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 \((T, \eta, (-)^\dagger)\)
  \[ \eta_A : A \rightarrow TA \quad (f : A \rightarrow TB)^\dagger_{A,B} : TA \rightarrow TB \]

- Plotkin and Power showed that most of these monads arise from
  - operations – representing sources of effects
    \[ \text{raise} : \text{Exc} \rightarrow 0 \quad \text{read} : \text{Loc} \rightarrow \text{Val} \quad \text{write} : \text{Loc} \times \text{Val} \rightarrow 1 \]
  - equations – describing the computational behaviour
    \[ \ell : \text{Loc} \mid w : 1 \vdash \text{read}_\ell(x.\text{write}_{\ell,x}(w(\star))) = w(\star) \]

- 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 \((T, \eta, (\_)^\dagger)\)

\[
\eta_A : A \to TA \quad (f : A \to TB)^\dagger_{A,B} : TA \to TB
\]

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

  \[
  \text{raise} : \text{Exc} \longrightarrow 0 \quad \text{read} : \text{Loc} \longrightarrow \text{Val} \quad \text{write} : \text{Loc} \times \text{Val} \longrightarrow 1
  \]

  \- equations – describing the computational behaviour

\[
\ell : \text{Loc} \mid w : 1 \vdash \text{read}_\ell (x.\text{write}_{\langle \ell, x \rangle} (w(\star))) = w(\star)
\]

- 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 \((T, \eta, (\_)^\dagger)\)

\[ \eta_A : A \rightarrow TA \quad (f : A \rightarrow TB)^{\dagger}_{A,B} : TA \rightarrow TB \]

• Plotkin and Power showed that most of these monads arise from
  • operations – representing sources of effects
    
    \[ \text{raise} : \text{Exc} \rightarrow 0 \quad \text{read} : \text{Loc} \rightarrow \text{Val} \quad \text{write} : \text{Loc} \times \text{Val} \rightarrow 1 \]
  
  • equations – describing the computational behaviour

\[ \ell : \text{Loc} \mid w : 1 \vdash \text{read}_\ell (x.\text{write}_{\langle \ell, x \rangle} (w(\star))) = w(\star) \]

• 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 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

\[
M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{op} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C N_{\text{ret}}
\]

denoting the homomorphism \( FA \rightarrow \{ \text{op}_x(x') \mapsto N_{op} \}_{\text{op} \in S_{\text{eff}}} \)

\[(\text{op}_V(y.M)) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C N_{\text{ret}} = N_{op}[V/x][\lambda y : O . \text{thunk}(M \text{ handled with } \ldots )/x']\]

and

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

\[
M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C 
\]

denoting the homomorphism \( FA \rightarrow \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \)

\[
(\text{op}_V(y.M)) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C 
\]

\[
= N_{\text{op}}[V/x][\lambda y : O \text{. thunk}(M \text{ handled with } \ldots)/x']
\]

and

\[
(\text{return } V) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C 
\]

\[
= N_{\text{ret}}[V/y]
\]
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

\[
M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : \mathsf{A} \text{ in}_C N_{\text{ret}}
\]

denoting the **homomorphism** \( FA \longrightarrow \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \)

\[
(\text{op}_V(y.M)) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : \mathsf{A} \text{ in}_C N_{\text{ret}}
\]
\[
= N_{\text{op}}[V/x][\lambda y : \mathsf{O} . \text{thunk}(M \text{ handled with } \ldots )]/x'
\]

and

\[
(\text{return } V) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : \mathsf{A} \text{ in}_C N_{\text{ret}} = N_{\text{ret}}[V/y]
\]
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

\[ M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in } C \]

denoting the **homomorphism** \( FA \rightarrow \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \)

\[(\text{op}_V(y.M)) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in } C \]

\[ = N_{\text{op}}[V/x][\lambda y:O. \text{thunk}(M \text{ handled with } \ldots)/x'] \]

and

\[(\text{return } V) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y:A \text{ in } C \]

\[ = N_{\text{ret}}[V/y] \]
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 \((\Gamma \vdash C)\)

  \[
  A, B ::= \ldots \mid UC \quad C, D ::= FA \mid \Pi x:A.C \mid \Sigma x:A.C
  \]

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

  \[
  V, W ::= x \mid \ldots \mid \text{thunk } M
  \]

- Computation terms \((\Gamma \vdash M : C)\)

  \[
  M, N ::= \text{return } V \mid M \text{ to } x:A \text{ in}_C N \mid \lambda x:A.M \mid M V
  \]
  \[
  \mid \langle V, M \rangle \mid M \text{ to } (x:A, z:C) \text{ in}_D K \mid \text{force}_C V
  \]

