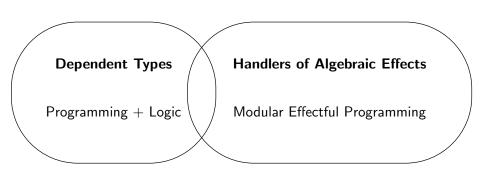
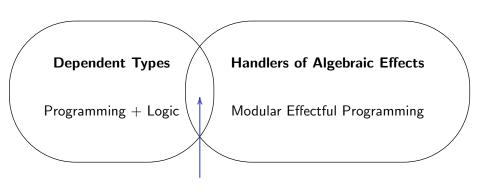
#### **Handling Fibred Algebraic Effects**

Danel Ahman INRIA Paris

POPL 2018 January 10, 2018





#### **Outline**

- Setting the scene
  - Algebraic effects and their handlers
  - A core effectful dependently typed calculus (FoSSaCS'16)

[A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Modular programming with handlers + expressiveness of d. types
  - Extrinsic reasoning about effectful computations
- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A new type-level treatment of handlers

#### **Outline**

- Setting the scene
  - Algebraic effects and their handlers
  - A core effectful dependently typed calculus (FoSSaCS'16)
     [A., Ghani, Plotkin'16]
- What can we gain from handlers + dependent types?
  - Modular programming with handlers + expressiveness of d. types
  - Extrinsic reasoning about effectful computations
- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A new type-level treatment of handlers

• Moggi taught us to model comp. effects using **monads**  $(T, \eta, (-)^{\dagger})$ 

$$\eta_A:A\to TA$$
  $(f:A\to TB)^{\dagger}_{A.B}:TA\to TB$ 

- Plotkin and Power showed that most of these monads arise from
  - operation symbols representing the sources of effects

$$\mathsf{raise} : \mathsf{Exc} \longrightarrow \mathsf{0} \qquad \mathsf{get} : \mathsf{Loc} \longrightarrow \mathsf{Val} \qquad \mathsf{put} : \mathsf{Loc} \times \mathsf{Val} \longrightarrow \mathsf{I}$$

• equations – describing the computational behaviour

$$\ell : \mathsf{Loc} \mid w : 1 \vdash \mathsf{get}_{\ell}(x.\mathsf{put}_{\langle \ell, x \rangle}(w(\star))) = w(\star)$$

- The algebraic approach significantly simplifies
  - choosing a monad/adjunction to model a given language
  - modelling combinations of two or more comp. effects
  - generic effectful programming (via handlers)

• Moggi taught us to model comp. effects using **monads**  $(T, \eta, (-)^{\dagger})$ 

$$\eta_A:A \to TA$$
  $(f:A \to TB)^{\dagger}_{A,B}:TA \to TB$ 

- Plotkin and Power showed that most of these monads arise from
  - operation symbols representing the sources of effects

$$\mathsf{raise} : \mathsf{Exc} \longrightarrow \mathsf{0} \qquad \mathsf{get} : \mathsf{Loc} \longrightarrow \mathsf{Val} \qquad \mathsf{put} : \mathsf{Loc} \times \mathsf{Val} \longrightarrow \mathsf{1}$$

equations – describing the computational behaviour

$$\ell$$
:Loc |  $w:1 \vdash \text{get}_{\ell}(x.\text{put}_{\langle \ell, \mathsf{x} \rangle}(w(\star))) = w(\star)$ 

- The algebraic approach significantly simplifies
  - choosing a monad/adjunction to model a given language
  - modelling combinations of two or more comp. effects
  - generic effectful programming (via handlers)

• Moggi taught us to model comp. effects using **monads**  $(T, \eta, (-)^{\dagger})$ 

$$\eta_A:A \to TA$$
  $(f:A \to TB)^{\dagger}_{A,B}:TA \to TB$ 

- Plotkin and Power showed that most of these monads arise from
  - operation symbols representing the sources of effects

$$\mathsf{raise} : \mathsf{Exc} \longrightarrow \mathsf{0} \qquad \mathsf{get} : \mathsf{Loc} \longrightarrow \mathsf{Val} \qquad \mathsf{put} : \mathsf{Loc} \times \mathsf{Val} \longrightarrow \mathsf{1}$$

• equations – describing the computational behaviour

$$\ell$$
:Loc |  $w:1 \vdash \text{get}_{\ell}(x.\text{put}_{\langle \ell, x \rangle}(w(\star))) = w(\star)$ 

- The algebraic approach significantly simplifies
  - choosing a monad/adjunction to model a given language
  - modelling combinations of two or more comp. effects
  - generic effectful programming (via handlers)

- Plotkin and Pretnar's handlers of algebraic effects
  - generalisation of exception handlers
  - given by redefining the given ops. (handlers denote algebras)
  - many uses rollbacks, stream redirection, concurrency, ...
- Usually included in languages using the handling construct

```
M handled with \{\operatorname{op}_{X_v}(x_k)\mapsto N_{\operatorname{op}}\}_{\operatorname{op}\in\mathcal{S}_{\operatorname{eff}}} to y\colon A in C N_{\operatorname{ret}} interpreted using the homomorphism FA \longrightarrow \langle U\underline{C}, \overline{N_{\operatorname{op}}}\rangle, i.e (\operatorname{op}_V(y.M)) handled with \{\ldots\}_{\operatorname{op}\in\mathcal{S}_{\operatorname{eff}}} to y\colon A in C N_{\operatorname{ret}} = N_{\operatorname{op}}[V/x_v][\lambda\,y\colon O. thunk (M handled with \ldots)/x_k] and
```

