# Handling Fibred Computational Effects

## Effect Handlers in a Dependently Typed Setting

Danel Ahman

Prosecco Team at Inria Paris

HOPE 2017

September 3, 2017

# Outline

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

    [A., Ghani, Plotkin'16]
- Why handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations
- Extending the FoSSaCS'16 calculus with 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)
- Why handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations
- Extending the FoSSaCS'16 calculus with 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

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

- 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 operations (they denote **algebras**)
  - example uses – rollbacks, stream redirection, concurrency, ...

- Usually included in languages using the **handling** construct

$$M \ \mathtt{handled\ with}\ \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\ \mathtt{to}\ y\!:\!A\ \mathtt{in}_{\underline{C}}\ N_{\mathsf{ret}}$$

denoting the **homomorphism** $FA \longrightarrow \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}$

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

and

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

# Algebraic effects and their handlers ctd.

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

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

denoting the **homomorphism** $FA \longrightarrow \{\mathtt{op}_x(x') \mapsto N_{\mathtt{op}}\}_{\mathtt{op} \in \mathcal{S}_{\mathrm{eff}}}$

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

and

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

# Algebraic effects and their handlers ctd.

- Plotkin and Pretnar's **handlers** of algebraic effects
  - generalise exception handlers
  - given by redefining the given operations (they denote **algebras**)
  - example uses – rollbacks, stream redirection, concurrency, ...

- Usually included in languages using the **handling** construct

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

denoting the **homomorphism** $FA \longrightarrow \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}$

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

and

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

# Algebraic effects and their handlers ctd.

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

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

denoting the **homomorphism** $FA \longrightarrow \{\mathrm{op}_x(x') \mapsto N_{\mathrm{op}}\}_{\mathrm{op}\,\in\,\mathcal{S}_{\mathrm{eff}}}$

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

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)
- Why handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations
- Extending the FoSSaCS'16 calculus with 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** ($\Gamma \vdash \underline{C}$)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# Outline

- Setting the scene
  - Algebraic effects and their handlers
  - A core dependently typed effectful calculus (FoSSaCS'16)
- Why handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations
- Extending the FoSSaCS'16 calculus with handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A type-level treatment of handlers

# Defining predicates on effectful comps.

- For time being, assume that we have **handlers** in the calculus
  - In particular, assume that we can also **handle into values**

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

- Also assume that we have a Tarski-style **value universe** $\mathcal{U}$

- Then we can define **predicates** $P : UFA \to \mathcal{U}$ (a value term) by
  - equipping $\mathcal{U}$ with an **algebra** structure
  - **handling** the given computation using that algebra
  - intuitively, $P\ (\mathtt{thunk}\ M)$ computes a **proof obligation** for $M$

- Examples
  - **lifting predicates** from return values to (I/O)-computations
  - Dijkstra's **weakest precondition semantics** of state
  - specifying **allowed patterns** of (I/O)-effects

# Defining predicates on effectful comps.

- For time being, assume that we have **handlers** in the calculus
  - In particular, assume that we can also **handle into values**

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

- Also assume that we have a Tarski-style **value universe** $\mathcal{U}$
- Then we can define **predicates** $P : UFA \to \mathcal{U}$ (a value term) by
  - equipping $\mathcal{U}$ with an **algebra** structure
  - **handling** the given computation using that algebra
  - intuitively, $P\ (\texttt{thunk}\ M)$ computes a **proof obligation** for $M$

- Examples
  - **lifting predicates** from return values to (I/O)-computations
  - Dijkstra's **weakest precondition semantics** of state
  - specifying **allowed patterns** of (I/O)-effects

# Defining predicates on effectful comps.

- For time being, assume that we have **handlers** in the calculus
  - In particular, assume that we can also **handle into values**

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

- Also assume that we have a Tarski-style **value universe** $\mathcal{U}$
- Then we can define **predicates** $P : UFA \to \mathcal{U}$ (a value term) by
  - equipping $\mathcal{U}$ with an **algebra** structure
  - **handling** the given computation using that algebra
  - intuitively, $P\ (\texttt{thunk}\, M)$ computes a **proof obligation** for $M$
- Examples
  - **lifting predicates** from return values to (I/O)-computations
  - Dijkstra's **weakest precondition semantics** of state
  - specifying **allowed patterns** of (I/O)-effects

# Lifting predicates to effectful comps.

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

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

  $$\lambda\, y : UFA \,.\, (\mathtt{force}\, y)\ \mathtt{handled\ with}\ \{\ldots\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{IO}}}\ \mathtt{to}\ y' : A\ \mathtt{in}_{\mathcal{U}}\ P\, y'$$

  using the handler given by

  $$V_{\mathsf{read}} \stackrel{\mathsf{def}}{=} \lambda\, y : (\Sigma\, x : 1 \,.\, \mathsf{Chr} \to \mathcal{U}) \,.\, \mathtt{v\text{-}pi\text{-}code}(\mathtt{chr\text{-}code}\,,\, y' \,.\, (\mathtt{snd}\, y)\, y')$$

