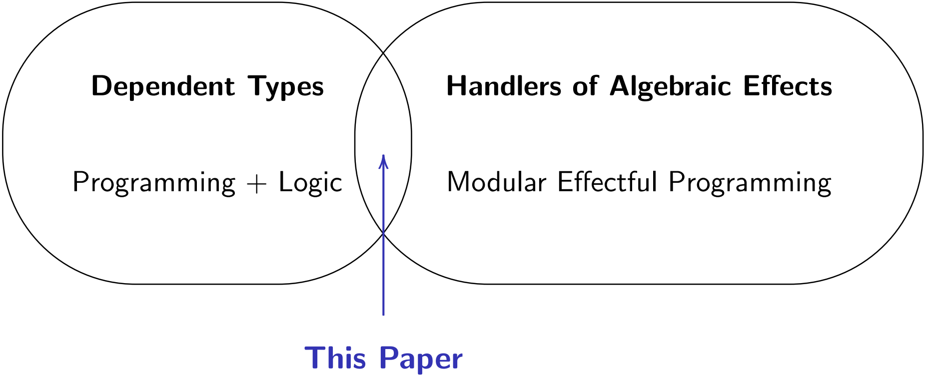
Handling Fibred Algebraic Effects

Danel Ahman

Prosecco Team, INRIA Paris

POPL 2018

January 10, 2018


Dependent Types
Programming + Logic
Handlers of Algebraic Effects
Modular Effectful Programming
This Paper

Outline

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

    [A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations

- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A type-level treatment of handlers

Outline

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

    [A., Ghani, Plotkin'16]
- What can we gain from handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations
- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A type-level treatment of handlers

Algebraic effects and their handlers

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

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

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

$$\mathsf{raise} : \mathsf{Exc} \longrightarrow 0 \qquad \mathsf{read} : \mathsf{Loc} \longrightarrow \mathsf{Val} \qquad \mathsf{write} : \mathsf{Loc} \times \mathsf{Val} \longrightarrow 1$$

  - **equations** – describing the computational behaviour

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

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

Algebraic effects and their handlers

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

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

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

$$\mathsf{raise} : \mathsf{Exc} \longrightarrow 0 \qquad \mathsf{read} : \mathsf{Loc} \longrightarrow \mathsf{Val} \qquad \mathsf{write} : \mathsf{Loc} \times \mathsf{Val} \longrightarrow 1$$

  - **equations** – describing the computational behaviour

$$\ell : \mathsf{Loc} \mid w : 1 \vdash \mathtt{read}_\ell\big(x.\mathtt{write}_{\langle \ell, x\rangle}(w(\star))\big) = w(\star)$$

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

Algebraic effects and their handlers

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

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

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

$$\mathsf{raise} : \mathsf{Exc} \longrightarrow 0 \qquad \mathsf{read} : \mathsf{Loc} \longrightarrow \mathsf{Val} \qquad \mathsf{write} : \mathsf{Loc} \times \mathsf{Val} \longrightarrow 1$$

  - **equations** – describing the computational behaviour

$$\ell : \mathsf{Loc} \mid w : 1 \vdash \mathtt{read}_\ell\big(x.\mathtt{write}_{\langle \ell, x\rangle}(w(\star))\big) = w(\star)$$

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

Algebraic effects and their handlers ctd.

- Plotkin and Pretnar's **handlers** of algebraic effects
  - generalise 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

$$M \ \texttt{handled with} \ \{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}} \ \texttt{to} \ y{:}A \ \texttt{in}_{\underline{C}} \ N_{\mathrm{ret}}$$

denoting a **homomorphism** $FA \longrightarrow \{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}}$

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

$$=$$

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

and

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

Algebraic effects and their handlers ctd.

- Plotkin and Pretnar's **handlers** of algebraic effects
  - generalise 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

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

denoting a **homomorphism** $FA \longrightarrow \{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}}$

$$(\mathrm{op}_V(y.M)) \;\texttt{handled with}\; \{\ldots\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\mathrm{ret}}$$
$$=$$
$$N_{\mathrm{op}}[V/x_v][\lambda\, y\!:\!O\,.\,\texttt{thunk}\,(M \;\texttt{handled with}\; \ldots)/x_k]$$

and

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

Algebraic effects and their handlers ctd.

- Plotkin and Pretnar's **handlers** of algebraic effects
  - generalise 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

$$M \texttt{ handled with } \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \texttt{ to } y\!:\!A \texttt{ in}_{\underline{C}}\ N_{\mathrm{ret}}$$

denoting a **homomorphism** $FA \longrightarrow \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}}$

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

$$=$$

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

and

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

Algebraic effects and their handlers ctd.

