{String}$$

$$\Gamma \,|\, \textbf{acquired} \vDash_{c} \texttt{update}_{V} : F_{\textbf{acquired}}\, 1$$

$$\Gamma \,|\, \textbf{acquired} \vDash_{c} \texttt{releaseLock} : F_{\textbf{released}}\, \mathsf{Bool}$$

(in terms of generic effects, omitting the corresponding signature)

Q: However, what to do with possibly failing operations?

$$\Gamma \,|\, \textbf{released} \vDash_{c} \texttt{acquireLock} : F_{???}\, \mathsf{Bool}$$

Fibred parametrised comp. effects

Aim: We can explain [Brady'14]'s resource-dependent effects

Example: We will look at the prototypical example of:

- locking-unlocking / opening-closing / authenticating / etc.

As usual, the non-failing operations are easy to specify, e.g.,

$$\Gamma \,|\, \textbf{acquired} \vDash_c \texttt{lookup} : F_{\textbf{acquired}}\, \mathrm{String}$$

$$\Gamma \,|\, \textbf{acquired} \vDash_c \texttt{update}_V : F_{\textbf{acquired}}\, 1$$

$$\Gamma \,|\, \textbf{acquired} \vDash_c \texttt{releaseLock} : F_{\textbf{released}}\, \mathrm{Bool}$$

(in terms of generic effects, omitting the corresponding signature)

Q: However, what to do with possibly failing operations?

$$\Gamma \,|\, \textbf{released} \vDash_c \texttt{acquireLock} : F_{???}\, \mathrm{Bool}$$

Fibred parametrised comp. effects

Aim: We can explain [Brady'14]'s resource-dependent effects

Example: We will look at the prototypical example of:

- locking-unlocking / opening-closing / authenticating / etc.

As usual, the non-failing operations are easy to specify, e.g.,

$$\Gamma \mid \mathbf{acquired} \vDash_{c} \mathtt{lookup} : F_{\mathbf{acquired}}\,\mathsf{String}$$

$$\Gamma \mid \mathbf{acquired} \vDash_{c} \mathtt{update}_{V} : F_{\mathbf{acquired}}\,1$$

$$\Gamma \mid \mathbf{acquired} \vDash_{c} \mathtt{releaseLock} : F_{\mathbf{released}}\,\mathsf{Bool}$$

(in terms of generic effects, omitting the corresponding signature)

Q: However, what to do with possibly failing operations?

$$\Gamma \mid \mathbf{released} \vDash_{c} \mathtt{acquireLock} : F_{???}\,\mathsf{Bool}$$

Fibred parametrised comp. effects

Aim: We can explain [Brady'14]'s resource-dependent effects

Example: We will look at the prototypical example of:

- locking-unlocking / opening-closing / authenticating / etc.

As usual, the non-failing operations are easy to specify, e.g.,

$$\Gamma \,|\, \mathbf{acquired} \vDash_{c} \mathtt{lookup} : F_{\mathbf{acquired}}\,\mathsf{String}$$

$$\Gamma \,|\, \mathbf{acquired} \vDash_{c} \mathtt{update}_{V} : F_{\mathbf{acquired}}\,1$$

$$\Gamma \,|\, \mathbf{acquired} \vDash_{c} \mathtt{releaseLock} : F_{\mathbf{released}}\,\mathsf{Bool}$$

(in terms of generic effects, omitting the corresponding signature)

Q: However, what to do with possibly failing operations?

$$\Gamma \,|\, \mathbf{released} \vDash_{c} \mathtt{acquireLock} : F_{???}\,\mathsf{Bool}$$

Fibred parametrised comp. effects

Q: What to do with possibly failing operations?

A1: If going with the monadic view, then we can try to define another (more dep.-parametrised) monad-like functor

$$\frac{\Gamma \vDash_{\mathsf{v}} W_1 : S \qquad \Gamma \vdash A \qquad \Gamma, x\!:\!A \vDash_{\mathsf{v}} W_2 : S}{\Gamma \vdash T_{W_1}\big((x\!:\!A).W_2\big)}$$

and specify the lock acquiring generic effect as

$$\Gamma \vdash \texttt{acquireLock} : T_{\textbf{released}}\big((x\!:\!\mathsf{Bool}).\texttt{if}\ x\ \texttt{then}\ \textbf{acquired}\ \texttt{else}\ \textbf{released}\big)$$

- a natural generalisation of the functor part of fib. par. monads
- this is the approach that [Brady'14] took for Idris
- but no clear way of equipping it with par. adjunction structure

But: We can achieve the same with our less dep.-typed F and U!

Fibred parametrised comp. effects

Q: What to do with possibly failing operations?