- Homomorphism terms \((\Gamma \mid z:C \vdash K : D)\)

  \[
  K, L ::= z \mid K \text{ to } x:A \text{ in}_C M \mid \ldots \quad \text{(stacks, eval. ctxs.)}
  \]
A core dependently typed effectful calculus

- (Model-theoretically) natural extension of type theory
  - clear distinction between values and computations (CBPV, EEC)

- **Value types** \((Γ ⊢ A)\) and **computation types** \((Γ ⊢ C)\)

  \[ A, B ::= \ldots \mid UC \quad C, D ::= FA \mid Π x : A . C \mid Σ x : A . C \]

- **Value terms** \((Γ ⊢ V : A)\)

  \[ V, W ::= x \mid \ldots \mid \text{thunk} \ M \]

- **Computation terms** \((Γ ⊢ M : C)\)

  \[ M, N ::= \text{return} \ V \mid M \text{ to} x : A \text{ in}_C N \mid λ x : A . M \mid M \ V \]
  \[ \mid \langle V, M \rangle \mid M \text{ to} (x : A, z : C) \text{ in}_D K \mid \text{force}_C V \]

- **Homomorphism terms** \((Γ \mid z : C ⊢ K : D)\)

  \[ K, L ::= z \mid K \text{ to} x : A \text{ in}_C M \mid \ldots \] (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 ($\Gamma \vdash C$)

  $$A, B ::= \ldots \mid UC \quad C, D ::= FA \mid \Pi x : A. C \mid \Sigma x : A. C$$

• Value terms ($\Gamma \vdash V : A$)

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

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

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

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

  $$K, L ::= z \mid K \to x : A \text{ in}_C M \mid \ldots \quad \text{(stacks, eval. ctxs.)}$$
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 ($\Gamma \vdash C$)
  
  $$A, B ::= \ldots \mid UC$$  
  
  $$C, D ::= FA \mid \Pi x:A . C \mid \Sigma x:A . C$$

• Value terms ($\Gamma \vdash V : A$)
  
  $$V, W ::= x \mid \ldots \mid \text{thunk } M$$

• Computation terms ($\Gamma \vdash M : C$)
  
  $$M, N ::= \text{return } V \mid M \text{ to } x:A \text{ in}_C N \mid \lambda x:A . M \mid M V \mid \langle V, M \rangle \mid M \text{ to } (x:A, z:C) \text{ in}_D K \mid \text{force}_C V$$

• Homomorphism terms ($\Gamma \mid z : C \vdash K : D$)
  
  $$K, L ::= z \mid K \text{ to } x:A \text{ in}_C M \mid \ldots$$
  
  (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** ($\Gamma \vdash C$)

\[
A, B ::= \ldots \mid UC \\
C, D ::= FA \mid \Pi x:A.C \mid \Sigma x:A.C
\]

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

\[
V, W ::= x \mid \ldots \mid \text{thunk } M
\]

**Computation terms** ($\Gamma \vdash M : C$)

\[
M, N ::= \text{return } V \mid M \text{ to } x:A \text{ in}_C N \mid \lambda x:A.M \mid M.V \\
| \langle V, M \rangle \mid M \text{ to } (x:A, z:C) \text{ in}_D K \mid \text{force}_C V
\]

**Homomorphism terms** ($\Gamma \mid z:C \vdash K : D$)

\[
K, L ::= z \mid K \text{ to } x:A \text{ in}_C M \mid \ldots \quad \text{(stacks, eval. ctxs.)}
\]
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**
    
    \[ M \text{ handled with } \{ \text{op}_x(x') \mapsto V_{\text{op}} \}_{\text{op} \in \mathcal{S}_\text{eff}} \text{ to } y : A \text{ in } B \ V_{\text{ret}} \]

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

- Then we can define **predicates** \( P : UFA \rightarrow \mathcal{U} \) (a value term) by
  - equipping \( \mathcal{U} \) with an **algebra** structure
  - handling the given computation using that algebra
  - intuitively, \( P (\text{thunk } M) \) 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**
    \[ M \text{ handled with } \{ \text{op}_x(x') \mapsto V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in } B \rightarrow V_{\text{ret}} \]

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

- Then we can define **predicates** \( P : UFA \rightarrow \mathcal{U} \) (a value term) by
  - equipping \( \mathcal{U} \) with an **algebra** structure
  - **handling** the given computation using that algebra
  - intuitively, \( P \text{ (thunk } M) \) 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
    \[ M \text{ handled with } \{ \text{op}_x(x') \mapsto V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in } B \ V_{\text{ret}} \]