6/2

- Plotkin and Pretnar's handlers of algebraic effects
  - generalisation of exception handlers
  - given by redefining the given ops. (handlers denote algebras)
  - many uses rollbacks, stream redirection, concurrency, ...
- Usually included in languages using the handling construct

```
M handled with \{\operatorname{op}_{\mathsf{X}_{\mathsf{V}}}(\mathsf{x}_k)\mapsto \mathsf{N}_{\operatorname{op}}\}_{\operatorname{op}\in\mathcal{S}_{\operatorname{eff}}} to y:A in C N_{\operatorname{ret}} interpreted using the homomorphism FA\longrightarrow \langle UC,\overline{N}_{\operatorname{op}}\rangle, i.e. (\operatorname{op}_V(y.M)) handled with \{\ldots\}_{\operatorname{op}\in\mathcal{S}_{\operatorname{eff}}} to y:A in C N_{\operatorname{ret}} = N_{\operatorname{op}}[V/\mathsf{x}_V][\lambda\,y:O thunk (M handled with \ldots)/\mathsf{x}_k] and
```

- Plotkin and Pretnar's handlers of algebraic effects
  - generalisation of exception handlers
  - given by redefining the given ops. (handlers denote algebras)
  - many uses rollbacks, stream redirection, concurrency, ...
- Usually included in languages using the handling construct

```
M handled with \{\operatorname{op}_{x_{v}}(x_{k}) \mapsto N_{\operatorname{op}}\}_{\operatorname{op} \in \mathcal{S}_{\operatorname{eff}}} to y: A \operatorname{in}_{\mathcal{C}} N_{\operatorname{ret}}
```

interpreted using the **homomorphism**  $FA \longrightarrow \langle U\underline{C}, \overrightarrow{N_{\mathrm{op}}} \rangle$ , i.e.,

$$(\operatorname{op}_V(y.M))$$
 handled with  $\{\ldots\}_{\operatorname{op} \in \mathcal{S}_{\operatorname{eff}}}$  to  $y:A$  in  $\underline{C}$   $N_{\operatorname{ret}}$ 

$$N_{\rm op}[V/x_v][\lambda\,y:O.\,{\rm thunk}\,(M\,\,{\rm handled}\,\,{\rm with}\,\,\ldots)/x_k]$$

and

 $(\text{return }V) \text{ handled with } \{\ldots\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in}_{\underline{C}} N_{\text{ret}} = N_{\text{ret}}[V/y]$ 

- Plotkin and Pretnar's handlers of algebraic effects
  - generalisation of exception handlers
  - given by redefining the given ops. (handlers denote algebras)
  - many uses rollbacks, stream redirection, concurrency, ...
- Usually included in languages using the **handling** construct

```
M handled with \{\operatorname{op}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{\mathsf{k}})\mapsto \mathsf{N}_{\operatorname{op}}\}_{\operatorname{op}\in\mathcal{S}_{\operatorname{eff}}} to y\colon A in \underline{c} \mathsf{N}_{\operatorname{ret}}
```

interpreted using the **homomorphism**  $FA \longrightarrow \langle U\underline{C}, \overrightarrow{N_{op}} \rangle$ , i.e.,

$$(op_V(y.M))$$
 handled with  $\{...\}_{op \in S_{eff}}$  to  $y:A$  in  $\underline{C}$   $N_{ret}$ 

$$N_{\mathrm{op}}[V/x_{\mathrm{v}}][\lambda\,y\!:\!O\,.\,\mathrm{thunk}\,(M\,\,\mathrm{handled}\,\,\mathrm{with}\,\,\ldots)/x_{k}]$$

and

```
(\text{return }V) handled with \{\ldots\}_{\text{op}\in\mathcal{S}_{\text{eff}}} to y:A in \underline{C} N_{\text{ret}}=N_{\text{ret}}[V/y]
```

#### **Outline**

- Setting the scene
  - Algebraic effects and their handlers
  - A core effectful dependently typed calculus (FoSSaCS'16)

[A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Modular programming with handlers + expressiveness of d. types
  - Extrinsic reasoning about effectful computations
- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A new type-level treatment of handlers

- (Model-theoretically) natural extension of type theory
  - clear distinction between values and computations (CBPV, EEC)
- Value types  $(\Gamma \vdash A)$  and computation types  $(\Gamma \vdash \underline{C})$

$$A,B ::= \ldots \mid U\underline{C} \quad \underline{C},\underline{D} ::= FA \mid \Pi x : A . \underline{C} \mid [\Sigma x : A . \underline{C}]$$

- Value terms  $(\Gamma \vdash V : A)$ 
  - $V, W ::= \dots \mid \text{thunk } M$
- Computation terms  $(\Gamma \vdash M : \underline{C})$

• Homomorphism terms  $(\Gamma \mid z : \underline{C} \vdash K : \underline{D})$  $K, L ::= z \mid K \text{ to } x : A \text{ in}_{\underline{C}} M \mid \dots$  (s

- (Model-theoretically) natural extension of type theory
  - clear distinction between values and computations (CBPV, EEC)
- Value types  $(\Gamma \vdash A)$  and computation types  $(\Gamma \vdash \underline{C})$

$$A,B ::= \dots \mid U\underline{C} \qquad \underline{C},\underline{D} ::= FA \mid \Pi x : A \cdot \underline{C} \mid \boxed{\Sigma x : A \cdot \underline{C}}$$

• Value terms  $(\Gamma \vdash V : A)$ 

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

• Computation terms  $(\Gamma \vdash M : \underline{C})$ 

• Homomorphism terms  $(\Gamma \mid z : \underline{C} \vdash K : \underline{D})$ 

$$K,L ::= Z \mid K \text{ to } X : A \text{ in}_{\underline{C}} M \mid \dots \quad \text{(stack terms, eval. ctxx}$$

- (Model-theoretically) natural extension of type theory
  - clear distinction between values and computations (CBPV, EEC)
- Value types  $(\Gamma \vdash A)$  and computation types  $(\Gamma \vdash \underline{C})$

$$A,B ::= \ldots \mid U\underline{C} \qquad \underline{C},\underline{D} ::= FA \mid \Pi x : A \cdot \underline{C} \mid \boxed{\Sigma x : A \cdot \underline{C}}$$

• Value terms  $(\Gamma \vdash V : A)$ 

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

• Computation terms  $(\Gamma \vdash M : \underline{C})$ 

• Homomorphism terms  $(\Gamma \mid z : \underline{C} \vdash K : \underline{D})$ 

 $K,L ::= z \mid K \text{ to } x : A \text{ in}_{\underline{C}} M \mid \dots$  (stack terms, eval. cbsss.)