- Plotkin and Pretnar's **handlers** of algebraic effects
  - generalise 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

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

denoting a **homomorphism** $FA \longrightarrow \{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}}$

$$(\mathrm{op}_V(y.M)) \;\texttt{handled with}\; \{\ldots\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\mathrm{ret}}$$
$$=$$
$$N_{\mathrm{op}}[V/x_v][\lambda\, y\!:\!O \,.\, \texttt{thunk}\,(M \;\texttt{handled with}\; \ldots)/x_k]$$

and

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

Outline

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

    [A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations

- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A 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** ($\Gamma \vdash \underline{C}$)

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

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

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

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

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

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

$$K, L \ ::= \ z \ \mid \ K \ \mathtt{to}\ x : A \ \mathtt{in}_{\underline{C}}\ M \ \mid \ \ldots \qquad \text{(stacks, eval. ctxs.)}$$

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** ($\Gamma \vdash \underline{C}$)

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

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

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

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

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

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

$$K, L \ ::= \ z \ \mid \ K \ \mathtt{to}\ x{:}A\ \mathtt{in}_{\underline{C}}\ M \ \mid \ \ldots \qquad \text{(stacks, eval. ctxs.)}$$

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** ($\Gamma \vdash \underline{C}$)

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

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

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

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

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

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

$$K, L \ ::= \ z \ \mid \ K \ \mathtt{to}\ x{:}A\ \mathtt{in}_{\underline{C}}\ M \ \mid \ \ldots \qquad \text{(stacks, eval. ctxs.)}$$

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** $(\Gamma \vdash \underline{C})$

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

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

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

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

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

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

$$K, L \ ::= \ z \ \mid \ K \ \mathtt{to}\ x{:}A \ \mathtt{in}_{\underline{C}}\ M \ \mid \ \ldots \qquad \text{(stacks, eval. ctxs.)}$$

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** $(\Gamma \vdash \underline{C})$

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

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

$$V, W \;::=\; \ldots \;\mid\; \mathtt{thunk}\, M$$

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

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

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

$$K, L \;::=\; z \;\mid\; K \;\mathtt{to}\; x\!:\!A \;\mathtt{in}_{\underline{C}}\; M \;\mid\; \ldots \qquad \text{(stacks, eval. ctxs.)}$$

Outline

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

    [A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations

- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A type-level treatment of handlers

Defining predicates on effectful comps.

- In our extension of the FoSSaCS'16 calculus,
  - we have a Tarski-style **value universe** $\mathcal{U}$ with codes $\widehat{\Pi}, \widehat{\Sigma}, \ldots$
  - we have **fibred algebraic effects** $\mathsf{op} : (x_v : I) \longrightarrow O(x_v)$
  - we can **handle computations into values**

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

- Then we can define **predicates** $P : UFA \to \mathcal{U}$ (a value term) by
  1) equipping $\mathcal{U}$ (or a corresponding type) with an **algebra** structure
  2) **handling** the given computation using that algebra

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

- The paper discusses **three examples** of such predicates

Defining predicates on effectful comps.

- In our extension of the FoSSaCS'16 calculus,
  - we have a Tarski-style **value universe** $\mathcal{U}$ with codes $\widehat{\Pi}, \widehat{\Sigma}, \ldots$
  - we have **fibred algebraic effects** $\mathsf{op} : (x_v : I) \longrightarrow O(x_v)$
  - we can **handle computations into values**

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

- Then we can define **predicates** $P : UFA \to \mathcal{U}$ (a value term) by
  1) equipping $\mathcal{U}$ (or a corresponding type) with an **algebra** structure
  2) **handling** the given computation using that algebra

- Intuitively, $P\ (\mathtt{thunk}\ M)$ computes a **proof obligation** for M

- The paper discusses **three examples** of such predicates

Ex1: Lifting predicates to effectful comps.

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

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

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

  using the **handler** given by

$$\begin{array}{lll} \mathsf{read}(x_k) & \mapsto & \widehat{\Pi}\, y\!:\!\mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, x_k\ y \qquad \text{(where } x_k\!:\!\mathsf{Chr} \to \mathcal{U}\text{)} \\ \mathsf{write}_{x_v}(x_k) & \mapsto & x_k\ \star \qquad \text{(where } x_v\!:\!\mathsf{Chr},\ x_k\!:\!1 \to \mathcal{U}\text{)} \end{array}$$

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

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

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

Ex1: Lifting predicates to effectful comps.

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

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

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

  using the **handler** given by

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

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

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

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

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

Ex1: Lifting predicates to effectful comps.

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

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

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

  using the **handler** given by

$$\begin{array}{lcll} \mathsf{read}(x_k) & \mapsto & \widehat{\Pi}\, y : \mathsf{El}(\widehat{\mathsf{Chr}})\,.\, x_k\ y & (\text{where } x_k : \mathsf{Chr} \to \mathcal{U}) \\ \mathsf{write}_{x_v}(x_k) & \mapsto & x_k\ \star & (\text{where } x_v : \mathsf{Chr},\ x_k : 1 \to \mathcal{U}) \end{array}$$

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

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

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

Ex1: Lifting predicates to effectful comps.