• 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, \( P \left( \text{thunk } M \right) \) 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 \( \hat{P} : UFA \to \mathcal{U} \) on (I/O)-comps. as

  \[
  \lambda y : UFA. (force y) \text{ handled with } \{ \ldots \}_{op \in S_{IO}} \text{ to } y' : A \text{ in } u \ P y'
  \]

  using the handler given by

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

  \[
  V_{\text{write}} \overset{\text{def}}{=} \lambda y : (\Sigma x : \text{Chr}. 1 \to \mathcal{U}). (\text{snd y})
  \]

- \( \hat{P} \) is similar to the necessity modality from Evaluation Logic

  \[
  \Gamma \vdash \text{El}(\hat{P} (\text{thunk (read}^{FA}(x.\text{return W})))) = \Pi x : \text{Chr}. P \ W
  \]

- To get possibility mod., replace \text{v-pi-code} with \text{v-sigma-code}
Lifting predicates to effectful comps.

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

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

$$\lambda y : UFA. \text{(force } y\text{)} \text{ handled with } \{\ldots\}_{op \in S_{IO}} \text{ to } y' : A \text{ in } u \ P \ y'$$

using the handler given by

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

$$V_{write} \overset{\text{def}}{=} \lambda y : (\Sigma x : \text{Chr}. 1 \rightarrow \mathcal{U}) \cdot (\text{snd } y) \ast$$

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

$$\Gamma \vdash El(\hat{P} \ (\text{thunk } (\text{read}^{FA}(x \cdot \text{return } W)))) = \Pi x : \text{Chr} . \ P \ W$$

- To get possibility mod., replace $v\text{-pi-code}$ with $v\text{-sigma-code}$
Lifting predicates to effectful comps.

- Given a predicate $P : A \rightarrow \mathcal{U}$ on return values, we define a predicate $\hat{P} : UFA \rightarrow \mathcal{U}$ on $(I/O)$-comps. as

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

  using the handler given by

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

  $$V_{\text{write}} \overset{\text{def}}{=} \lambda y : (\Sigma x : \text{Chr}. 1 \rightarrow \mathcal{U}). (\text{snd } y) \star$$

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

  $$\Gamma \vdash \text{El}(\hat{P} (\text{thunk} (\text{read}^{FA}(x. \text{return } W)))) = \Pi x : \text{Chr}. P W$$

- To get possibility mod., replace $v\text{-pi-code}$ with $v\text{-sigma-code}$
Lifting predicates to effectful comps.

- Given a predicate $P : A \rightarrow U$ on return values, we define a predicate $\hat{P} : UFA \rightarrow U$ on (I/O)-comps. as

  $$\lambda y : UFA. (\text{force } y) \text{ handled with } \{\ldots\}_{op \in S_{\text{IO}}} \text{ to } y' : A \text{ in } U \ P y'$$

  using the handler given by

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

  $$V_{\text{write}} \overset{\text{def}}{=} \lambda y : (\Sigma x : \text{Chr. } 1 \rightarrow U). (\text{snd } y) *$$

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

  $$\Gamma \vdash \text{El}(\hat{P} (\text{thunk (read}^{FA}(x.\text{return } W)))) = \Pi x : \text{Chr. } P \ W$$

- 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
  \[ Q : A \rightarrow St \rightarrow U \]
  we define a precondition for stateful comps. on initial states
  \[ wp_Q : UFA \rightarrow St \rightarrow U \]
  by
  i) handling the given comp. into a state-passing function using
  \( V_{get}, V_{put} \) on \( St \rightarrow (U \times St) \) and \( V_{ret} "=" V_Q \)
  ii) feeding in the initial state, and iii) projecting out the proposition

- Then \( wp_Q \) satisfies the expected properties, e.g.,
  \[ \Gamma \vdash wp_Q \left( \text{thunk}(\text{return} \ V) \right) = \lambda x_S : St. \ wp_Q (thunk M) \ V_S \ : St \rightarrow U \]
  \[ \Gamma \vdash wp_Q \left( \text{thunk}(\text{put}^{FA} \ V_S (M)) \right) = \lambda x_S : St. \ wp_Q (thunk M) \ V_S \ : St \rightarrow U \]
Dijkstra’s weakest precondition semantics