- (Model-theoretically) natural extension of type theory
  - clear distinction between values and computations (CBPV, EEC)
- Value types  $(\Gamma \vdash A)$  and computation types  $(\Gamma \vdash \underline{C})$

$$A,B ::= \dots \mid U\underline{C} \qquad \underline{C},\underline{D} ::= FA \mid \Pi x : A . \underline{C} \mid \boxed{\Sigma x : A . \underline{C}}$$

• **Value terms** (Γ ⊢ *V* : *A*)

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

• Computation terms  $(\Gamma \vdash M : \underline{C})$ 

```
M, N ::= \operatorname{return} V \mid M \text{ to } x : A \text{ in}_{\underline{C}} N \mid \lambda x : A . M \mid M V \mid \langle V, M \rangle \mid M \text{ to } (x : A, z : \underline{C}) \text{ in}_{\underline{D}} K \mid \operatorname{force}_{\underline{C}} V
```

• Homomorphism terms  $(\Gamma \mid z : \underline{C} \vdash K : \underline{D})$ 

 $K,L ::= z \mid K \text{ to } x : A \text{ in}_{\underline{C}} M \mid \dots$  (stack terms, eval. ctxss.)

- (Model-theoretically) natural extension of type theory
  - clear distinction between values and computations (CBPV, EEC)
- Value types  $(\Gamma \vdash A)$  and computation types  $(\Gamma \vdash \underline{C})$  $A, B ::= \dots \mid U\underline{C} \quad \underline{C}, \underline{D} ::= FA \mid \Pi x : A . \underline{C} \mid [\Sigma x : A . \underline{C}]$
- Value terms (Γ ⊢ V : A)

  V, W ::= ... | thunk M
- Computation terms  $(\Gamma \vdash M : \underline{C})$
- $M, N ::= \operatorname{return} V \mid M \operatorname{to} x : A \operatorname{in}_{\underline{C}} N \mid \lambda x : A . M \mid M V$   $\mid \langle V, M \rangle \mid M \operatorname{to} (x : A, z : \underline{C}) \operatorname{in}_{D} K \mid \operatorname{force}_{C} V$
- Homomorphism terms  $(\Gamma \mid z : \underline{C} \vdash K : \underline{D})$
- $K, L := z \mid K \text{ to } x : A \text{ in}_{\underline{C}} M \mid \dots \text{ (stack terms, eval. ctxs.)}$

#### **Outline**

- Setting the scene
  - Algebraic effects and their handlers
  - A core effectful dependently typed calculus (FoSSaCS'16)

[A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Modular programming with handlers + expressiveness of d. types
  - Extrinsic reasoning about effectful computations
- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A new type-level treatment of handlers

#### The calculus we work in

- We work in an extension to the FoSSaCS'16 calculus, with
  - ullet a Tarski-style value universe  ${\cal U}$ 
    - with **codes** written as  $\widehat{\Pi}$ ,  $\widehat{\Sigma}$ ,  $\widehat{0}$ ,  $\widehat{1}$ , ...
    - but thinking of them as  $\forall$ ,  $\exists$ ,  $\bot$ ,  $\top$ , ...
  - fibred algebraic effects
    - dep. typed **operation symbols** op :  $(x_v:I) \longrightarrow O(x_v)$
    - ops. determine **comp. terms** op $\frac{C}{V}(y:O[V/x_v].M)$
    - effect eqs. determine definitional eqs.
  - a derivable "into-comps." variant of handlers and handling

$${\it M}$$
 handled with  $\{{\it op}_{x_v}(x_k)\mapsto {\it N}_{\it op}; \overrightarrow{W_{\it eq}}\}_{\it op}\in {\it S}_{\it eff}$  to  $y\!:\!{\it A}$  in  $\underline{\it C}$   ${\it N}_{\it ret}$ 

• a derivable "into-values" variant of handlers and handling

$$extcolor{M}$$
 handled with  $\{\operatorname{op}_{\mathsf{x}_v}(\mathsf{x}_k)\mapsto V_{\operatorname{op}}; \overrightarrow{W_{\operatorname{eq}}}\}_{\operatorname{op}\,\in\,\mathcal{S}_{\operatorname{eff}}}$  to  $y\!:\!A$  in  $_BV_{\operatorname{ret}}$ 

### Reasoning about effectful computations

- Handlers are useful in various ways!
- They enable **extrinsic reasoning** about computations *M* : *FA* 
  - ullet Can be used to define **predicates**  $P: \mathit{UFA} \to \mathcal{U}$  by
    - 1) equipping  $\mathcal{U}$  (or a resp. type) with an algebra structure
    - 2) handling the given computation using that algebra
  - Intuitively, P (thunk M) computes a proof obligation for M
  - We discuss three examples of such predicates
- Also, an alternative to mon. reification for rel. reasoning
  - E.g., relating stateful comps. M: FA as functions  $S \to A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for reification-based rel. reasoning

#### Reasoning about effectful computations

- Handlers are useful in various ways!
- They enable extrinsic reasoning about computations M : FA
  - Can be used to define **predicates**  $P: UFA \rightarrow \mathcal{U}$  by
    - 1) equipping  $\mathcal{U}$  (or a resp. type) with an **algebra** structure
    - 2) handling the given computation using that algebra
  - Intuitively, P (thunk M) computes a proof obligation for M
  - We discuss three examples of such predicates
- Also, an alternative to mon. reification for rel. reasoning
  - E.g., relating stateful comps. M: FA as functions  $S \to A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for reification-based rel. reasoning