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

- $\widehat{P}$ is similar to the **necessity modality** from Evaluation Logic

  $$\Gamma \vdash \mathsf{El}(\widehat{P}\ (\mathtt{thunk}\,(\mathtt{read}^{FA}(x \,.\, \mathtt{return}\, W)))) = \Pi\, x : \mathsf{Chr} \,.\, P\, W$$

- To get **possibility mod.**, replace $\mathtt{v\text{-}pi\text{-}code}$ with $\mathtt{v\text{-}sigma\text{-}code}$

# Lifting predicates to effectful comps.

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

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

  $$\lambda\, y : UFA\, .\, (\texttt{force}\, y)\ \texttt{handled with}\ \{\ldots\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{IO}}}\ \texttt{to}\ y' : A\ \texttt{in}_{\mathcal{U}}\ P\, y'$$

  using the handler given by

  $$V_{\mathsf{read}} \stackrel{\mathsf{def}}{=} \lambda\, y : (\Sigma\, x : 1\, .\, \mathsf{Chr} \to \mathcal{U})\, .\, \texttt{v-pi-code}(\texttt{chr-code}\, , y'.\, (\texttt{snd}\, y)\, y')$$

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

- $\widehat{P}$ is similar to the **necessity modality** from Evaluation Logic

  $$\Gamma \vdash \mathsf{El}\big(\widehat{P}\ (\texttt{thunk}\,(\texttt{read}^{FA}(x\, .\, \texttt{return}\, W)))\big) = \Pi\, x : \mathsf{Chr}\, .\, P\, W$$

- To get **possibility mod.**, replace `v-pi-code` with `v-sigma-code`

# Lifting predicates to effectful comps.

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

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

  $$\lambda\, y : UFA\,.\,(\mathtt{force}\; y)\;\mathtt{handled\;with}\;\{\ldots\}_{\mathsf{op}\,\in\,\mathcal{S}_{\mathsf{IO}}}\;\mathtt{to}\;y' : A\;\mathtt{in}_{\mathcal{U}}\;P\,y'$$

  using the handler given by

  $$V_{\mathsf{read}} \stackrel{\mathrm{def}}{=} \lambda\, y : (\Sigma\, x : 1\,.\,\mathrm{Chr} \to \mathcal{U})\,.\,\mathtt{v\text{-}pi\text{-}code}\big(\mathtt{chr\text{-}code}\,,\,y'.\,(\mathtt{snd}\,y)\,y'\big)$$

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

- $\widehat{P}$ is similar to the **necessity modality** from Evaluation Logic

  $$\Gamma \vdash \mathrm{El}\big(\widehat{P}\,(\mathtt{thunk}\,(\mathtt{read}^{FA}(x\,.\,\mathtt{return}\,W)))\big) = \Pi\, x : \mathrm{Chr}\,.\,P\,W$$

- To get **possibility mod.**, replace v-pi-code with v-sigma-code

# Lifting predicates to effectful comps.

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

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

  $$\lambda\, y : UFA \,.\, (\mathtt{force}\, y)\ \mathtt{handled\ with}\ \{\ldots\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{IO}}}\ \mathtt{to}\ y' : A\ \mathtt{in}_{\mathcal{U}}\ P\, y'$$

  using the handler given by

  $$V_{\mathsf{read}} \stackrel{\mathsf{def}}{=} \lambda\, y : (\Sigma\, x : 1 \,.\, \mathsf{Chr} \to \mathcal{U}) \,.\, \mathtt{v\text{-}pi\text{-}code}\big(\mathtt{chr\text{-}code}\,,\, y' . (\mathtt{snd}\, y)\, y'\big)$$

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

- $\widehat{P}$ is similar to the **necessity modality** from Evaluation Logic

  $$\Gamma \vdash \mathsf{El}\big(\widehat{P}\ (\mathtt{thunk}\,(\mathtt{read}^{FA}(x \,.\, \mathtt{return}\, W)))\big) = \Pi\, x : \mathsf{Chr} \,.\, P\, W$$