- Given a postcondition on return values and final states
  \[ Q : A \rightarrow St \rightarrow U \]
  we define a precondition for stateful comps. on initial states
  \[ \text{wp}_Q : UFA \rightarrow St \rightarrow U \]
  by
  \[ \begin{align*}
  & \text{i)} \text{ handling the given comp. into a state-passing function using } V_{get}, V_{put} \text{ on } St \rightarrow (U \times St) \text{ and } V_{ret} "=" V_Q \\
  & \text{ii)} \text{ feeding in the initial state, and } \text{iii)} \text{ projecting out the proposition }
  \end{align*} \]
- Then \( \text{wp}_Q \) satisfies the expected properties, e.g.,
  \[ \Gamma \vdash \text{wp}_Q \left( \text{thunk} \left( \text{return} \ V \right) \right) = \lambda x_S : St. Q \ V \ x_S : St \rightarrow U \]
  \[ \Gamma \vdash \text{wp}_Q \left( \text{thunk} \left( \text{put}_{V_S}^{FA} \left( M \right) \right) \right) = \lambda x_S : St. \text{wp}_Q \left( \text{thunk} \ M \right) \ V_S : St \rightarrow U \]
Dijkstra’s weakest precondition semantics

- Given a postcondition on return values and final states
  \[ Q : A \rightarrow St \rightarrow U \]
  we define a precondition for stateful comps. on initial states
  \[ wp_Q : UFA \rightarrow St \rightarrow U \]
  by
  
  i) handling the given comp. into a state-passing function using
  \[ V_{\text{get}}, V_{\text{put}} \quad \text{on} \quad St \rightarrow (U \times St) \quad \text{and} \quad V_{\text{ret}} "=" V_Q \]
  ii) feeding in the initial state, and
  iii) projecting out the proposition

- Then \( wp_Q \) satisfies the expected properties, e.g.,
  \[ \Gamma \vdash wp_Q \left( \text{thunk (return } V \right) )) = \lambda x_S : St. Q V x_S : St \rightarrow U \]
  \[ \Gamma \vdash wp_Q \left( \text{thunk (put}_{V_S}^{FA}(M)) \right) = \lambda x_S : St. wp_Q \left( \text{thunk } M \right) V_S : St \rightarrow U \]
Dijkstra’s weakest precondition semantics

- Given a postcondition on return values and final states
  \[ Q : A \rightarrow St \rightarrow \mathcal{U} \]
  we define a precondition for stateful comps. on initial states
  \[ wp_Q : UFA \rightarrow St \rightarrow \mathcal{U} \]
  by
  
  i) handling the given comp. into a state-passing function using
  \[ V_{\text{get}}, V_{\text{put}} \text{ on } St \rightarrow (\mathcal{U} \times St) \text{ and } V_{\text{ret}} \text{ }\]
  ii) feeding in the initial state, and
  iii) projecting out the proposition

- Then \( wp_Q \) satisfies the expected properties, e.g.,

  \[ \Gamma \vdash wp_Q \left( \text{thunk} \left( \text{return} \ V \right) \right) = \lambda \ x_S : St. \ Q \ V \ x_S : St \rightarrow \mathcal{U} \]
  \[ \Gamma \vdash wp_Q \left( \text{thunk} \left( \text{put}^{FA} (M) \right) \right) = \lambda \ x_S : St. \ wp_Q \left( \text{thunk} \ M \right) \ V_S : St \rightarrow \mathcal{U} \]
Specifying allowed patterns of I/O-effects

- We assume an inductive type Protocol, given by

\[ e : \text{Protocol} \quad \quad r : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol} \]
\[ w : (\text{Chr} \rightarrow U) \times \text{Protocol} \rightarrow \text{Protocol} \]

and potentially also by \(\land, \lor, \ldots\)

- Given a protocol \(\text{Pr} : \text{Protocol}\), we define

\[ \hat{\text{Pr}} : \text{UFA} \rightarrow U \]

by handling a given comp. using

\[ V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \rightarrow U \]

where

\[ V_{\text{read}} \langle V, V_{\text{rk}} \rangle (r \text{ Pr}') \quad \text{def} \quad v\text{-pi-code}(\text{chr-code}, y.(V_{\text{rk}}y)(\text{Pr}'y)) \]
\[ V_{\text{write}} \langle V, V_{\text{wk}} \rangle (w \langle P, \text{Pr}' \rangle) \quad \text{def} \quad v\text{-sigma-code}(P \ V, y. V_{\text{wk}} \ast \text{Pr}') \]
\[ \quad \text{def} \quad \text{empty-code} \]
Specifying allowed patterns of I/O-effects