#### Reasoning about effectful computations

- Handlers are **useful** in various ways!
- They enable extrinsic reasoning about computations M : FA
  - Can be used to define **predicates**  $P: UFA \rightarrow \mathcal{U}$  by
    - 1) equipping  $\mathcal{U}$  (or a resp. type) with an **algebra** structure
    - 2) handling the given computation using that algebra
  - Intuitively, P (thunk M) computes a proof obligation for M
  - We discuss three examples of such predicates
- Also, an alternative to mon. reification for rel. reasoning
  - E.g., relating stateful comps. M: FA as functions  $S \to A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for reification-based rel. reasoning

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

$$\operatorname{read}(x_k) \mapsto \widehat{\Pi} y : \operatorname{El}(\widehat{\operatorname{Chr}}) . x_k y \qquad (\text{where } x_k : \operatorname{Chr} \to \mathcal{U})$$

$$\operatorname{write}_{x_v}(x_k) \mapsto x_k \star \qquad (\text{where } x_v : \operatorname{Chr}, \ x_k : 1 \to \mathcal{U})$$

• LP is similar to the **necessity modality** from Evaluation Logic

$$\Gamma \vdash \Box P \text{ (thunk (read(x, write_w(return V))))} = \widehat{\Pi} x : El(\widehat{Chr}) \cdot P V$$

To get  $\Diamond P$ , we only have to replace  $\widehat{\Pi}$  with  $\widehat{\Sigma}$  in the handler

• Given a predicate  $P: A \rightarrow \mathcal{U}$  on **return values**,

we define a predicate  $\Box P: \mathit{UFA} \to \mathcal{U}$  on (I/O)-comps. as

$$\Box P \stackrel{\mathsf{def}}{=} \lambda \, y \colon UFA \, . \, (\mathsf{force} \, y) \, \, \mathsf{handled} \, \, \mathsf{with} \, \, \{ \ldots \}_{\mathsf{op} \in \mathcal{S}_{\mathsf{I/O}}} \, \, \mathsf{to} \, \, y' \colon A \, \, \mathsf{in}_{\,\mathcal{U}} \, P \, y$$
 using the **handler** given by 
$$\mathsf{read}(x_k) \quad \mapsto \quad \widehat{\Pi} \, y \colon \mathsf{El}(\widehat{\mathsf{Chr}}) \, . \, x_k \, y \qquad \qquad (\mathsf{where} \, x_k \colon \mathsf{Chr} \to \mathcal{U})$$
 
$$\mathsf{write}_{x_k}(x_k) \quad \mapsto \quad x_k \, \star \qquad \qquad (\mathsf{where} \, x_k \colon \mathsf{Chr}, \, \, x_k \colon 1 \to \mathcal{U})$$

ullet Is similar to the **necessity modality** from Evaluation Logic

$$\Gamma \vdash \Box P \left( \text{thunk} \left( \text{read}(x.\text{write}_{e'}(\text{return } V)) \right) \right) = \widehat{\Pi} x : \widehat{\text{El}(\widehat{\mathsf{Chr}})} . P V$$

• To get  $\Diamond P$ , we only have to replace  $\Pi$  with  $\Sigma$  in the handler

• Given a predicate  $P: A \rightarrow \mathcal{U}$  on **return values**,

we define a predicate  $\Box P: \mathit{UFA} \to \mathcal{U}$  on (I/O)-comps. as

 $\Box P \stackrel{\text{def}}{=} \lambda y : UFA . \text{ (force } y) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{S}_{\text{I/O}}} \text{ to } y' : A \text{ in}_{\mathcal{U}} P y'$  using the **handler** given by

$$\mathsf{read}(x_k) \quad \mapsto \quad \widehat{\mathsf{\Pi}} \, y \colon \mathsf{El}(\widehat{\mathsf{Chr}}) \, . \, x_k \, y \qquad \qquad (\mathsf{where} \, x_k \colon \mathsf{Chr} \to \mathcal{U})$$

$$\mathsf{write}_{x_v}(x_k) \quad \mapsto \quad x_k \, \star \qquad \qquad (\mathsf{where} \, x_v \colon \mathsf{Chr}, \, x_k \colon 1 \to \mathcal{U})$$

ullet Is similar to the **necessity modality** from Evaluation Logic

$$\Gamma \vdash \Box P (\text{thunk}(\text{read}(x.\text{write}_{e'}(\text{return }V)))) = \widehat{\Pi} x : El(\widehat{\text{Chr}}).PV$$

• To get  $\Diamond P$ , we only have to replace  $\Pi$  with  $\Sigma$  in the handler

Given a predicate P: A → U on return values,
 we define a predicate □P: UFA → U on (I/O)-comps. as

$$\Box P \stackrel{\text{def}}{=} \lambda y : \textit{UFA} . (\texttt{force} \ y) \ \texttt{handled} \ \texttt{with} \ \{ \ldots \}_{\texttt{op} \in \mathcal{S}_{\mathsf{I/O}}} \ \texttt{to} \ y' : A \ \texttt{in}_{\mathcal{U}} \ P \ y'$$
 using the **handler** given by

```
\operatorname{\mathsf{read}}(x_k) \mapsto \widehat{\mathsf{\Pi}} \, y \colon \mathsf{El}(\widehat{\mathsf{Chr}}) \, . \, x_k \, y \qquad \qquad (\mathsf{where} \, x_k \colon \mathsf{Chr} \to \mathcal{U})
\mathsf{write}_{x_v}(x_k) \mapsto x_k \, \star \qquad \qquad (\mathsf{where} \, x_v \colon \mathsf{Chr}, \, x_k \colon 1 \to \mathcal{U})
```

•  $\square P$  is similar to the **necessity modality** from Evaluation Logic

$$\Gamma \vdash \Box P \left( \text{thunk} \left( \text{read}(x . \text{write}_{e'}(\text{return } V)) \right) \right) = \widehat{\Pi} x : El(\widehat{Chr}) . P V$$