- To get **possibility mod.**, replace v-pi-code with v-sigma-code

# Dijkstra's weakest precondition semantics

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

$$Q : A \to \mathrm{St} \to \mathcal{U}$$

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

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

  by

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

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

  **ii)** feeding in the initial state, and **iii)** projecting out the proposition

- Then $\mathrm{wp}_Q$ satisfies the expected properties, e.g.,

$$\Gamma \vdash \mathrm{wp}_Q\ (\mathtt{thunk}\,(\mathtt{return}\,V)) \;=\; \lambda\, x_S\!:\!\mathrm{St}\,.\,Q\ V\ x_S \qquad : \mathrm{St} \to \mathcal{U}$$

$$\Gamma \vdash \mathrm{wp}_Q\ (\mathtt{thunk}\,(\mathtt{put}^{FA}_{V_S}(M))) \;=\; \lambda\, x_S\!:\!\mathrm{St}\,.\,\mathrm{wp}_Q\ (\mathtt{thunk}\,M)\ V_S \quad : \mathrm{St} \to \mathcal{U}$$

# Dijkstra's weakest precondition semantics

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

$$Q : A \to \mathsf{St} \to \mathcal{U}$$

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

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

by

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

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

**ii)** feeding in the initial state, and **iii)** projecting out the proposition

- Then $\mathsf{wp}_Q$ satisfies the expected properties, e.g.,

$$\Gamma \vdash \mathsf{wp}_Q\ (\mathtt{thunk}\,(\mathtt{return}\ V)) \ = \ \lambda x_S{:}\mathsf{St}\,.\,Q\ V\ x_S \qquad : \mathsf{St} \to \mathcal{U}$$

$$\Gamma \vdash \mathsf{wp}_Q\ (\mathtt{thunk}\,(\mathtt{put}^{FA}_{V_S}(M))) \ = \ \lambda x_S{:}\mathsf{St}\,.\,\mathsf{wp}_Q\ (\mathtt{thunk}\,M)\ V_S \quad : \mathsf{St} \to \mathcal{U}$$

# Dijkstra's weakest precondition semantics

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

$$Q : A \to \mathsf{St} \to \mathcal{U}$$

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

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

by

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

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

**ii)** feeding in the initial state, and **iii)** projecting out the proposition

- Then $\mathsf{wp}_Q$ satisfies the expected properties, e.g.,

$$\Gamma \vdash \mathsf{wp}_Q\ (\mathtt{thunk}\,(\mathtt{return}\,V)) \ = \ \lambda x_S{:}\mathsf{St}\,.\,Q\ V\ x_S \qquad : \mathsf{St} \to \mathcal{U}$$

$$\Gamma \vdash \mathsf{wp}_Q\ (\mathtt{thunk}\,(\mathtt{put}^{FA}_{V_S}(M))) \ = \ \lambda x_S{:}\mathsf{St}\,.\,\mathsf{wp}_Q\ (\mathtt{thunk}\,M)\ V_S \quad : \mathsf{St} \to \mathcal{U}$$

# Dijkstra's weakest precondition semantics

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

  $$Q : A \to \mathsf{St} \to \mathcal{U}$$

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

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

  by

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

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

  **ii)** feeding in the initial state, and **iii)** projecting out the proposition

- Then $\mathsf{wp}_Q$ satisfies the expected properties, e.g.,

$$\Gamma \vdash \mathsf{wp}_Q\ (\mathtt{thunk}\,(\mathtt{return}\,V)) \;=\; \lambda\, x_S : \mathsf{St}\,.\, Q\ V\ x_S \qquad : \mathsf{St} \to \mathcal{U}$$

$$\Gamma \vdash \mathsf{wp}_Q\ (\mathtt{thunk}\,(\mathtt{put}^{FA}_{V_S}(M))) \;=\; \lambda\, x_S : \mathsf{St}\,.\, \mathsf{wp}_Q\ (\mathtt{thunk}\,M)\ V_S \qquad : \mathsf{St} \to \mathcal{U}$$

# Specifying allowed patterns of I/O-effects

- We assume an **inductive type** Protocol, given by

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

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

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

- Given a **protocol** $\mathrm{Pr} : \mathrm{Protocol}$, we define

$$\widehat{\mathrm{Pr}} : \mathit{UFA} \to \mathcal{U}$$

by handling a given comp. using

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

where

$$\begin{aligned}
V_{\mathrm{read}}\ \langle V, V_{\mathrm{rk}} \rangle\ (\mathtt{r}\ \mathrm{Pr}') &\stackrel{\mathrm{def}}{=} \mathtt{v\text{-}pi\text{-}code}(\mathtt{chr\text{-}code}\,, y\,.\,(V_{\mathrm{rk}}\, y)\,(\mathrm{Pr}'\, y)) \\
V_{\mathrm{write}}\ \langle V, V_{\mathrm{wk}} \rangle\ (\mathtt{w}\ \langle P, \mathrm{Pr}' \rangle) &\stackrel{\mathrm{def}}{=} \mathtt{v\text{-}sigma\text{-}code}(P\, V\,, y\,.\,V_{\mathrm{wk}} \star \mathrm{Pr}') \\
_ &\stackrel{\mathrm{def}}{=} \mathtt{empty\text{-}code}
\end{aligned}$$