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

\[
\begin{align*}
  e : \text{Protocol} & \quad r : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol} \\
  w : (\text{Chr} \rightarrow \mathcal{U}) \times \text{Protocol} & \rightarrow \text{Protocol}
\end{align*}
\]

and potentially also by \&, \lor, \ldots

- Given a **protocol** \( Pr : \text{Protocol} \), we define

\[
\hat{Pr} : \text{UFA} \rightarrow \mathcal{U}
\]

by handling a given comp. using

\[
V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \rightarrow \mathcal{U}
\]

where

\[
\begin{align*}
V_{\text{read}} \langle V, V_{rk} \rangle (r \ Pr') & \overset{\text{def}}{=} v\text{-pi-code}(\text{chr-code}, y. (V_{rk} y)(Pr' y)) \\
V_{\text{write}} \langle V, V_{wk} \rangle (w \langle P, Pr' \rangle) & \overset{\text{def}}{=} v\text{-sigma-code}(P V, y. V_{wk} \star Pr') \\
- & \overset{\text{def}}{=} \text{empty-code}
\end{align*}
\]
Specifying allowed patterns of I/O-effects

- We assume an **inductive type** Protocol, given by
  
  \[ e : \text{Protocol} \quad r : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol} \]
  
  \[ w : (\text{Chr} \rightarrow \mathcal{U}) \times \text{Protocol} \rightarrow \text{Protocol} \]

  and potentially also by \( \land, \lor, \ldots \)

- Given a **protocol** \( \text{Pr} : \text{Protocol} \), we define
  
  \[ \hat{\text{Pr}} : \text{UFA} \rightarrow \mathcal{U} \]

  by handling a given comp. using

  \[ V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \rightarrow \mathcal{U} \]

  where

  \[ V_{\text{read}} \langle V, V_{\text{rk}} \rangle (r \text{Pr}') \]

  \( \overset{\text{def}}{=} \)

  \[ v\text{-pi-code}(\text{chr-code}, y.(V_{\text{rk}} y)(\text{Pr}' y)) \]

  \[ V_{\text{write}} \langle V, V_{\text{wk}} \rangle (w \langle P, \text{Pr}' \rangle) \]

  \( \overset{\text{def}}{=} \)

  \[ v\text{-sigma-code}(P V, y. V_{\text{wk}} \ast \text{Pr}') \]

  \( \overset{\text{def}}{=} \)

  \[ \text{empty-code} \]
Specifying allowed patterns of I/O-effects

- We assume an inductive type Protocol, given by
  
  $$e : \text{Protocol} \quad r : (\text{Chr} \to \text{Protocol}) \to \text{Protocol}$$
  
  $$w : (\text{Chr} \to \mathcal{U}) \times \text{Protocol} \to \text{Protocol}$$
  
  and potentially also by $\land, \lor, \ldots$.

- Given a protocol $Pr : \text{Protocol}$, we define
  
  $$\hat{Pr} : \text{UFA} \to \mathcal{U}$$
  
  by handling a given comp. using
  
  $$V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \to \mathcal{U}$$
  
  where
  
  $$V_{\text{read}} \langle V, V_{rk} \rangle (r Pr') \overset{\text{def}}{=} v\text{-pi-code}(\text{chr-code}, y.(V_{rk}y)(Pr'y))$$
  
  $$V_{\text{write}} \langle V, V_{wk} \rangle (w \langle P, Pr' \rangle) \overset{\text{def}}{=} v\text{-sigma-code}(P V, y.V_{wk} \ast Pr')$$
  
  $$\text{def} = \text{empty-code}$$
Specifying allowed patterns of I/O-effects

- We assume an **inductive type** `Protocol`, given by
  
  \[
  e : \text{Protocol} \quad \quad \quad \quad \quad r : (\text{Chr} \to \text{Protocol}) \to \text{Protocol}
  \]
  
  \[
  w : (\text{Chr} \to \mathcal{U}) \times \text{Protocol} \to \text{Protocol}
  \]
  
  and potentially also by `∧`, `∨`, . . .

- Given a **protocol** `Pr : \text{Protocol}`, we define
  
  \[
  \hat{Pr} : \mathcal{UFA} \to \mathcal{U}
  \]
  
  by handling a given comp. using
  
  \[
  V_{\text{read}}, V_{\text{write}} \quad \text{on} \quad \text{Protocol} \to \mathcal{U}
  \]
  
  where
  
  \[
  V_{\text{read}} \langle V, V_{rk} \rangle (r \ Pr') \quad \overset{\text{def}}{=} \quad \text{v-pi-code}(\text{chr-code}, y.(V_{rk} y)(Pr' y))
  \]
  
  \[
  V_{\text{write}} \langle V, V_{wk} \rangle (w \langle P, Pr' \rangle) \quad \overset{\text{def}}{=} \quad \text{v-sigma-code}(P \ V, y. V_{wk} \star Pr')
  \]
  
  \[
  - \quad \overset{\text{def}}{=} \quad \text{empty-code}
  \]
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 \((S_{\text{eff}}, E_{\text{eff}})\) in our calculus, we
  
