Handling Fibred Algebraic Effects

Danel Ahman

INRIA Paris

POPL 2018

January 10, 2018

Dependent Types

and

Program Specs.

Algebraic Effects

and

Effect Handlers

Dependent Types

and

Program Specs.


Handling Fibred Algebraic Effects

DANEL AHMAN, Inria Paris, France

Algebraic Effects

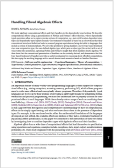
and

Effect Handlers

Dependent Types

and

Program Specs.


Handling Fibred Algebraic Effects

DANEL AHMAN, Inria Paris, France

1 INTRODUCTION

7

Algebraic Effects

and

Effect Handlers

What is it useful for?

How to do it?

Outline

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

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

Outline

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

Algebraic effects

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

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

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

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

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

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

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

Algebraic effects

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

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

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

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

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

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

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

Algebraic effects

- 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
  - **operation symbols** – representing the **sources** of effects

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

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

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

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

Handlers of algebraic effects

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

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

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

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

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

and

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

Handlers of algebraic effects

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

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

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

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

and

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

Handlers of algebraic effects

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

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

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

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

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

$$=$$

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

and

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

Handlers of algebraic effects

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

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

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

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

and

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

Outline

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

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

A core dependently typed effectful calculus

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

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

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

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

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

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

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

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

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

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

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

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

Outline

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

The calculus we work in

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

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

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

$$M \texttt{ handled with } \{\mathrm{op}_{x_v}(x_k) \mapsto V_{\mathrm{op}}; \overrightarrow{W_{\mathrm{eq}}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}} \texttt{ to } y : A \texttt{ in}_{B}\ V_{\mathrm{ret}}$$

The calculus we work in

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

    M `handled with` $\{\mathrm{op}_{x_v}(x_k) \mapsto N_{\mathrm{op}} ; \overrightarrow{W_{\mathrm{eq}}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}}$ `to` $y : A$ $\mathtt{in}_{\underline{C}}$ N_{ret}
  - a **derivable** "into-values" variant of **handlers** and **handling**

    M `handled with` $\{\mathrm{op}_{x_v}(x_k) \mapsto V_{\mathrm{op}} ; \overrightarrow{W_{\mathrm{eq}}}\}_{\mathrm{op} \in \mathcal{S}_{\mathrm{eff}}}$ `to` $y : A$ $\mathtt{in}_{B}$ V_{ret}

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!

- They help us to reason about effectful computations $M : FA$
  - Can be used to define **predicates** $P : UFA \to \mathcal{U}$ by
    1) equipping $\mathcal{U}$ (or a resp. type) with an **algebra** structure
    2) **handling** the given computation using that algebra
  - Intuitively, $P\ (\mathtt{thunk}\, M)$ computes a **proof obligation** for M
  - We discuss **three examples** of such predicates

- Also, can be an alternative to mon. reification for **rel. reasoning**
  - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \to A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for **reification-based** relational reasoning

Reasoning about effectful computations

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

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!

- They help us to reason about effectful computations $M : FA$
  - Can be used to define **predicates** $P : UFA \to \mathcal{U}$ by
    1) equipping $\mathcal{U}$ (or a resp. type) with an **algebra** structure
    2) **handling** the given computation using that algebra
  - Intuitively, $P\ (\mathtt{thunk}\, M)$ computes a **proof obligation** for M
  - We discuss **three examples** of such predicates

- Also, can be an alternative to mon. reification for **rel. reasoning**
  - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \to A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for **reification-based** relational reasoning

Reasoning about effectful computations

- Handlers are useful for **extrinsic reasoning**!

- They help us to reason about effectful computations $M : FA$
  - Can be used to define **predicates** $P : UFA \to \mathcal{U}$ by
    1) equipping $\mathcal{U}$ (or a resp. type) with an **algebra** structure
    2) **handling** the given computation using that algebra
  - Intuitively, $P\ (\mathtt{thunk}\, M)$ computes a **proof obligation** for M
  - We discuss **three examples** of such predicates

- Also, can be an alternative to mon. reification for **rel. reasoning**
  - E.g., relating **stateful comps.** $M, N : FA$ as **functions** $S \to A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for **reification-based** relational reasoning

Ex1: Lifting predicates to effectful comps.

- Given a predicate $P : A \to \mathcal{U}$ 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

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

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

- $\Box P$ 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 only 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

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

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

Ex1: Lifting predicates to effectful comps.