# Specifying allowed patterns of I/O-effects

- We assume an **inductive type** Protocol, given by

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

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

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

- Given a **protocol** $\mathrm{Pr} : \mathrm{Protocol}$, we define

$$\widehat{\mathrm{Pr}} : \mathit{UFA} \to \mathcal{U}$$

  by handling a given comp. using

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

  where

$$\begin{aligned}
V_{\mathrm{read}}\ \langle V, V_{\mathrm{rk}} \rangle\ (\mathtt{r}\ \mathrm{Pr}') &\stackrel{\mathrm{def}}{=} \mathtt{v\text{-}pi\text{-}code}(\mathtt{chr\text{-}code}\,, y\,.\,(V_{\mathrm{rk}}\ y)\,(\mathrm{Pr}'\ y)) \\
V_{\mathrm{write}}\ \langle V, V_{\mathrm{wk}} \rangle\ (\mathtt{w}\ \langle P, \mathrm{Pr}' \rangle) &\stackrel{\mathrm{def}}{=} \mathtt{v\text{-}sigma\text{-}code}(P\ V\,, y\,.\,V_{\mathrm{wk}} \star \mathrm{Pr}') \\
_ &\stackrel{\mathrm{def}}{=} \mathtt{empty\text{-}code}
\end{aligned}$$

# Specifying allowed patterns of I/O-effects

- We assume an **inductive type** Protocol, given by

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

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

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

- Given a **protocol** $\mathsf{Pr} : \mathsf{Protocol}$, we define

$$\widehat{\mathsf{Pr}} : \mathit{UFA} \to \mathcal{U}$$

  by handling a given comp. using

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

  where

$$V_{\mathsf{read}}\ \langle V, V_{\mathsf{rk}}\rangle\ (\mathtt{r}\ \mathsf{Pr}') \stackrel{\mathrm{def}}{=} \texttt{v-pi-code}(\texttt{chr-code}, y.(V_{\mathsf{rk}}\ y)(\mathsf{Pr}'\ y))$$

$$V_{\mathsf{write}}\ \langle V, V_{\mathsf{wk}}\rangle\ (\mathtt{w}\ \langle P, \mathsf{Pr}'\rangle) \stackrel{\mathrm{def}}{=} \texttt{v-sigma-code}(P\ V, y. V_{\mathsf{wk}} \star \mathsf{Pr}')$$

$$\_ \stackrel{\mathrm{def}}{=} \texttt{empty-code}$$

# Specifying allowed patterns of I/O-effects

- We assume an **inductive type** Protocol, given by

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

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

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

- Given a **protocol** $\mathsf{Pr} : \mathsf{Protocol}$, we define

$$\widehat{\mathsf{Pr}} : \mathit{UFA} \to \mathcal{U}$$

  by handling a given comp. using

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

  where

$$V_{\mathsf{read}}\ \langle V, V_{\mathsf{rk}} \rangle\ (\mathtt{r}\ \mathsf{Pr}') \stackrel{\text{def}}{=} \mathtt{v\text{-}pi\text{-}code}(\mathtt{chr\text{-}code}, y \,.\, (V_{\mathsf{rk}}\ y)(\mathsf{Pr}'\ y))$$

$$V_{\mathsf{write}}\ \langle V, V_{\mathsf{wk}} \rangle\ (\mathtt{w}\ \langle P, \mathsf{Pr}' \rangle) \stackrel{\text{def}}{=} \mathtt{v\text{-}sigma\text{-}code}(P\ V, y \,.\, V_{\mathsf{wk}} \star \mathsf{Pr}')$$

$$\_ \stackrel{\text{def}}{=} \mathtt{empty\text{-}code}$$

# Specifying allowed patterns of I/O-effects

- We assume an **inductive type** Protocol, given by

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

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

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

- Given a **protocol** $\mathsf{Pr} : \mathsf{Protocol}$, we define

$$\widehat{\mathsf{Pr}} : \mathit{UFA} \to \mathcal{U}$$

  by handling a given comp. using

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

  where

$$\begin{array}{lcl}
V_{\mathsf{read}}\ \langle V, V_{\mathsf{rk}} \rangle\ (\texttt{r}\ \mathsf{Pr}') & \stackrel{\text{def}}{=} & \texttt{v-pi-code}\big(\texttt{chr-code}\,, y\,.\,(V_{\mathsf{rk}}\ y)\,(\mathsf{Pr}'\ y)\big) \\
V_{\mathsf{write}}\ \langle V, V_{\mathsf{wk}} \rangle\ (\texttt{w}\ \langle P, \mathsf{Pr}' \rangle) & \stackrel{\text{def}}{=} & \texttt{v-sigma-code}\big(P\ V\,, y\,.\,V_{\mathsf{wk}} \star \mathsf{Pr}'\big) \\
_ & \stackrel{\text{def}}{=} & \texttt{empty-code}
\end{array}$$