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

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

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

  using the **handler** given by

$$\begin{array}{lll} \mathsf{read}(x_k) & \mapsto & \widehat{\Pi}\, y : \mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, x_k\ y & \qquad (\text{where } x_k : \mathsf{Chr} \to \mathcal{U}) \\ \mathsf{write}_{x_v}(x_k) & \mapsto & x_k\ \star & \qquad (\text{where } x_v : \mathsf{Chr},\ x_k : 1 \to \mathcal{U}) \end{array}$$

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

$$\Gamma \vdash \Box P\, \big(\texttt{thunk}\, (\texttt{read}(x \,.\, \texttt{write}_{\prime e'}(\texttt{return}\, V)))\big) = \widehat{\Pi}\, x : \mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, P\ V$$

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

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

by

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

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

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

- Then, wp_Q satisfies the expected properties, such as

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

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

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

by

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

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

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

- Then, wp_Q satisfies the expected properties, such as

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

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

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

  by

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

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

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

- Then, wp_Q satisfies the expected properties, such as

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

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

Ex2: Dijkstra's weakest precondition sem.

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

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

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

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

  by

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

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

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

- Then, wp_Q satisfies the expected properties, such as

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

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

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

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

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

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

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

$$\text{Allowed} : \mathit{UFA} \to \text{Protocol} \to \mathcal{U}$$

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

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

given by (using pattern-matching lambda notation)

$$\begin{array}{lcll} \text{read}(x_k) & \mapsto & \lambda\,\{(\mathtt{r}\ x_{pr}) & \to \widehat{\Pi}\, y \,{:}\, \text{El}(\widehat{\text{Chr}})\,.\, x_k\ y\ (x_{pr}\ y)\ ; \\ & & \quad\ \_ & \to \widehat{0}\,\} \\ & & & \\ \text{write}_{x_v}(x_k) & \mapsto & \lambda\,\{(\mathtt{w}\ P\ x_{pr}) & \to \widehat{\Sigma}\, y \,{:}\, \text{El}(P\, x_v)\,.\, x_k\ \star\ x_{pr}\ ; \\ & & \quad\ \_ & \to \widehat{0}\,\} \end{array}$$

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

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

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

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

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

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

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

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

given by (using pattern-matching lambda notation)

$$\begin{array}{lcll} \mathsf{read}(x_k) & \mapsto & \lambda \{ (\mathtt{r}\ x_{pr}) & \to \widehat{\Pi}\, y : \mathsf{El}(\widehat{\mathsf{Chr}}) \,.\, x_k\ y\ (x_{pr}\ y)\ ; \\ & & \quad \_ & \to \widehat{0}\, \} \\ \mathsf{write}_{x_v}(x_k) & \mapsto & \lambda \{ (\mathtt{w}\ P\ x_{pr}) & \to \widehat{\Sigma}\, y : \mathsf{El}(P\, x_v) \,.\, x_k\ \star\ x_{pr}\ ; \\ & & \quad \_ & \to \widehat{0}\, \} \end{array}$$

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

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

$$\mathtt{e} : \text{Protocol} \qquad \mathtt{r} : (\text{Chr} \to \text{Protocol}) \to \text{Protocol}$$
$$\mathtt{w} : (\text{Chr} \to \mathcal{U}) \times \text{Protocol} \to \text{Protocol}$$

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

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

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

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

given by (using pattern-matching lambda notation)

$$\begin{array}{lcll}
\text{read}(x_k) & \mapsto & \lambda \{ (\mathtt{r}\ x_{pr}) & \to \widehat{\Pi}\, y : \text{El}(\widehat{\text{Chr}})\,.\, x_k\ y\ (x_{pr}\ y)\ ; \\
 & & \quad _ & \to \widehat{0}\, \} \\
\text{write}_{x_v}(x_k) & \mapsto & \lambda \{ (\mathtt{w}\ P\ x_{pr}) & \to \widehat{\Sigma}\, y : \text{El}(P\, x_v)\,.\, x_k\ \star\ x_{pr}\ ; \\
 & & \quad _ & \to \widehat{0}\, \}
\end{array}$$

Outline

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

    [A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations

- Extending the FoSSaCS'16 calculus with alg. effects and handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A type-level treatment of handlers

Extending the FoSSaCS'16 calculus

- Given a **fibred effect theory** $\mathcal{T} = (\mathcal{S}, \mathcal{E})$
- First, we extend the calculus with **algebraic effects** as follows:
  - we extend FoSSaCS'16 **computation terms** with