  - extend its computation terms with algebraic operations

\[
\Gamma \vdash V : I \quad \Gamma \vdash C \quad \Gamma, y : O[V/x] \vdash M : C
\]

\[
\Gamma \vdash \text{op}_V^C(y.M) : C
\]

for every dep. typed op. symbol \(\text{op} : (x : I) \rightarrow O\) in \(S_{\text{eff}}\)

- include equations \(\Gamma \mid \Delta \vdash T_1 = T_2\) given in \(E_{\text{eff}}\)

- include a general algebraicity equation

\[
\Gamma \mid z : C \vdash K : D \quad \Gamma \vdash V : I \quad \Gamma, y : O[V/x] \vdash M : C
\]

\[
\Gamma \vdash K[\text{op}_V^C(y.M)/z] = \text{op}_V^D(y.K[M/z]) : D
\]
**Fibred algebraic effects**

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

\[
\Gamma \vdash V : I \quad \Gamma \vdash C \quad \Gamma, y : O[V/x] \vdash M : C
\]

\[
\Gamma \vdash \text{op}^{C}_V(y.M) : C
\]

for every dep. typed op. symbol \( \text{op} : (x:I) \rightarrow O \) in \( S_{\text{eff}} \)

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

- include a general algebraicity equation

\[
\Gamma \mid z : C \vdash K : D \quad \Gamma \vdash V : I \quad \Gamma, y : O[V/x] \vdash M : C
\]

\[
\Gamma \vdash K[\text{op}^{C}_V(y.M)/z] = \text{op}^{D}_V(y.K[M/z]) : D
\]
Fibred algebraic effects

- To include fib. alg. effects \((S_{\text{eff}}, E_{\text{eff}})\) in our calculus, we
  - extend its computation terms with **algebraic operations**
    \[
    \Gamma \vdash V : I \quad \Gamma \vdash C \quad \Gamma, y : O[V/x] \vdash M : C \\
    \Gamma \vdash \text{op}_{V}^{C}(y.M) : C
    \]
    for every dep. typed op. symbol \(\text{op} : (x : I) \rightarrow O\) in \(S_{\text{eff}}\)

- include **equations** \(\Gamma \mid \Delta \vdash T_1 = T_2\) given in \(E_{\text{eff}}\)

- include a general **algebraicity equation**
  \[
  \Gamma \mid z : C \vdash K : D \quad \Gamma \vdash V : I \quad \Gamma, y : O[V/x] \vdash M : C \\
  \Gamma \vdash K[\text{op}_{V}^{C}(y.M)/z] = \text{op}_{V}^{D}(y.K[M/z]) : D
  \]
**Handlers for fibred algebraic effects**

- **Take 1:** Let’s use their conventional term-level definition
  
  - include the handling construct for computation terms
    
    \[ M \text{ handled with } \{ \text{op}_x (x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in } \mathcal{C} \text{ N}_{\text{ret}} \]
  
  - as handling denotes a homomorphism, also for hom. terms
    
    \[ K \text{ handled with } \{ \text{op}_x (x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in } \mathcal{C} \text{ N}_{\text{ret}} \]
  
  - but then we can prove the unsound equation
    
    \[ \Gamma \vdash \text{write}^{F_1}_a (\text{return} \star) = \text{write}^{F_1}_z (\text{return} \star) : F_1 \]
    
    by handling
    
    \[ \text{write}^{F_1}_a (\text{return} \star) \]
    
    with
    
    \[ \text{write}_x (x') \mapsto \text{write}_z (\text{force} (x' \star)) \]
    
    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**
    
    \[ M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \} \text{ for } y:A \text{ in } C \text{ to } y:A \text{ in } N_{\text{ret}} \]

  - as handling denotes a homomorphism, also for **hom. terms**
    
    \[ K \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \} \text{ for } y:A \text{ in } C \text{ to } y:A \text{ in } N_{\text{ret}} \]

  - but then we can prove the **unsound equation**
    
    \[ \Gamma \vdash \text{write}_{F_1}^a(\text{return } \star) = \text{write}_{F_1}^z(\text{return } \star) : F_1 \]

    by handling

    \[ \text{write}_{F_1}^a(\text{return } \star) \]

    with

    \[ \text{write}_x(x') \mapsto \text{write}_z(\text{force } (x' \star)) \]