A1: If going with the monadic view, then we can try to define another (more dep.-parametrised) monad-like functor

$$\frac{\Gamma \vDash_{\mathsf{v}} W_1 : S \qquad \Gamma \vdash A \qquad \Gamma, x{:}A \vDash_{\mathsf{v}} W_2 : S}{\Gamma \vdash T_{W_1}\big((x{:}A).W_2\big)}$$

and specify the lock acquiring generic effect as

$$\Gamma \vdash \mathtt{acquireLock} : T_{\textbf{released}}\big((x{:}\mathsf{Bool}).\mathtt{if}\ x\ \mathtt{then}\ \textbf{acquired}\ \mathtt{else}\ \textbf{released}\big)$$

- a natural generalisation of the functor part of fib. par. monads
- this is the approach that [Brady'14] took for Idris
- but no clear way of equipping it with par. adjunction structure

But: We can achieve the same with our less dep.-typed F and U!

Fibred parametrised comp. effects

Q: What to do with possibly failing operations?

A1: If going with the monadic view, then we can try to define another (more dep.-parametrised) monad-like functor

$$\frac{\Gamma \vDash_{\mathsf{v}} W_1 : S \qquad \Gamma \vdash A \qquad \Gamma, x\!:\!A \vDash_{\mathsf{v}} W_2 : S}{\Gamma \vdash T_{W_1}\big((x\!:\!A).W_2\big)}$$

and specify the lock acquiring generic effect as

$$\Gamma \vdash \texttt{acquireLock} : T_{\textbf{released}}\big((x\!:\!\mathsf{Bool}).\texttt{if}\ x\ \texttt{then}\ \textbf{acquired}\ \texttt{else}\ \textbf{released}\big)$$

- a natural generalisation of the functor part of fib. par. monads
- this is the approach that [Brady'14] took for Idris
- but no clear way of equipping it with par. adjunction structure

But: We can achieve the same with our less dep.-typed F and U!

Fibred parametrised comp. effects

Q: What to do with possibly failing operations?

A2a: If we keep with the (par.) adjunctions view, we can define the more dependently-parametrised monad-like functor as

$$\frac{\Gamma \vDash_{\mathsf{v}} W_1 : S \qquad \Gamma \vdash A \qquad \Gamma, x : A \vDash_{\mathsf{v}} W_2 : S}{\Gamma \vdash T_{W_1}\big((x : A).W_2\big) \quad \stackrel{\text{def}}{=} \quad U_{W_1}\,(\Sigma x : A.(F_{W_2}\,1))}$$

using the comp. Σ-types to quantify over the possible outcomes

A2b: We can then specify the lock acquiring generic effect as

$$\Gamma \,|\, \mathbf{released} \vDash_{\mathsf{c}} \mathtt{acquireLock} : \Sigma x : \mathsf{Bool}.(F_{(\mathtt{if}\ x\ \mathtt{then}\ \mathbf{acquired}\ \mathtt{else}\ \mathbf{released})}\,1)$$

Fibred parametrised comp. effects

Q: What to do with possibly failing operations?

A2a: If we keep with the (par.) adjunctions view, we can define the more dependently-parametrised monad-like functor as

$$\frac{\Gamma \vDash_{\mathrm{v}} W_1 : S \qquad \Gamma \vdash A \qquad \Gamma, x : A \vDash_{\mathrm{v}} W_2 : S}{\Gamma \vdash T_{W_1}\big((x : A).W_2\big) \quad \stackrel{\text{def}}{=} \quad U_{W_1}\left(\Sigma x : A.(F_{W_2}\, 1)\right)}$$

using the comp. Σ-types to quantify over the possible outcomes

A2b: We can then specify the lock acquiring generic effect as

$$\Gamma \mid \textbf{released} \vDash_{\mathrm{c}} \texttt{acquireLock} : \Sigma x : \mathsf{Bool}.(F_{(\texttt{if } x \texttt{ then } \textbf{acquired} \texttt{ else } \textbf{released})}\, 1)$$

Parametrised fibred algebraic effects

Parametrised effect theories:

- we consider signatures of typed operation symbols

$$\frac{x_{\mathrm{w}} : S \vdash I \qquad x_{\mathrm{w}} : S, x_{\mathrm{in}} : I \vdash O \qquad x_{\mathrm{w}} : S, x_{\mathrm{in}} : I, x_{\mathrm{in}} : O \vDash_{\mathrm{v}} W_{\mathrm{out}} : S}{\mathrm{op}_{x_{\mathrm{w}}, x_{\mathrm{in}}, x_{\mathrm{out}}} : I \longrightarrow O, W_{\mathrm{out}}}$$

- equipped with equations on derivable effect terms

Algebraic operations:

$$\frac{\Gamma \vDash_{\mathrm{v}} V : I[W/x_{\mathrm{w}}] \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x : O[W/x_{\mathrm{w}}, V/x_{\mathrm{in}}] \mid W_{\mathrm{out}}[W/x_{\mathrm{w}}, \ldots] \vDash_{\mathrm{c}} M : \underline{C}}{\Gamma \mid W \vDash_{\mathrm{c}} \mathrm{op}^{\underline{C}}_{V}(x.M) : \underline{C}}$$

