

Handling Fibred Algebraic Effects

Danel Ahman

INRIA Paris

POPL 2018

January 10, 2018

A Venn diagram consisting of two overlapping rounded rectangles. The left rectangle has a blue border and contains the text 'Dependent Types' and 'Programming + Logic' in blue. The right rectangle has a red border and contains the text 'Handlers of Algebraic Effects' and 'Modular Effectful Programming' in red. The overlapping area is the intersection of the two sets.

Dependent Types

Programming + Logic

Handlers of Algebraic Effects

Modular Effectful Programming



A Venn diagram consisting of two overlapping rounded rectangles. The left rectangle has a blue border and contains the text 'Dependent Types' and 'Programming + Logic'. The right rectangle has a red border and contains the text 'Handlers of Algebraic Effects' and 'Modular Effectful Programming'. The intersection of the two rectangles is highlighted by a black arrow pointing upwards from the text 'This Paper' below.

Dependent Types

Programming + Logic

Handlers of Algebraic Effects

Modular Effectful Programming

This Paper

Outline

- Setting the scene
 - **Algebraic effects** and their **handlers**
 - An effectful dependently typed **core 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 (unsoundness)
 - Take 2: A new **type-level treatment** of handlers

Outline

- Setting the scene
 - **Algebraic effects** and their **handlers**
 - An effectful dependently typed **core 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 (unsoundness)
 - Take 2: A new **type-level treatment** of handlers

Algebraic effects

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

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

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

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

- **equations** – describing the computational **behaviour**

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

- 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**)

Algebraic effects

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

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

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

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

- **equations** – describing the computational **behaviour**

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

- 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**)

Algebraic effects

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

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

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

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

- **equations** – describing the computational **behaviour**

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

- 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**)

Handlers of algebraic effects

- 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 $\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}}$ to $y:A \text{ in } \underline{C} \ N_{\text{ret}}$

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

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

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

and

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

Handlers of algebraic effects

- 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 $\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}}$ to $y:A \text{ in } \underline{C} \ N_{\text{ret}}$

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

$$\begin{aligned} (\text{op}_V(y.M)) \text{ handled with } \{\dots\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y:A \text{ in } \underline{C} \ N_{\text{ret}} \\ = \\ N_{\text{op}}[V/x_v][\lambda y:O. \text{thunk}(M \text{ handled with } \dots)/x_k] \end{aligned}$$

and

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

Handlers of algebraic effects

- 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 $\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}}$ to $y:A \text{ in } \underline{C} \ N_{\text{ret}}$

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

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

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

and

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

Handlers of algebraic effects

- 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 $\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}}$ to $y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

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

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

and

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

Outline

- Setting the scene
 - **Algebraic effects** and their **handlers**
 - An effectful dependently typed **core 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 (unsoundness)
 - Take 2: A new **type-level treatment** of handlers

A core dependently typed effectful calculus

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

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

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

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

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

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

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

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

A core dependently typed effectful calculus

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

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

- **Value terms** $(\Gamma \vdash V : A)$

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

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

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

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

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

A core dependently typed effectful calculus

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

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

- **Value terms** $(\Gamma \vdash V : A)$

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

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

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

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

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

A core dependently typed effectful calculus

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

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

- **Value terms** $(\Gamma \vdash V : A)$

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

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

$$\begin{aligned} M, N ::= & \text{return } V \mid M \text{ to } x:A \text{ in}_{\underline{C}} N \mid \lambda x:A. M \mid M V \\ & \mid \langle V, M \rangle \mid \boxed{M \text{ to } (x:A, z:\underline{C}) \text{ in}_{\underline{D}} K} \mid \text{force}_{\underline{C}} V \end{aligned}$$

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

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

A core dependently typed effectful calculus

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

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

- **Value terms** $(\Gamma \vdash V : A)$

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

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

$$\begin{aligned} M, N ::= & \text{return } V \mid M \text{ to } x:A \text{ in}_{\underline{C}} N \mid \lambda x:A. M \mid M V \\ & \mid \langle V, M \rangle \mid \boxed{M \text{ to } (x:A, z:\underline{C}) \text{ in}_{\underline{D}} K} \mid \text{force}_{\underline{C}} V \end{aligned}$$