    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**
    
    $M$ handled with \( \{ \text{op}_x(x' \mapsto N_{\text{op}}) \}_{\text{op} \in S_{\text{eff}}} \) to \( y : A \text{ in } \mathcal{C} N_{\text{ret}} \)
  
  - as handling denotes a homomorphism, also for **hom. terms**
    
    $K$ handled with \( \{ \text{op}_x(x' \mapsto N_{\text{op}}) \}_{\text{op} \in S_{\text{eff}}} \) to \( y : A \text{ in } \mathcal{C} N_{\text{ret}} \)
  
  - but then we can prove the **unsound equation**
    
    \[ \Gamma \vdash \text{write}_{_{a}}^{F_{1}}(\text{return } \star) = \text{write}_{_{z}}^{F_{1}}(\text{return } \star) : F_{1} \]
    
    by handling
    
    \[ \text{write}_{_{a}}^{F_{1}}(\text{return } \star) \]
    
    with
    
    \[ \text{write}_{x}(x' \mapsto \text{write}_{z}(\text{force } (x' \star))) \]
    
    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**
    \[ M \text{ handled with } \{ op_x(x') \mapsto N_{op} \}_{op \in S_{eff}} \text{ to } y : A \text{ in } N_{ret} \]
  - as handling denotes a homomorphism, also for **hom. terms**
    \[ K \text{ handled with } \{ op_x(x') \mapsto N_{op} \}_{op \in S_{eff}} \text{ to } y : A \text{ in } N_{ret} \]
  - but then we can prove the **unsound equation**
    \[ \Gamma \vdash write^{F1}_a(\text{return} \star) = write^{F1}_z(\text{return} \star) : F1 \]

by handling

\[ write^{F1}_a(\text{return} \star) \]

with

\[ write_x(x') \mapsto write_z(\text{force} (x' \star)) \]

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**
    
    \[ M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in } \mathcal{C} N_{\text{ret}} \]
  
    - as handling denotes a homomorphism, also for **hom. terms**
      
      \[ K \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in } \mathcal{C} N_{\text{ret}} \]
  
    - but then we can prove the **unsound equation**
      
      \[ \Gamma \vdash \text{write}^{F_1}_{a}(\text{return} \star) = \text{write}^{F_1}_{z}(\text{return} \star) : F_1 \]

      by **handling**

      \[ \text{write}^{F_1}_{a}(\text{return} \star) \]

      with

      \[ \text{write}_x(x') \mapsto \text{write}_z(\text{force}(x' \star)) \]

      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)

    \[
    \Gamma \vdash A \quad \{ \Gamma \vdash V_{\text{op}} : (\Sigma x : I. O \rightarrow A) \rightarrow A\}_{\text{op} \in \mathcal{S}_{\text{eff}}} \]

    \[V_{\text{op}} \text{ satisfy the equations in } \mathcal{E}_{\text{eff}}\]

    \[\Gamma \vdash \langle A, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle\]

    - comps. of this type are **introduced** by force

    \[\langle A, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle \]

    - we introduce corresponding **elimination form**

    \[\Gamma \vdash M : \langle A, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle \quad \Gamma \vdash C \quad \Gamma, x : A \vdash N : C\]

    \[N \text{ behaves as a homomorphism in } x \text{ (i.e., commutes with ops.)}\]

    \[\Gamma \vdash M \text{ as } x : U\langle A, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle \text{ in } N : C\]

    and similarly for homomorphism terms.
• **Take 2:** A type-based treatment of handlers

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

    \[
    \Gamma \vdash A \quad \{ \Gamma \vdash V_{op} : (\Sigma x : I. O \to A) \to A \}_{op \in S_{eff}}
    \]

    \[
    V_{op} \text{ satisfy the equations in } E_{eff}
    \]

    \[
    \Gamma \vdash \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle
    \]

  - comps. of this type are **introduced** by force \( \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \) \( V \)

  - we introduce corresponding **elimination form**

    \[
    \Gamma \vdash M : \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \quad \Gamma \vdash C \quad \Gamma, x : A \vdash N : C
    \]

    \[
    N \text{ behaves as a homomorphism in } x \text{ (i.e., commutes with ops.)}
    \]

    \[
    \Gamma \vdash M \text{ as } x : U \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \text{ in } N : C
    \]

    and similarly for homomorphism terms
Handers for fibred algebraic effects ctd.