• To get  $\Diamond P$ , we only have to replace  $\widehat{\Pi}$  with  $\widehat{\Sigma}$  in the handler

Given a postcondition on return values and final states

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

we define a precondition for stateful comps. on initial states

$$\mathsf{wp}_{\mathcal{Q}}: \mathit{UFA} o \mathit{S} o \mathcal{U}$$

by

$$V_{\mathrm{get}}$$
 ,  $V_{\mathrm{put}}$  on  $S \to \mathcal{U} \times S$  and  $V_{\mathrm{ret}}$  "="  $G$ 

- **2)** feeding in the **initial state**; and **3)** projecting out the **value of**  $\mathcal{U}$
- Then, wp<sub>Q</sub> satisfies the expected properties, such as

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{return} \, V)) = \lambda \, x_S \colon S \cdot Q \, V \, x_S$$

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{put}_{(\ell, V)}(M))) = \lambda \, x_S \colon S \cdot \mathsf{wp}_Q \; (\mathsf{thunk} \, M) \, x_S[\ell \mapsto V]$$

• Given a postcondition on return values and final states

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

we define a precondition for stateful comps. on initial states

$$\mathsf{wp}_{\mathcal{O}}: \mathit{UFA} \to \mathit{S} \to \mathcal{U}$$

by

$$V_{\mathrm{get}}$$
 ,  $V_{\mathrm{put}}$  on  $S \to \mathcal{U} \times S$  and  $V_{\mathrm{ret}}$  "="  $Q$ 

- 2) feeding in the initial state; and 3) projecting out the value of  $\mathcal U$
- Then, wp<sub>O</sub> satisfies the expected properties, such as

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{return} \, V)) = \lambda \, x_S \colon S \cdot Q \, V \, x_S$$

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{put}_{\langle \ell, V \rangle}(M))) = \lambda \, x_S \colon S \cdot \mathsf{wp}_Q \; (\mathsf{thunk} \, M) \, x_S[\ell \mapsto V]$$

• Given a postcondition on return values and final states

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

we define a precondition for stateful comps. on initial states

$$\mathsf{wp}_{\mathcal{O}}: \mathit{UFA} \to \mathcal{S} \to \mathcal{U}$$

by

$$V_{\mathsf{get}} \,,\, V_{\mathsf{put}} \quad \mathsf{on} \quad S o \mathcal{U} imes S \qquad \qquad \mathsf{and} \qquad \qquad V_{\mathsf{ret}} \ \text{``='} \ \ \mathcal{Q}$$

- 2) feeding in the **initial state**; and 3) projecting out the **value of**  $\mathcal{U}$
- Then, wp<sub>O</sub> satisfies the expected properties, such as

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{return} \, V)) = \lambda \, x_S \colon S \cdot Q \; V \; x_S$$

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{put}_{\ell \in V}(M))) = \lambda \, x_S \colon S \cdot \mathsf{wp}_Q \; (\mathsf{thunk} \, M) \; x_S[\ell \mapsto V]$$

• Given a postcondition on return values and final states

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

we define a precondition for  ${\bf stateful}\ {\bf comps.}$  on  ${\bf initial}\ {\bf states}$ 

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

by

$$V_{\mathsf{get}}\,,\,V_{\mathsf{put}}$$
 on  $S \to \mathcal{U} \times S$  and  $V_{\mathsf{ret}}$  "="  $Q$ 

- 2) feeding in the initial state; and 3) projecting out the value of  $\mathcal U$
- Then, wp o satisfies the **expected properties**, such as

$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{return} \, {\color{red} V})) \; = \; \lambda \, x_S \colon S \cdot Q \; {\color{red} V} \; x_S$$
 
$$\Gamma \vdash \mathsf{wp}_Q \; (\mathsf{thunk} \, (\mathsf{put}_{\langle \ell, \, V \rangle}(M))) \; = \; \lambda \, x_S \colon S \cdot \mathsf{wp}_Q \; (\mathsf{thunk} \, M) \; x_S[\ell \mapsto {\color{red} V}]$$

# Ex3: Allowed patterns of (I/O)-effects

Assuming an inductive type of I/O-protocols, given by

e : Protocol 
$$\mathbf{r}: (\mathsf{Chr} \to \mathsf{Protocol}) \to \mathsf{Protocol}$$
  
 $\mathbf{w}: (\mathsf{Chr} \to \mathcal{U}) \times \mathsf{Protocol} \to \mathsf{Protocol}$ 

• We can define a **relation** between **comps.** and **protocols** 

Allowed : 
$$\mathit{UFA} o \mathsf{Protocol} o \mathcal{U}$$

by handling the given computation using a handler on

$$\mathsf{Protocol} o \mathcal{U}$$

given by (using pattern-matching lambda notation)

read
$$(x_k)$$
  $\mapsto \lambda \{(\mathbf{r} x_{pr}) \to \widehat{\Pi} y : El(\widehat{\mathsf{Chr}}) . x_k y (x_{pr} y) ; \to \widehat{0} \}$ 

$$\mathsf{write}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{k}) \quad \mapsto \quad \lambda \left\{ \left( \mathsf{w} \ P \ \mathsf{x}_{\mathsf{pr}} \right) \to \widehat{\Sigma} \ \mathsf{y} : \mathsf{El}(P \ \mathsf{x}_{\mathsf{v}}) \, . \, \mathsf{x}_{k} \ \star \ \mathsf{x}_{\mathsf{pr}} \ ; \right. \\ \left. \to \widehat{\mathsf{o}} \ \right\}$$

# Ex3: Allowed patterns of (I/O)-effects

• Assuming an inductive type of I/O-protocols, given by

$$\begin{tabular}{ll} \textbf{e} : Protocol & \textbf{r} : (Chr \rightarrow Protocol) \rightarrow Protocol \\ & \textbf{w} : (Chr \rightarrow \mathcal{U}) \times Protocol \rightarrow Protocol \\ \end{tabular}$$