# Outline

- Setting the scene
  - Algebraic effects and their handlers
  - A core dependently typed effectful calculus (FoSSaCS'16)
- Why handlers + dependent types?
  - Programming with handlers + expressiveness of dep. types
  - Useful for defining predicates/types depending on computations
- Extending the FoSSaCS'16 calculus with handlers
  - Take 1: The common term-level def. of handlers (unsound)
  - Take 2: A type-level treatment of handlers

# Fibred algebraic effects

- To include fib. alg. effects $(\mathcal{S}_{\text{eff}}, \mathcal{E}_{\text{eff}})$ in our calculus, we
  - extend its computation terms with **algebraic operations**

$$\frac{\Gamma \vdash V : I \qquad \Gamma \vdash \underline{C} \qquad \Gamma, y : O[V/x] \vdash M : \underline{C}}{\Gamma \vdash \mathrm{op}^{\underline{C}}_{V}(y.M) : \underline{C}}$$

    for every dep. typed op. symbol $\mathrm{op} : (x : I) \longrightarrow O$ in $\mathcal{S}_{\text{eff}}$

  - include **equations** $\Gamma \mid \Delta \vdash T_1 = T_2$ given in $\mathcal{E}_{\text{eff}}$

  - include a general **algebraicity equation**

$$\frac{\Gamma \mid z : \underline{C} \vdash K : \underline{D} \qquad \Gamma \vdash V : I \qquad \Gamma, y : O[V/x] \vdash M : \underline{C}}{\Gamma \vdash K[\mathrm{op}^{\underline{C}}_{V}(y.M)/z] = \mathrm{op}^{\underline{D}}_{V}(y.K[M/z]) : \underline{D}}$$

# Fibred algebraic effects

- To include fib. alg. effects $(\mathcal{S}_{\mathsf{eff}}, \mathcal{E}_{\mathsf{eff}})$ in our calculus, we
  - extend its computation terms with **algebraic operations**

$$\frac{\Gamma \vdash V : I \qquad \Gamma \vdash \underline{C} \qquad \Gamma, y \!:\! O[V/x] \vdash M : \underline{C}}{\Gamma \vdash \mathrm{op}^{\underline{C}}_{V}(y.M) : \underline{C}}$$

for every dep. typed op. symbol $\mathsf{op} : (x \!:\! I) \longrightarrow O$ in $\mathcal{S}_{\mathsf{eff}}$

  - include **equations** $\Gamma \mid \Delta \vdash T_1 = T_2$ given in $\mathcal{E}_{\mathsf{eff}}$
  - include a general **algebraicity equation**

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

# Fibred algebraic effects

- To include fib. alg. effects $(\mathcal{S}_{\mathsf{eff}}, \mathcal{E}_{\mathsf{eff}})$ in our calculus, we
  - extend its computation terms with **algebraic operations**

$$\frac{\Gamma \vdash V : I \qquad \Gamma \vdash \underline{C} \qquad \Gamma, y \,{:}\, O[V/x] \vdash M : \underline{C}}{\Gamma \vdash \mathtt{op}^{\underline{C}}_{V}(y.M) : \underline{C}}$$

  for every dep. typed op. symbol $\mathsf{op} : (x \,{:}\, I) \longrightarrow O$ in $\mathcal{S}_{\mathsf{eff}}$

  - include **equations** $\Gamma \mid \Delta \vdash T_1 = T_2$ given in $\mathcal{E}_{\mathsf{eff}}$

  - include a general **algebraicity equation**

$$\frac{\Gamma \mid z \,{:}\, \underline{C} \vdash K : \underline{D} \qquad \Gamma \vdash V : I \qquad \Gamma, y \,{:}\, O[V/x] \vdash M : \underline{C}}{\Gamma \vdash K[\mathtt{op}^{\underline{C}}_{V}(y.M)/z] = \mathtt{op}^{\underline{D}}_{V}(y.K[M/z]) : \underline{D}}$$

# Handlers for fibred algebraic effects

- **Take 1:** Let's use their conventional term-level definition

  - include the handling construct for **computation terms**

    $$M \;\texttt{handled with}\; \{\texttt{op}_x(x') \mapsto N_{\texttt{op}}\}_{\texttt{op} \in \mathcal{S}_{\text{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\text{ret}}$$