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

\[
\Gamma \vdash A \quad \{ \Gamma \vdash V_{op} : (\Sigma x : I.O \to A) \to A \}_{op \in S_{eff}} \quad V_{op} \text{ satisfy the equations in } E_{eff}
\]

\[
\Gamma \vdash \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle
\]

- comps. of this type are **introduced** by force \( \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \)

- we introduce corresponding **elimination form**

\[
\Gamma \vdash M : \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \quad \Gamma \vdash C \quad \Gamma, x : A \vdash N : C
\]

\( N \) behaves as a homomorphism in \( x \) (i.e., commutes with ops.)

\[
\Gamma \vdash M \text{ as } x : U \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \text{ in } N : C
\]

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)

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

  - comps. of this type are **introduced** by force \( \langle A, \{ V_{\text{op}} \}_{\text{op} \in \text{S}_{\text{eff}}} \rangle \)

  - we introduce corresponding **elimination form**

    \[
    \Gamma \vdash M : \langle A, \{ V_{\text{op}} \}_{\text{op} \in \text{S}_{\text{eff}}} \rangle \\
    \Gamma \vdash C \quad \Gamma, x : A \vdash N : C \\
    N \text{ behaves as a homomorphism in } x \text{ (i.e., commutes with ops.)} \\
    \Gamma \vdash M \text{ as } x : U\langle A, \{ V_{\text{op}} \}_{\text{op} \in \text{S}_{\text{eff}}} \rangle \text{ in } N : C
    \]

    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

    \[ \Gamma \vdash U \langle A, \{ V_{op} \}_{op \in S_{\text{eff}}} \rangle = A \]

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

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

    \[ \Gamma \vdash op_{V}^{\langle A, \{ V_{op} \}_{op \in S_{\text{eff}}} \rangle} (y.M) \]
    
    \[ = \text{force} \langle A, \{ V_{op} \}_{op \in S_{\text{eff}}} \rangle (V_{op} \langle V, \lambda y.\text{thunk } M \rangle) : \langle A, \{ V_{op} \}_{op \in S_{\text{eff}}} \rangle \]
**Handlers for fibred algebraic effects ctd.**

- **Take 2:** A type-based treatment of handlers
  - extend the equational theory of **value types** with
    \[
    \Gamma \vdash U \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle = A
    \]
  - extend the eq. th. of **comp.** and **hom. terms** with \(\beta\eta\)-equations
    \[
    \Gamma \vdash \text{force}_{\langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle} (V_{op} \langle V, \lambda y. \text{thunk } M \rangle) : \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle
    \]
  - extend the eq. th. of **comp. terms** with unfolding of ops.
Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - extend the equational theory of *value types* with
    \[ \Gamma \vdash U \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle = A \]
  - extend the eq. th. of *comp.* and *hom.* terms with $\beta\eta$-equations
  - extend the eq. th. of *comp. terms* with unfolding of ops.
    \[ \Gamma \vdash \text{force} \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle (V_{op} \langle V, \lambda y. \text{thunk} \ M \rangle) : \langle A, \{ V_{op} \}_{op \in S_{eff}} \rangle \]
Handlers for fibred algebraic effects ctd.

- **Take 2:** A type-based treatment of handlers
  - we can then routinely derive the **handling construct**
    
    $$M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y : A \text{ in}_{\mathcal{C}} N_{\text{ret}}$$
    
    using sequential composition, thunking, and forcing:
    
    $$\text{force}_{\mathcal{C}}(\text{thunk}(M \text{ to } y : A \text{ in } (\text{force}_{\langle U_{\mathcal{C}}, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle} (\text{thunk } N_{\text{ret}})))))$$
    
    has type $$\langle U_{\mathcal{C}}, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle$$
    
    where $$\langle U_{\mathcal{C}}, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle$$ is derived from $$\{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}}$$
    
    - satisfies the standard $\beta$-equations for handling
    
    - handling into values can be derived analogously
• **Take 2:** A type-based treatment of handlers

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

\[ M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y:A \text{ in}_C N_{\text{ret}} \]

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

\[
\text{force}_C (\text{thunk} (M \text{ to } y:A \text{ in } (\text{force}_{\langle UC, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle} (\text{thunk} N_{\text{ret}}))))
\]

has type \( \langle UC, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle \)

where \( \langle UC, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle \) is derived from \( \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \)

• 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**

  \[
  M \text{ handled with } \{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \text{ to } y:A \text{ in } \mathcal{C} N_{\text{ret}}
  \]

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

  \[
  \text{force}_\mathcal{C}(\text{thunk}(M \text{ to } y:A \text{ in } (\text{force}_{\langle UC, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle} (\text{thunk } N_{\text{ret}}))))
  \]

  has type \(\langle UC, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle\)

  where \(\langle UC, \{ V_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \rangle\) is derived from \(\{ \text{op}_x(x') \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_{\text{eff}}}\)

  - 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?