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

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

Outline

- Setting the scene
 - **Algebraic effects** and their **handlers**
 - An effectful dependently typed **core 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 (unsoundness)
 - 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
 - a Tarski-style **value universe** \mathcal{U}
 - with **codes** written as $\hat{\Pi}, \hat{\Sigma}, \hat{0}, \hat{1}, \dots$
 - but thinking of them as $\forall, \exists, \perp, \top, \dots$
 - fibred **algebraic effects**
 - dep. typed **operation symbols** $\text{op} : (x_v : I) \longrightarrow O(x_v)$
 - ops. determine **comp. terms** $\text{op}_V^C(y : O[V/x_v]. M)$
 - effect eqs. determine **definitional eqs.**
 - a **derivable** “into-comps.” variant of **handlers** and **handling**
 M handled with $\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}; \overrightarrow{W_{\text{eq}}}\}_{\text{op} \in S_{\text{eff}}}$ to $y : A$ in $\underline{C} N_{\text{ret}}$
 - a **derivable** “into-values” variant of **handlers** and **handling**
 M handled with $\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}; \overrightarrow{W_{\text{eq}}}\}_{\text{op} \in S_{\text{eff}}}$ to $y : A$ in $\underline{B} V_{\text{ret}}$

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!
- They enable us to reason about effectful 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 (**think** M) computes a **proof obligation** for M
 - We discuss **three examples** of such predicates
- Also, can be an alternative to mon. reification for **rel. reasoning**
 - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \rightarrow A \times S$
 - Not investigated in this paper
 - See [Grimm et al.'18] for **reification-based** relational reasoning

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!
- They enable us to reason about effectful 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, can be an alternative to mon. reification for **rel. reasoning**
 - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \rightarrow A \times S$
 - Not investigated in this paper
 - See [Grimm et al.'18] for **reification-based** relational reasoning

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!
- They enable us to reason about effectful 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, $P(\text{thunk } M)$ computes a **proof obligation** for M
 - We discuss **three examples** of such predicates
- Also, can be an alternative to mon. reification for **rel. reasoning**
 - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \rightarrow A \times S$
 - Not investigated in this paper
 - See [Grimm et al.'18] for **reification-based** relational reasoning

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!
- They enable us to reason about effectful 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, $P(\text{thunk } M)$ computes a **proof obligation** for M
 - We discuss **three examples** of such predicates
- Also, can be an alternative to mon. reification for **rel. reasoning**
 - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \rightarrow A \times S$
 - Not investigated in this paper
 - See [Grimm et al.'18] for **reification-based** relational reasoning

Ex1: Lifting predicates to effectful comps.

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

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

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

using the **handler** given by

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

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

- $\Box P$ is similar to the **necessity modality** from Evaluation Logic

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

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

Ex1: Lifting predicates to effectful comps.

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

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

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

using the **handler** given by

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

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

- $\Box P$ is similar to the **necessity modality** from Evaluation Logic

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

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

Ex1: Lifting predicates to effectful comps.

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

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

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

using the **handler** given by

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

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

- $\Box P$ is similar to the **necessity modality** from Evaluation Logic

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

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

Ex1: Lifting predicates to effectful comps.

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

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

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

using the **handler** given by

$$\text{read}(x_k) \mapsto \hat{\Pi} y : \text{El}(\widehat{\text{Chr}}). x_k y \quad (\text{where } x_k : \text{Chr} \rightarrow \mathcal{U})$$

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

- $\Box P$ is similar to the **necessity modality** from Evaluation Logic

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

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

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

by

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

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

- 2) feeding in the **initial state**; and 3) projecting out the **value of \mathcal{U}**

- Then, wp_Q satisfies the **expected properties**, such as

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

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

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

by

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

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

2) feeding in the **initial state**; and 3) projecting out the **value of \mathcal{U}**

- Then, wp_Q satisfies the **expected properties**, such as

$$\Gamma \vdash \text{wp}_Q (\text{think} (\text{return } V)) \quad = \quad \lambda x_S : S . Q \ V \ x_S$$

$$\Gamma \vdash \text{wp}_Q (\text{think} (\text{put}_{(\ell, V)} (M))) \quad = \quad \lambda x_S : S . \text{wp}_Q (\text{think } M) \ x_S [\ell \mapsto V]$$

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

by

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

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

- 2) feeding in the **initial state**; and 3) projecting out the **value of \mathcal{U}**