  - as handling denotes a homomorphism, also for **hom. terms**

    $$K \;\texttt{handled with}\; \{\texttt{op}_x(x') \mapsto N_{\texttt{op}}\}_{\texttt{op} \in \mathcal{S}_{\text{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\text{ret}}$$

  - but then we can prove the **unsound equation**

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

    by **handling**

    $$\texttt{write}^{F1}_{\mathbf{a}}(\texttt{return}\,\star)$$

    with

    $$\texttt{write}_x(x') \mapsto \texttt{write}_{\mathbf{z}}(\texttt{force}\,(x'\,\star))$$

    and using $\beta$-**eqs. for handling** and the **general algebraicity eq.**

# Handlers for fibred algebraic effects

- **Take 1:** Let's use their conventional term-level definition
  - include the handling construct for **computation terms**

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

  - as handling denotes a homomorphism, also for **hom. terms**

    $$K \;\texttt{handled with}\; \{\mathtt{op}_x(x') \mapsto N_{\mathtt{op}}\}_{\mathtt{op} \in \mathcal{S}_{\mathrm{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\mathrm{ret}}$$

  - but then we can prove the **unsound equation**

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

    by **handling**

    $$\mathtt{write}^{F1}_{\mathsf{a}}(\mathtt{return}\,\star)$$

    with

    $$\mathtt{write}_x(x') \mapsto \mathtt{write}_{\mathsf{z}}(\mathtt{force}\,(x'\,\star))$$

    and using $\beta$-**eqs. for handling** and the **general algebraicity eq.**

# Handlers for fibred algebraic effects

- **Take 1:** Let's use their conventional term-level definition
  - include the handling construct for **computation terms**

    $$M \ \mathtt{handled\ with}\ \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\ \mathtt{to}\ y\!:\!A\ \mathtt{in}_{\underline{C}}\ N_{\mathsf{ret}}$$

  - as handling denotes a homomorphism, also for **hom. terms**

    $$K \ \mathtt{handled\ with}\ \{\mathtt{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}\ \mathtt{to}\ y\!:\!A\ \mathtt{in}_{\underline{C}}\ N_{\mathsf{ret}}$$

  - but then we can prove the **unsound equation**

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

    by **handling**

    $$\mathtt{write}^{F1}_{\mathsf{a}}(\mathtt{return}\,\star)$$

    with

    $$\mathtt{write}_x(x') \mapsto \mathtt{write}_{\mathsf{z}}(\mathtt{force}\,(x'\,\star))$$

    and using $\beta$-**eqs. for handling** and the **general algebraicity eq.**

# Handlers for fibred algebraic effects

- **Take 1:** Let's use their conventional term-level definition
  - include the handling construct for **computation terms**

    $$M \;\texttt{handled with}\; \{\texttt{op}_x(x') \mapsto N_{\texttt{op}}\}_{\texttt{op} \in \mathcal{S}_{\text{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\text{ret}}$$

  - as handling denotes a homomorphism, also for **hom. terms**

    $$K \;\texttt{handled with}\; \{\texttt{op}_x(x') \mapsto N_{\texttt{op}}\}_{\texttt{op} \in \mathcal{S}_{\text{eff}}} \;\texttt{to}\; y\!:\!A \;\texttt{in}_{\underline{C}}\; N_{\text{ret}}$$

  - but then we can prove the **unsound equation**

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

    by **handling**

    $$\texttt{write}^{F1}_{\mathbf{a}}(\texttt{return}\,\star)$$

    with

    $$\texttt{write}_x(x') \mapsto \texttt{write}_{\mathbf{z}}(\texttt{force}\,(x'\,\star))$$

    and using $\beta$-**eqs. for handling** and the **general algebraicity eq.**

# Handlers for fibred algebraic effects

- **Take 1:** Let's use their conventional term-level definition
  - include the handling construct for **computation terms**

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

  - as handling denotes a homomorphism, also for **hom. terms**

    $$K \;\texttt{handled with}\; \{\mathtt{op}_x(x') \mapsto N_{\mathtt{op}}\}_{\mathtt{op} \in \mathcal{S}_{\mathsf{eff}}} \;\texttt{to}\; y \!:\! A \;\texttt{in}_{\underline{C}}\; N_{\mathsf{ret}}$$

  - but then we can prove the **unsound equation**

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

    by **handling**

    $$\mathtt{write}^{F1}_{\mathsf{a}}(\mathtt{return}\,\star)$$

    with

    $$\mathtt{write}_x(x') \mapsto \mathtt{write}_{\mathsf{z}}(\mathtt{force}\,(x'\,\star))$$

    and using $\beta$**-eqs. for handling** and the **general algebraicity eq.**

# Handlers for fibred algebraic effects ctd.

- Possible ways to solve this unsoundness problem
  - **Option 1:** Change the FoSSaCS'16 calculus
    - change the equational theory of homomorphism terms
    - hom. terms wouldn't denote homomorphisms any more
    - investigated for exceptions in CBPV with stacks in [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