- Given a predicate $P : A \to \mathcal{U}$ 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 only 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

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

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

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

- To get $\Diamond P$, we only 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

$$V_{\text{get}}\,, V_{\text{put}} \quad \text{on} \quad S \to \mathcal{U} \times S \qquad\qquad \text{and} \qquad\qquad V_{\text{ret}} \ \text{“}{=}\text{”}\ Q$$

  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**

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

  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

$$V_{\mathrm{get}}\,,\, V_{\mathrm{put}} \quad \text{on} \quad S \to \mathcal{U} \times S \qquad \text{and} \qquad V_{\mathrm{ret}} \ \text{“}=\text{”}\ Q$$

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

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

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

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

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

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

  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

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

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

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

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

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

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 **relation** between **comps.** and **protocols**

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

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

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

given by (using pattern-matching lambda notation)

$$\begin{array}{lcl} \text{read}(x_k) & \mapsto & \lambda \left\{ (\mathtt{r}\ x_{pr}) \to \widehat{\Pi}\, y \,{:}\, \text{El}(\widehat{\text{Chr}})\,.\, x_k\ y\ (x_{pr}\ y)\ ; \right. \\ & & \left. \quad \_ \qquad\qquad \to \widehat{0} \right\} \\[1em] \text{write}_{x_v}(x_k) & \mapsto & \lambda \left\{ (\mathtt{w}\ P\ x_{pr}) \to \widehat{\Sigma}\, y \,{:}\, \text{El}(P\, x_v)\,.\, x_k \star x_{pr}\ ; \right. \\ & & \left. \quad \_ \qquad\qquad \to \widehat{0} \right\} \end{array}$$

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 **relation** between **comps.** and **protocols**

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

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

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

given by (using pattern-matching lambda notation)

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

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

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

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

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

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

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

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

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

given by (using pattern-matching lambda notation)

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

Outline

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

    [A., Ghani, Plotkin'16]

- What can we gain from handlers + dependent types?
  - Modular programming with handlers + expressiveness of d. types
  - **Extrinsic reasoning** about effectful computations

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

Extending the FoSSaCS'16 calculus

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

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

  - 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

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

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

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

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

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

Extending the FoSSaCS'16 calculus

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

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

  - we extend the **computation terms** with

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- 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 by extending the FoSSaCS'16 **computation 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 a **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.,

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

- 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 denotational semantics

How to proceed?

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

How to proceed?

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

Take 2: A type-level treatment of handlers

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

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

  where

    - A is the **carrier** value type
    - $\overrightarrow{V_{\mathsf{op}}}$ 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

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

$$\stackrel{\text{def}}{=}$$

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

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

Take 2: A type-level treatment of handlers

- Instead, we extend the FoSSaCS'16 **computation types** with
  - a **user-defined algebra type**
$$\underline{C}\,,\underline{D} \;\; ::= \;\; \ldots \;\; \mid \;\; \langle A; \overrightarrow{V_{\mathsf{op}}}; \overrightarrow{W_{\mathsf{eq}}} \rangle$$
    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

$$M \; \mathtt{handled\ with} \; \{\mathsf{op}_{x_v}(x_k) \mapsto N_{\mathsf{op}}; \overrightarrow{W_{\mathsf{eq}}}\}_{\mathsf{op} \in \mathcal{S}_{\mathsf{eff}}} \; \mathtt{to} \; y : A \; \mathtt{in}_{\underline{C}} \; N_{\mathsf{ret}}$$
$$\stackrel{\mathrm{def}}{=}$$
$$\mathtt{force}_{\underline{C}}(\mathtt{thunk}\,(\underbrace{M \; \mathtt{to} \; y : A \; \mathtt{in} \; \mathtt{force}_{\langle U\underline{C}; \overrightarrow{V_{N_{\mathsf{op}}}}; \overrightarrow{W_{\mathsf{eq}}} \rangle}(\mathtt{thunk}\, N_{\mathsf{ret}})}_{\text{temporarily working at type } \langle U\underline{C}; \overrightarrow{V_{N_{\mathsf{op}}}}; \overrightarrow{W_{\mathsf{eq}}} \rangle}))$$

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

Take 2: A type-level treatment of handlers

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

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

  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

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

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

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

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

Take 2: A type-level treatment of handlers

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

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

    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

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

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

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

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

Conclusion

- In conclusion
  - handlers are natural for defining **predicates on computations**
    - lifting predicates from return values to computations
    - Dijkstra's weakest precondition semantics of state
    - specifying patterns of allowed (I/O)-effects
  - they admit a principled **type-based treatment**

- See the paper for
  - **formal details** of what I have shown you today
  - families fibrations based **denotational semantics**
  - discussion about the calculus's inherent **extensional nature**
  - **Agda code** for the example predicates $P : UFA \to \mathcal{U}$

Thank you!

Questions?