- Then, wp_Q satisfies the **expected properties**, such as

$$\Gamma \vdash \text{wp}_Q (\text{think} (\text{return } V)) \quad = \quad \lambda x_S : S . Q \ V \ x_S$$

$$\Gamma \vdash \text{wp}_Q (\text{think} (\text{put}_{(\ell, V)} (M))) \quad = \quad \lambda x_S : S . \text{wp}_Q (\text{think } M) \ x_S [\ell \mapsto V]$$

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

by

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

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

- 2) feeding in the **initial state**; and 3) projecting out the **value of \mathcal{U}**

- Then, wp_Q satisfies the **expected properties**, such as

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

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

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

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

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

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

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

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

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

$$\text{Protocol} \rightarrow \mathcal{U}$$

given by (using pattern-matching lambda notation)

$$\begin{aligned} \text{read}(x_k) &\mapsto \lambda \{ (r \ x_{pr}) \rightarrow \widehat{\Pi} y : \text{El}(\widehat{\text{Chr}}) . x_k \ y \ (x_{pr} \ y) ; \\ &\quad - \rightarrow \widehat{0} \} \end{aligned}$$

$$\begin{aligned} \text{write}_{x_v}(x_k) &\mapsto \lambda \{ (w \ P \ x_{pr}) \rightarrow \widehat{\Sigma} y : \text{El}(P \ x_v) . x_k \ \star \ x_{pr} ; \\ &\quad - \rightarrow \widehat{0} \} \end{aligned}$$

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

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

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

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

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

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

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

$$\text{Protocol} \rightarrow \mathcal{U}$$

given by (using pattern-matching lambda notation)

$$\begin{aligned} \text{read}(x_k) &\mapsto \lambda \{ (r \ x_{pr}) \rightarrow \widehat{\Pi} y : \text{El}(\widehat{\text{Chr}}) . x_k \ y \ (x_{pr} \ y) ; \\ &\quad - \rightarrow \widehat{0} \} \end{aligned}$$

$$\begin{aligned} \text{write}_{x_v}(x_k) &\mapsto \lambda \{ (w \ P \ x_{pr}) \rightarrow \widehat{\Sigma} y : \text{El}(P \ x_v) . x_k \ \star \ x_{pr} ; \\ &\quad - \rightarrow \widehat{0} \} \end{aligned}$$

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

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

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

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

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

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

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

$$\text{Protocol} \rightarrow \mathcal{U}$$

given by (using pattern-matching lambda notation)

$$\begin{aligned} \text{read}(x_k) \quad \mapsto \quad & \lambda \{ (r \ x_{pr}) \rightarrow \widehat{\Pi} y : \text{El}(\widehat{\text{Chr}}) . x_k \ y \ (x_{pr} \ y) ; \\ & \quad \quad \quad \rightarrow \widehat{0} \} \end{aligned}$$

$$\begin{aligned} \text{write}_{x_v}(x_k) \quad \mapsto \quad & \lambda \{ (w \ P \ x_{pr}) \rightarrow \widehat{\Sigma} y : \text{El}(P \ x_v) . x_k \ \star \ x_{pr} ; \\ & \quad \quad \quad \rightarrow \widehat{0} \} \end{aligned}$$

Outline

- Setting the scene
 - **Algebraic effects** and their **handlers**
 - An effectful dependently typed **core 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 (unsoundness)
 - 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

$$M, N ::= \dots \mid \text{op}_V^{\underline{C}}(y : O[V/x_v]. M) \quad (\text{op} : (x_v : I) \longrightarrow O \in \mathcal{S})$$

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

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

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

Extending the FoSSaCS'16 calculus

- 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

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

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

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

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

Extending the FoSSaCS'16 calculus

- 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

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

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

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

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

Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS'16 **computation terms** with
$$M, N ::= \dots \mid M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$$
- But as handling denotes a **homomorphism**, then perhaps also
$$K, L ::= \dots \mid K \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$$
- However, this leads to an **inconsistent** system, e.g.,

$$\Gamma \vdash \text{write}_a(\text{return } \star) = \text{write}_z(\text{return } \star) : F1$$

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

Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS'16 **computation terms** with

$M, N ::= \dots \mid M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

$K, L ::= \dots \mid K \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

$$\Gamma \vdash \text{write}_a(\text{return } \star) = \text{write}_z(\text{return } \star) : F1$$

- At a very high-level, the problem is (see the paper for details)

- interaction between K s and ops. is governed by comp. types
- but the type of handled with does not mention the handler

Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS'16 **computation terms** with

$M, N ::= \dots \mid M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

$K, L ::= \dots \mid K \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

$$\Gamma \vdash \text{write}_a(\text{return } \star) = \text{write}_z(\text{return } \star) : F1$$

- At a very high-level, the problem is (see the paper for details)

- interaction between K s and ops. is governed by comp. types
- but the type of handled with does not mention the handler

Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS'16 **computation terms** with

$M, N ::= \dots \mid M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

$K, L ::= \dots \mid K \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$

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

$$\Gamma \vdash \text{write}_a(\text{return } \star) = \text{write}_z(\text{return } \star) : F1$$

- At a very high-level, the problem is (see the paper for details)

- interaction between K s and ops. is governed by comp. types
- but the type of handled with does not mention the handler

Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS'16 **computation terms** with
$$M, N ::= \dots \mid M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$$
- But as handling denotes a **homomorphism**, then perhaps also
$$K, L ::= \dots \mid K \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}}$$
- However, this leads to an **inconsistent** system, e.g.,

$$\Gamma \vdash \text{write}_a(\text{return } \star) = \text{write}_z(\text{return } \star) : F1$$

- At a very high-level, the problem is (see the paper for details)
 - interaction between K s 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**

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

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

$$\begin{aligned} & M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto \textcolor{red}{N_{\text{op}}}; \overrightarrow{W_{\text{eq}}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}} \\ & \quad \underline{\underline{\text{def}}} \\ & \text{force}_{\underline{C}}(\text{thunk}(\underbrace{M \text{ to } y:A \text{ in force}_{\langle \underline{U}\underline{C}; \textcolor{red}{\overrightarrow{V}_{N_{\text{op}}}}; \overrightarrow{W_{\text{eq}}}} \rangle}_{\text{temporarily working at type } \langle \underline{U}\underline{C}; \textcolor{red}{\overrightarrow{V}_{N_{\text{op}}}}; \overrightarrow{W_{\text{eq}}}} \rangle}(\text{thunk } N_{\text{ret}}))) \end{aligned}$$

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**

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

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

$$\begin{aligned} M \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}; \overrightarrow{W_{\text{eq}}}\}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} N_{\text{ret}} \\ \stackrel{\text{def}}{=} \\ \text{force}_{\underline{C}}(\underbrace{\text{thunk}(M \text{ to } y:A \text{ in force}_{\langle \underline{C}; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle}(\text{thunk } N_{\text{ret}}))) \\ \text{temporarily working at type } \langle \underline{C}; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle \end{aligned}$$

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**

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

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

$$\begin{array}{c}
 \textcolor{blue}{M} \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto \textcolor{red}{N}_{\text{op}}; \overrightarrow{W_{\text{eq}}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} \textcolor{blue}{N}_{\text{ret}} \\
 \underline{\text{def}} \\
 \text{force}_{\underline{C}}(\text{thunk}(\underbrace{\textcolor{blue}{M} \text{ to } y:A \text{ in force}_{\langle \underline{U}_{\underline{C}}; \textcolor{red}{V}_{\text{Nop}}; \overrightarrow{W_{\text{eq}}} \rangle}(\text{thunk } \textcolor{blue}{N}_{\text{ret}}))}_{\text{temporarily working at type } \langle \underline{U}_{\underline{C}}; \textcolor{red}{V}_{\text{Nop}}; \overrightarrow{W_{\text{eq}}} \rangle}))
 \end{array}$$

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**

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

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

$$\underline{M} \text{ handled with } \{\text{op}_{x_v}(x_k) \mapsto \textcolor{red}{N_{\text{op}}}; \overrightarrow{W_{\text{eq}}}\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y:A \text{ in}_{\underline{C}} \underline{N_{\text{ret}}}$$

$\underline{\text{def}}$

$$\text{force}_{\underline{C}}(\text{thunk}(\underbrace{\underline{M} \text{ to } y:A \text{ in force}_{\langle \underline{U}_{\underline{C}}; \textcolor{red}{V}_{N_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle}(\text{thunk } \underline{N_{\text{ret}}}))}_{\text{temporarily working at type } \langle \underline{U}_{\underline{C}}; \textcolor{red}{V}_{N_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle}))$$

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**
 - discussion about the calculus's inherent **extensional nature**
 - **Agda code** for the example predicates $P : UFA \rightarrow \mathcal{U}$

Thank you!

Questions?