# Handlers for fibred algebraic effects ctd.

- Possible ways to solve this unsoundness problem
  - **Option 1:** Change the FoSSaCS'16 calculus
    - change the equational theory of homomorphism terms
    - hom. terms wouldn't denote homomorphisms any more
    - investigated for exceptions in CBPV with stacks in [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

# Handlers for fibred algebraic effects ctd.

- Possible ways to solve this unsoundness problem
  - **Option 1:** Change the FoSSaCS'16 calculus
    - change the equational theory of homomorphism terms
    - hom. terms wouldn't denote homomorphisms any more
    - investigated for exceptions in CBPV with stacks in [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

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers

  - we introduce the **user-defined algebra type** (comp. type)

$$\frac{\Gamma \vdash A \qquad \{\Gamma \vdash V_{\mathsf{op}} : (\Sigma x \,{:}\, I . O \to A) \to A\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \qquad V_{\mathsf{op}} \text{ satisfy the equations in } \mathcal{E}_{\mathsf{eff}}}{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}$$

  - comps. of this type are **introduced** by $\mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}\, V$

  - we introduce corresponding **elimination form**

$$\frac{\Gamma \vdash M : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x \,{:}\, A \vdash N : \underline{C} \qquad N \text{ behaves as a homomorphism in } x \text{ (i.e., commutes with ops.)}}{\Gamma \vdash M \;\mathtt{as}\; x \,{:}\, U\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \;\mathtt{in}\; N : \underline{C}}$$

    and similarly for homomorphism terms

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - we introduce the **user-defined algebra type** (comp. type)

$$\frac{\Gamma \vdash A \qquad \{\Gamma \vdash V_{\mathsf{op}} : (\Sigma x \,{:}\, I . O \to A) \to A\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \qquad V_{\mathsf{op}} \text{ satisfy the equations in } \mathcal{E}_{\mathsf{eff}}}{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}$$

  - comps. of this type are **introduced** by $\mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}\ V$
  - we introduce corresponding **elimination form**

$$\frac{\Gamma \vdash M : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x \,{:}\, A \vdash N : \underline{C} \qquad N \text{ behaves as a homomorphism in x (i.e., commutes with ops.)}}{\Gamma \vdash M \ \mathtt{as}\ x \,{:}\, U\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \ \mathtt{in}\ N : \underline{C}}$$

  and similarly for homomorphism terms

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - we introduce the **user-defined algebra type** (comp. type)

$$\frac{\Gamma \vdash A \qquad \{\Gamma \vdash V_{\mathsf{op}} : (\Sigma x\!:\!I.O \to A) \to A\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \qquad V_{\mathsf{op}} \text{ satisfy the equations in } \mathcal{E}_{\mathsf{eff}}}{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}$$

  - comps. of this type are **introduced** by $\mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}\ V$
  - we introduce corresponding **elimination form**

$$\frac{\Gamma \vdash M : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x\!:\!A \vdash N : \underline{C} \qquad N \text{ behaves as a homomorphism in x (i.e., commutes with ops.)}}{\Gamma \vdash M \ \mathtt{as}\ x\!:\!U\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \ \mathtt{in}\ N : \underline{C}}$$

  and similarly for homomorphism terms

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - we introduce the **user-defined algebra type** (comp. type)

$$\frac{\Gamma \vdash A \qquad \{\Gamma \vdash V_{\mathsf{op}} : (\Sigma x \!:\! I . O \to A) \to A\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \qquad V_{\mathsf{op}} \text{ satisfy the equations in } \mathcal{E}_{\mathsf{eff}}}{\Gamma \vdash \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}$$

  - comps. of this type are **introduced** by $\mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}\ V$
  - we introduce corresponding **elimination form**

$$\frac{\Gamma \vdash M : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x \!:\! A \vdash N : \underline{C} \qquad N \text{ behaves as a homomorphism in } x \text{ (i.e., commutes with ops.)}}{\Gamma \vdash M \ \mathtt{as}\ x \!:\! U\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle \ \mathtt{in}\ N : \underline{C}}$$

    and similarly for homomorphism terms

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers

  - extend the equational theory of **value types** with

$$\Gamma \vdash U \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle = A$$

  - extend the eq. th. of **comp.** and **hom. terms** with $\beta\eta$-equations

  - extend the eq. th. of **comp. terms** with unfolding of ops.

$$\begin{aligned}&\Gamma \vdash \mathtt{op}_V^{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}(y.M)\\ &\quad = \mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}(V_{\mathsf{op}} \langle V, \lambda\, y.\mathtt{thunk}\, M \rangle) : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle\end{aligned}$$