We can define a relation between comps. and protocols

Allowed : 
$$UFA o \mathsf{Protocol} o \mathcal{U}$$

by handling the given computation using a handler on

$$\mathsf{Protocol} o \mathcal{U}$$

given by (using pattern-matching lambda notation)

read
$$(x_k)$$
  $\mapsto \lambda \{(\mathbf{r} \ x_{pr}) \rightarrow \widehat{\Pi} \ y : El(\widehat{\mathsf{Chr}}) . \ x_k \ y \ (x_{pr} \ y) ; \\ - \rightarrow \widehat{0} \}$ 

$$\operatorname{write}_{x_{v}}(x_{k}) \mapsto \lambda \left\{ \left( w P x_{pr} \right) \to \widehat{\Sigma} y : \operatorname{El}(P x_{v}) . x_{k} \star x_{pr} ; \right.$$

$$\left. - \widehat{0} \right\}$$

#### Ex3: Allowed patterns of (I/O)-effects

• Assuming an inductive type of I/O-protocols, given by

$$\begin{tabular}{ll} \bf e: Protocol & \bf r: (Chr \rightarrow Protocol) \rightarrow Protocol \\ \hline \bf w: (Chr \rightarrow \mathcal{U}) \times Protocol \rightarrow Protocol \\ \end{tabular}$$

• We can define a **relation** between **comps.** and **protocols** 

Allowed : 
$$UFA \rightarrow Protocol \rightarrow \mathcal{U}$$

by handling the given computation using a **handler** on

$$\mathsf{Protocol} o \mathcal{U}$$

given by (using pattern-matching lambda notation)

read
$$(x_k)$$
  $\mapsto$   $\lambda \{(\mathbf{r} \ x_{pr}) \rightarrow \widehat{\Pi} \ y : \widehat{\mathsf{El}}(\widehat{\mathsf{Chr}}) . x_k \ y \ (x_{pr} \ y) ;$ 

$$\qquad \qquad \qquad \rightarrow \widehat{0} \}$$

$$\mathsf{write}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{\mathsf{k}}) \;\; \mapsto \;\; \lambda \left\{ \left( \mathsf{w} \; P \; \mathsf{x}_{\mathsf{pr}} \right) \to \widehat{\Sigma} \; \mathsf{y} : \mathsf{El}(P \; \mathsf{x}_{\mathsf{v}}) \, . \, \mathsf{x}_{\mathsf{k}} \; \star \; \mathsf{x}_{\mathsf{pr}} \; ; \right.$$

#### **Outline**

- Setting the scene
  - Algebraic effects and their handlers
  - A core effectful dependently typed calculus (FoSSaCS'16)
     [A., Ghani, Plotkin'16]
- What can we gain from handlers + dependent types?
  - Modular programming with handlers + expressiveness of d. types
  - Extrinsic reasoning about effectful computations
- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A new type-level treatment of handlers

### Extending the FoSSaCS'16 calculus

- We assume given a **fibred effect theory**  $\mathcal{T} = (\mathcal{S}, \mathcal{E})$
- First, we extend the calculus with algebraic effects as follows:
  - we extend the computation terms with

$$M, N ::= \ldots \mid \operatorname{op}_{V}^{\underline{C}}(y : \mathcal{O}[V/x_{v}] \cdot M) \quad (\operatorname{op} : (x_{v} : t) \longrightarrow \mathcal{O} \in \mathcal{S})$$

- ullet we extend the **equational theory** with equations given in  ${\mathcal E}$
- we capture the interaction of comp. terms and ops. with the eq

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

Second, we extend the calculus with a support for handlers . . .

## Extending the FoSSaCS'16 calculus

- We assume given a **fibred effect theory**  $\mathcal{T} = (\mathcal{S}, \mathcal{E})$
- First, we extend the calculus with algebraic effects as follows:
  - we extend the computation terms with

$$M, N ::= \ldots \mid \operatorname{op}_{\overline{V}}^{\underline{C}}(y : O[V/x_v] . M) \quad (\operatorname{op} : (x_v : I) \longrightarrow O \in S)$$

- ullet we extend the **equational theory** with equations given in  ${\mathcal E}$
- we capture the interaction of comp. terms and ops. with the eq.

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

• Second, we extend the calculus with a support for handlers

## Extending the FoSSaCS'16 calculus

- ullet We assume given a **fibred effect theory**  $\mathcal{T}=(\mathcal{S},\mathcal{E})$
- First, we extend the calculus with algebraic effects as follows:
  - we extend the computation terms with

$$M, N ::= \ldots \mid \operatorname{op}_{\overline{V}}^{\underline{C}}(y : O[V/x_v] . M) \quad (\operatorname{op} : (x_v : I) \longrightarrow O \in S)$$

- ullet we extend the **equational theory** with equations given in  ${\mathcal E}$
- we capture the interaction of comp. terms and ops. with the eq.

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

• Second, we extend the calculus with a support for handlers . . .