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

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

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

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

Extending the FoSSaCS'16 calculus

- Given a **fibred effect theory** $\mathcal{T} = (\mathcal{S}, \mathcal{E})$
- First, we extend the calculus with **algebraic effects** as follows:
  - we extend FoSSaCS'16 **computation terms** with

$$M, N \ ::= \ \ldots \ | \ \mathrm{op}^{\underline{C}}_{V}(y : O[V/x_v] \,.\, M) \qquad (\mathrm{op} : (x_v : I) \longrightarrow O \in \mathcal{S})$$

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

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

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

Extending the FoSSaCS'16 calculus

- Given a **fibred effect theory** $\mathcal{T} = (\mathcal{S}, \mathcal{E})$

- First, we extend the calculus with **algebraic effects** as follows:

  - we extend FoSSaCS'16 **computation terms** with

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

  - we extend the **equational theory** with equations given in $\mathcal{E}$

  - we capture the **interaction** of comp. terms and ops. with the eq.

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

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

Take 1: Term-level definition of handlers

- Begin with extending FoSSaCS'16 **comp. terms** with

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

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

$$K, L \;::=\; \ldots \;\mid\; K \;\texttt{handled with}\; \{\texttt{op}_{x_v}(x_k) \mapsto N_{\texttt{op}}\}_{\texttt{op} \in \mathcal{S}_{\text{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\text{ret}}$$

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

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

- 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

Take 1: Term-level definition of handlers

- Begin with extending FoSSaCS'16 **comp. terms** with

$$M, N \ ::= \ \ldots \ \mid \ M \ \texttt{handled with} \ \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \ \texttt{to} \ y\!:\!A \ \texttt{in}_{\underline{C}} \ N_{\mathsf{ret}}$$

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

$$K, L \ ::= \ \ldots \ \mid \ K \ \texttt{handled with} \ \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \ \texttt{to} \ y\!:\!A \ \texttt{in}_{\underline{C}} \ N_{\mathsf{ret}}$$

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

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

- 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

Take 1: Term-level definition of handlers

- Begin with extending FoSSaCS'16 **comp. terms** with

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

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

$$K, L \ ::= \ \ldots \ \mid \ K \ \mathtt{handled\ with}\ \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\ \mathtt{to}\ y \colon A\ \mathtt{in}_{\underline{C}}\ N_{\mathsf{ret}}$$

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

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

- 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

Take 1: Term-level definition of handlers

- Begin with extending FoSSaCS'16 **comp. terms** with

$$M, N \ ::= \ \ldots \ \mid \ M \ \texttt{handled with} \ \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathtt{op}}\}_{\mathtt{op} \in \mathcal{S}_{\mathrm{eff}}} \ \texttt{to} \ y \!:\! A \ \texttt{in}_{\underline{C}} \ N_{\mathrm{ret}}$$

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

$$K, L \ ::= \ \ldots \ \mid \ K \ \texttt{handled with} \ \{\mathtt{op}_{x_v}(x_k) \mapsto N_{\mathtt{op}}\}_{\mathtt{op} \in \mathcal{S}_{\mathrm{eff}}} \ \texttt{to} \ y \!:\! A \ \texttt{in}_{\underline{C}} \ N_{\mathrm{ret}}$$

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

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

- 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 categorical 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 categorical 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 categorical semantics

Take 2: A type-level treatment of handlers

- We extend FoSSaCS'16 **comp. types** with
  - a **user-defined algebra type**

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

  where

    - A is the **carrier** value type
    - $\overrightarrow{V_{\mathrm{op}}}$ is a set of user-defined **operations**
    - $\overrightarrow{W_{\mathrm{eq}}}$ is a set of **witnesses** of equational proof obligations

- As a result, we can **derive the handing construct** as

$$M \;\mathtt{handled\ with}\; \{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \;\mathtt{to}\; y\!:\!A \;\mathtt{in}_{\underline{C}}\; N_{\mathrm{ret}}$$

$$\stackrel{\mathrm{def}}{=}$$

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

  and similarly for the **value variant** of it

Take 2: A type-level treatment of handlers

- We extend FoSSaCS'16 **comp. 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
    - $\overrightarrow{V_{\mathsf{op}}}$ is a set of user-defined **operations**
    - $\overrightarrow{W_{\mathsf{eq}}}$ is a set of **witnesses** of equational proof obligations

- As a result, we can **derive the handing construct** as

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

$$\stackrel{\mathsf{def}}{=}$$

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

  and similarly for the **value variant** of it

Conclusion

- In this talk, I have shown you that
  - handlers are natural for defining preds./types on computations
  - unsoundness issues can easily arise when accommodating handlers
  - handlers admit a principled type-based treatment
- Look in the paper for
  - the formal details of what I have shown you today
  - a families fibrations based semantics of the extended calculus
  - a discussion about the inherent extensional nature of the calculus
  - an Agda formalisation of the example predicates $P : UFA \to \mathcal{U}$

Thank you!

Questions?