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers

  - extend the equational theory of **value types** with

$$\Gamma \vdash U \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle = A$$

  - extend the eq. th. of **comp.** and **hom. terms** with $\beta\eta$-equations

  - extend the eq. th. of **comp. terms** with unfolding of ops.

$$\Gamma \vdash \mathtt{op}_V^{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}(y.M)$$

$$= \mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}\left(V_{\mathsf{op}}\langle V, \lambda\, y.\mathtt{thunk}\, M\rangle\right) : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle$$

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers

  - extend the equational theory of **value types** with

$$\Gamma \vdash U \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle = A$$

  - extend the eq. th. of **comp.** and **hom. terms** with $\beta\eta$-equations

  - extend the eq. th. of **comp. terms** with unfolding of ops.

$$\begin{aligned}\Gamma \vdash\ & \mathtt{op}_V^{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle}(y.M) \\ & = \mathtt{force}_{\langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle} \left( V_{\mathsf{op}} \langle V, \lambda\, y. \mathtt{thunk}\, M \rangle \right) : \langle A, \{V_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \rangle\end{aligned}$$

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers

  - we can then routinely derive the **handling construct**

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

  using **sequential composition**, thunking, and forcing:

$$\texttt{force}_{\underline{C}}\left(\texttt{thunk}\left(\underbrace{M\ \texttt{to}\ y\!:\!A\ \texttt{in}\ \left(\texttt{force}_{\langle U\underline{C}, \{V_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}}\rangle}\left(\texttt{thunk}\, N_{\mathrm{ret}}\right)\right)}_{\text{has type } \langle U\underline{C}, \{V_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}}\rangle}\right)\right)$$

  where $\langle U\underline{C}, \{V_{\mathrm{op}}\}_{\mathrm{op}\in\mathcal{S}_{\mathrm{eff}}}\rangle$ is derived from $\{\mathrm{op}_x(x') \mapsto N_{\mathrm{op}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}}$

  - satisfies the standard $\beta$-equations for handling

  - **handling into values** can be derived analogously

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - we can then routinely derive the **handling construct**

    $$M \,\texttt{handled with}\, \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \,\texttt{to}\, y\!:\!A \,\texttt{in}_{\underline{C}}\, N_{\mathsf{ret}}$$

    using **sequential composition**, thunking, and forcing:

$$\texttt{force}_{\underline{C}}\left(\texttt{thunk}\left(\underbrace{M \,\texttt{to}\, y\!:\!A \,\texttt{in}\, \left(\texttt{force}_{\langle U\underline{C},\{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\left(\texttt{thunk}\, N_{\mathsf{ret}}\right)\right)}_{\text{has type } \langle U\underline{C},\{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\right)\right)$$

    where $\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle$ is derived from $\{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}}$

  - satisfies the standard $\beta$-equations for handling
  - **handling into values** can be derived analogously

# Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - we can then routinely derive the **handling construct**

    $$M\ \texttt{handled with}\ \{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\,\in\,\mathcal{S}_{\mathsf{eff}}}\ \texttt{to}\ y\!:\!A\ \texttt{in}_{\underline{C}}\ N_{\mathsf{ret}}$$

    using **sequential composition**, thunking, and forcing:

$$\texttt{force}_{\underline{C}}\left(\texttt{thunk}\left(\underbrace{M\ \texttt{to}\ y\!:\!A\ \texttt{in}\ \left(\texttt{force}_{\langle U\underline{C},\{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\left(\texttt{thunk}\, N_{\mathsf{ret}}\right)\right)}_{\text{has type } \langle U\underline{C},\{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle}\right)\right)$$

    where $\langle U\underline{C}, \{V_{\mathsf{op}}\}_{\mathsf{op}\in\mathcal{S}_{\mathsf{eff}}}\rangle$ is derived from $\{\mathsf{op}_x(x') \mapsto N_{\mathsf{op}}\}_{\mathsf{op}\,\in\,\mathcal{S}_{\mathsf{eff}}}$

  - satisfies the standard $\beta$-equations for handling
  - **handling into values** can be derived analogously

# Conclusion

- In this talk, we saw that
  - handlers are useful for defining preds./types on computations
    - more generally, homomorphic type dep. on comps. is natural
    - this naturality was also observed in [Pédrot,Tabareau'17]
  - unsoundness problems can arise when accommodating handlers
    - handlers defined at term-level, while denoting algebras
  - handlers admit a principled type-based treatment
    - conventional term-level def. is derivable using seq. comp.

- Future work includes
  - general account of defining predicates on alg. effects
  - operational semantics (complex values + eq. for ops.)
  - presentations of the calculus without hom. terms (eq. proof obl.)

Thank you!

Questions?