Begin by extending the FoSSaCS'16 computation terms with

```
M,N ::= \ldots \mid M \text{ handled with } \{ \operatorname{op}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{\mathsf{k}}) \mapsto \mathsf{N}_{\operatorname{op}} \}_{\operatorname{op} \in \mathcal{S}_{\operatorname{eff}}} \text{ to } y : A \text{ in}_{\underline{C}} \ \mathsf{N}_{\operatorname{ret}} \}
```

• But as handling denotes a **homomorphism**, then perhaps also

$$\Gamma \vdash \text{write}_{a}(\text{return} \star) = \text{write}_{z}(\text{return} \star) : F1$$

- At a very high-level, the problem is (see the paper for details)
  - interaction between Ks and ops. is governed by comp. types
  - but the type of handled with does not mention the handler

• Begin by extending the FoSSaCS'16 computation terms with

```
M,N ::= \ldots \mid M \text{ handled with } \{ \operatorname{op}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{\mathsf{k}}) \mapsto \mathsf{N}_{\operatorname{op}} \}_{\operatorname{op} \in \mathcal{S}_{\operatorname{eff}}} \text{ to } y : A \text{ in}_{\underline{C}} \ \mathsf{N}_{\operatorname{ret}} \}
```

• But as handling denotes a **homomorphism**, then perhaps also

$$\begin{cases} \textit{K},\textit{L} \end{cases} ::= \end{cases} \begin{cases} \textit{K} \end{cases} \begin{cases} \textit{h} \end{cases} \begin{cases} \textit{h} \end{cases} \begin{cases} \textit{op} \end{cases} \begin{cases} \textit{N} \end{cases} \begin{cases} \textit{op} \end{cases} \begin{cases} \textit{N} \end{cases} \begin{cases} \textit{op} \end{cases} \begin{cases}$$

However, this leads to an inconsistent system, e.g.,

$$\Gamma \vdash \text{write}_{a}(\text{return} *) = \text{write}_{z}(\text{return} *) : F1$$

- At a very high-level, the problem is (see the paper for details).
  - interaction between Ks and ops. is governed by comp. types
  - but the type of handled with does not mention the handler

• Begin by extending the FoSSaCS'16 computation terms with

```
M,N ::= \ldots \mid M \text{ handled with } \{ \operatorname{op}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{\mathsf{k}}) \mapsto \mathsf{N}_{\operatorname{op}} \}_{\operatorname{op} \in \mathcal{S}_{\operatorname{eff}}} \text{ to } y : A \text{ in}_{\underline{C}} \ \mathsf{N}_{\operatorname{ret}} \}
```

• But as handling denotes a **homomorphism**, then perhaps also

$$\begin{cases} \textit{K},\textit{L} ::= & \dots & | & \textit{K} \mbox{ handled with } \{ \mbox{op}_{x_{\nu}}(x_k) \mapsto \textit{N}_{\mbox{op}} \}_{\mbox{op} \in \mathcal{S}_{\mbox{eff}}} \mbox{ to } y \colon \! A \mbox{ in}_{\underline{C}} \mbox{ $N_{\mbox{ret}}$} \mbox{ ret} \mbox{ } \mbox{op} \}_{\mbox{op} \in \mathcal{S}_{\mbox{eff}}} \mbox{ to } y \colon \! A \mbox{ in}_{\underline{C}} \mbox{ $N_{\mbox{ret}}$} \mbox{ } \mbox{ } \mbox{op} \}_{\mbox{op} \in \mathcal{S}_{\mbox{eff}}} \mbox{ to } y \colon \! A \mbox{ in}_{\underline{C}} \mbox{ } \mbox{op} \mbox{op} \mbox{ } \mbox{ } \mbox{op} \mbox{ } \mbox{op} \mbox{ } \mbox{op} \mbox{ } \mbox{ } \mbox{ } \mbox{op} \mbox{ } \mbox{ } \mbox{op} \mbox{op} \mbox{op} \mbox$$

However, this leads to an inconsistent system, e.g.,

$$\Gamma \vdash \text{write}_{\mathbf{a}}(\text{return} \star) = \text{write}_{\mathbf{z}}(\text{return} \star) : F1$$

- At a very high-level, the problem is (see the paper for details)
  - interaction between Ks and ops. is governed by comp. types
  - but the type of handled with does not mention the handler

• Begin by extending the FoSSaCS'16 computation terms with

```
M,N ::= \ldots \mid M \text{ handled with } \{ \operatorname{op}_{x_v}(x_k) \mapsto N_{\operatorname{op}} \}_{\operatorname{op} \in \mathcal{S}_{\operatorname{eff}}} \text{ to } y : A \operatorname{in}_{\underline{C}} N_{\operatorname{ret}}
```

But as handling denotes a homomorphism, then perhaps also

$${\color{black} {\it K}}, {\color{black} {\it L}} ::= \ldots \mid {\color{black} {\it K}} \; {\rm handled} \; {\rm with} \; \{{\rm op}_{{\rm x}_{\rm v}}(x_k) \mapsto {\color{black} {\it N}}_{{\rm op}}\}_{{\rm op} \; \in \; {\cal S}_{{\rm eff}}} \; {\rm to} \; y \, : {\color{black} {\it A}} \; {\rm in}_{\underline{{\it C}}} \; {\color{black} {\it N}}_{{\rm ret}}$$

• However, this leads to an **inconsistent** system, e.g.,

$$\Gamma \vdash \mathsf{write}_{\mathsf{a}}(\mathsf{return}\,\star) = \mathsf{write}_{\mathsf{z}}(\mathsf{return}\,\star) : F1$$

- At a very high-level, the problem is (see the paper for details)
  - interaction between Ks and ops. is governed by comp. types
  - but the type of handled with does not mention the handler

### How to proceed?

- Possible ways to solve this unsoundness problem
  - Option 1: Change the FoSSaCS'16 calculus
    - change the equational theory of homomorphism terms
    - hom. terms would not denote homomorphisms any more
    - investigated for exceptions in CBPV with stacks by [Levy'06]
  - Option 2: Keep the FoSSaCS'16 calculus unchanged
    - extend it so that handling for comp. terms is derivable
    - while making sure that the calculus remains sound
    - key idea: comp. types and handlers both denote algebras
    - extended calculus admits a natural denotational semantics

### How to proceed?

- Possible ways to solve this unsoundness problem
  - Option 1: Change the FoSSaCS'16 calculus
    - change the equational theory of homomorphism terms
    - hom. terms would not denote homomorphisms any more
    - investigated for exceptions in CBPV with stacks by [Levy'06]
  - Option 2: Keep the FoSSaCS'16 calculus unchanged
    - extend it so that handling for comp. terms is derivable
    - while making sure that the calculus remains sound
    - key idea: comp. types and handlers both denote algebras
    - extended calculus admits a natural denotational semantics