Generic effects:

$$\frac{\Gamma \vDash_{\mathrm{v}} V : I[W/x_{\mathrm{w}}]}{\Gamma \mid W \vDash_{\mathrm{c}} \mathrm{genop}_{V} : \Sigma\, x : O[W/x_{\mathrm{w}}, V/x_{\mathrm{in}}] \,.\, F_{W_{\mathrm{out}}[W/x_{\mathrm{w}}, V/x_{\mathrm{in}}, x/x_{\mathrm{out}}]}\, 1}$$

Result: Such alg. ops. and gen. effs. are in 1-1 relationship

Note: Currently working on equipping W's with order/morphisms

Parametrised fibred algebraic effects

Parametrised effect theories:

- we consider signatures of typed operation symbols

$$\frac{x_{\mathrm{w}} : S \vdash I \quad x_{\mathrm{w}} : S, x_{\mathrm{in}} : I \vdash O \quad x_{\mathrm{w}} : S, x_{\mathrm{in}} : I, x_{\mathrm{in}} : O \models_{\mathrm{v}} W_{\mathrm{out}} : S}{\mathrm{op}_{x_{\mathrm{w}}, x_{\mathrm{in}}, x_{\mathrm{out}}} : I \longrightarrow O, W_{\mathrm{out}}}$$

- equipped with equations on derivable effect terms

Algebraic operations:

$$\frac{\Gamma \models_{\mathrm{v}} V : I[W/x_{\mathrm{w}}] \quad \Gamma \vdash \underline{C} \quad \Gamma, x : O[W/x_{\mathrm{w}}, V/x_{\mathrm{in}}] \mid W_{\mathrm{out}}[W/x_{\mathrm{w}}, \ldots] \models_{\mathrm{c}} M : \underline{C}}{\Gamma \mid W \models_{\mathrm{c}} \mathrm{op}^{\underline{C}}_{V}(x.M) : \underline{C}}$$

Generic effects:

$$\frac{\Gamma \models_{\mathrm{v}} V : I[W/x_{\mathrm{w}}]}{\Gamma \mid W \models_{\mathrm{c}} \mathrm{genop}_V : \Sigma x : O[W/x_{\mathrm{w}}, V/x_{\mathrm{in}}] . F_{W_{\mathrm{out}}[W/x_{\mathrm{w}}, V/x_{\mathrm{in}}, x/x_{\mathrm{out}}]} 1}$$

Result: Such alg. ops. and gen. effs. are in 1-1 relationship

Note: Currently working on equipping W's with order/morphisms

Parametrised fibred algebraic effects

Parametrised effect theories:

- we consider signatures of typed operation symbols

$$\frac{x_w : S \vdash I \quad x_w : S, x_{in} : I \vdash O \quad x_w : S, x_{in} : I, x_{in} : O \vDash_{v} W_{out} : S}{\mathrm{op}_{x_w, x_{in}, x_{out}} : I \longrightarrow O, W_{out}}$$

- equipped with equations on derivable effect terms

Algebraic operations:

$$\frac{\Gamma \vDash_{v} V : I[W/x_w] \quad \Gamma \vdash \underline{C} \quad \Gamma, x : O[W/x_w, V/x_{in}] \mid W_{out}[W/x_w, \ldots] \vDash_{c} M : \underline{C}}{\Gamma \mid W \vDash_{c} \mathrm{op}^{\underline{C}}_{V}(x.M) : \underline{C}}$$

Generic effects:

$$\frac{\Gamma \vDash_{v} V : I[W/x_w]}{\Gamma \mid W \vDash_{c} \mathtt{genop}_V : \Sigma\, x : O[W/x_w, V/x_{in}] \,.\, F_{W_{out}[W/x_w, V/x_{in}, x/x_{out}]}\, 1}$$

Result: Such alg. ops. and gen. effs. are in 1-1 relationship

Note: Currently working on equipping W's with order/morphisms

Conclusions

A dependently-typed computational language with

- clear distinction between values and computations
- new and useful structure on comp. types (Σ-types)
- dep.-typed algebraic effects and first-class handlers
- general recursion as comp. effect
- universes of value and comp. types (omitted)
- natural categorical semantics, using standard tools
- parametrised fibred computational effects and a principled account of Brady's resource-dependent effects in Idris

Thank you for listening!

Conclusions

A dependently-typed computational language with

- clear distinction between values and computations
- new and useful structure on comp. types (Σ-types)
- dep.-typed algebraic effects and first-class handlers
- general recursion as comp. effect
- universes of value and comp. types (omitted)
- natural categorical semantics, using standard tools
- parametrised fibred computational effects and a principled account of Brady's resource-dependent effects in Idris

Thank you for listening!