#### How to proceed?

- Possible ways to solve this unsoundness problem
  - Option 1: Change the FoSSaCS'16 calculus
    - change the equational theory of homomorphism terms
    - hom. terms would not denote homomorphisms any more
    - investigated for exceptions in CBPV with stacks by [Levy'06]
  - Option 2: Keep the FoSSaCS'16 calculus unchanged
    - extend it so that handling for comp. terms is derivable
    - while making sure that the calculus remains sound
    - key idea: comp. types and handlers both denote algebras
    - extended calculus admits a natural denotational semantics

# Take 2: A type-level treatment of handlers

- Instead, we extend the FoSSaCS'16 computation types with
  - a user-defined algebra type

$$\underline{C},\underline{D} ::= \ldots \mid \langle A; \overrightarrow{V_{\mathsf{op}}}; \overrightarrow{W_{\mathsf{eq}}} \rangle$$

where

- A is the carrier value type
- $\overrightarrow{V_{\text{op}}}$  is a set of user-defined **operations**
- $\overrightarrow{W}_{eq}$  is a set of witnesses of equational proof obligations
- As a result, we can derive the handing construct as

$$M$$
 handled with  $\{\operatorname{op}_{x_v}(x_k)\mapsto N_{\operatorname{op}};W_{\operatorname{eq}}^{'}\}_{\operatorname{op}\in\mathcal{S}_{\operatorname{eff}}}$  to  $y:A$  in  $C$ 

 $\mathtt{force}_{\underline{C}}(\mathtt{thunk}\,(\underline{M}\,\mathtt{to}\,\,y\!:\!A\,\mathtt{in}\,\,\mathtt{force}_{(\underline{U}\underline{C};\overrightarrow{V_{N_{\mathrm{op}}}};\overline{W_{\mathrm{eq}}})}(\mathtt{thunk}\,(N_{\mathrm{ret}}))))$ 

temporarily working at type  $\langle U\underline{C}; \overline{V_{N_{op}}}; \overline{W'_{eq}} \rangle$ 

and similarly for the "**into-values**" variant of it

## Take 2: A type-level treatment of handlers

- Instead, we extend the FoSSaCS'16 computation types with
  - a user-defined algebra type

$$\underline{C},\underline{D} ::= \ldots \mid \langle A; \overrightarrow{V_{op}}; \overrightarrow{W_{eq}} \rangle$$

where

- A is the carrier value type
- $\overrightarrow{V_{\text{op}}}$  is a set of user-defined **operations**
- $\overrightarrow{W_{\text{eq}}}$  is a set of **witnesses** of equational proof obligations
- As a result, we can derive the handing construct as

$$M$$
 handled with  $\{\operatorname{op}_{x_{v}}(x_{k})\mapsto N_{\operatorname{op}}; \overrightarrow{W_{\operatorname{eq}}}\}_{\operatorname{op}\,\in\,\mathcal{S}_{\operatorname{eff}}}$  to  $y:A$  in  $\underline{c}$   $N_{\operatorname{ret}}$   $\stackrel{\operatorname{def}}{=}$  
$$\operatorname{force}_{\underline{C}}(\operatorname{thunk}(M\text{ to }y:A\text{ in }\operatorname{force}_{\underline{VUC};\overrightarrow{V_{\operatorname{Nop}}}:\overrightarrow{W_{\operatorname{eq}}}}(\operatorname{thunk}N_{\operatorname{ret}})))$$

temporarily working at type  $\langle U\underline{C}; \overrightarrow{V_{N_{op}}}; \overrightarrow{W_{eq}} \rangle$ 

# Take 2: A type-level treatment of handlers

- Instead, we extend the FoSSaCS'16 computation types with
  - a user-defined algebra type

$$\underline{C},\underline{D} ::= \ldots \mid \langle A; \overrightarrow{V_{\sf op}}; \overrightarrow{W_{\sf eq}} \rangle$$

where

- A is the carrier value type
- $\overrightarrow{V_{\text{op}}}$  is a set of user-defined **operations**
- $\overrightarrow{W}_{eq}$  is a set of **witnesses** of equational proof obligations
- As a result, we can derive the handing construct as

$$\begin{array}{c} M \text{ handled with } \{\operatorname{op}_{\mathsf{x}_{\mathsf{v}}}(\mathsf{x}_{k}) \mapsto \bigvee_{\mathsf{op}}; \overrightarrow{W_{\mathsf{eq}}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \text{ to } y \colon A \text{ in}_{\underline{C}} \bigvee_{\mathsf{ret}} \bigvee_{\mathsf{def}} \\ & \stackrel{\mathsf{def}}{=} \\ & \text{force}_{\underline{C}}(\mathsf{thunk}\left(\underbrace{M \text{ to } y \colon A \text{ in force}_{\langle U\underline{C}; \overrightarrow{V_{\mathsf{Nop}}}; \overrightarrow{W_{\mathsf{eq}}} \rangle}(\mathsf{thunk} \bigvee_{\mathsf{ret}})\right)) \\ & \xrightarrow{\mathsf{temporarily working at type} \langle U\underline{C}; \overrightarrow{V_{\mathsf{Nop}}}; \overrightarrow{W_{\mathsf{eq}}} \rangle} \end{array}$$

and similarly for the "into-values" variant of it

#### **Conclusion**

- In conclusion
  - handlers are natural for defining predicates on computations
    - lifting predicates from return values to computations
    - Dijkstra's weakest precondition semantics of state
    - specifying patterns of allowed (I/O)-effects
  - they admit a principled type-based treatment
- See the paper for
  - formal details of what I have shown you today
  - families fibrations based denotational semantics of the calculus
  - discussion about the calculus's inherent extensional nature
  - **Agda code** for the example predicates  $P: \mathit{UFA} \to \mathcal{U}$

Thank you!

Questions?