Handling Fibred Algebraic Effects

Danel Ahman
INRIA Paris

POPL 2018
January 10, 2018
Dependent Types and Logical Reasoning

Algebraic Effects and Effect Handlers
Dependent Types and Logical Reasoning

Algebraic Effects and Effect Handlers
Dependent Types
and
Logical Reasoning

What can we do?

Algebraic Effects
and
Effect Handlers

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 (has issues)
  - 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 (has issues)
    - Take 2: A new **type-level treatment** of handlers
Algebraic effects

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

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

- Plotkin and Power showed that most of these monads arise from
  - operation symbols – representing the sources of effects
    
    \[
    \text{raise} : \text{Exc} \to 0 \quad \text{get} : \text{Loc} \to \text{Val} \quad \text{put} : \text{Loc} \times \text{Val} \to 1
    \]
  - equations – describing the computational behaviour

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

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

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

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

- Plotkin and Power showed that most of these monads arise from
  - **operation symbols** – representing the **sources** of effects
    
    \[
    \text{raise} : \text{Exc} \to 0 \quad \text{get} : \text{Loc} \to \text{Val} \quad \text{put} : \text{Loc} \times \text{Val} \to 1
    \]

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

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

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

- 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
  - operation symbols – representing the sources of effects
    - raise : Exc $\rightarrow$ 0
    - get : Loc $\rightarrow$ Val
    - put : Loc $\times$ Val $\rightarrow$ 1
  - equations – describing the computational behaviour
    $$\ell : \text{Loc} \mid w : 1 \vdash \text{get}_\ell (x. \text{put}_{\langle \ell, x \rangle} (w(\star))) = w(\star)$$

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

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

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

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

interpreted using the homomorphism \( F A \longrightarrow \langle UC, f_{N_{\text{op}}} \rangle \), i.e.,

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

and

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

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

\[
M \text{ handled with } \left\{ \{op_{x_v}(x_k) \rightarrow N_{op}\}_{op \in S_{eff}; W_{eq}} \right\} \text{ to } y : A \text{ in } C \ N_{ret}
\]

interpreted using the homomorphism \( F A \rightarrow \langle UC, f_{N_{op}} \rangle \), i.e.,

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

and

\[
(\text{return } V) \text{ handled with } \{\ldots\}_{op \in S_{eff}} \text{ to } y:A \text{ in } C \ N_{ret} = N_{ret}[V/y]
\]
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 – stream redirection, state, rollbacks, concurrency, ...

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

\[
M \text{ handled with } \{(\text{op}_x(v(x_k)) \mapsto N_{\text{op}})_{\text{op} \in S_{\text{eff}}; \overrightarrow{\text{Weq}}} \} \text{ to } y : A \text{ in}_C N_{\text{ret}}
\]

interpreted using the **homomorphism** \(FA \rightarrow \langle UC, \overrightarrow{f_{\text{N}_{\text{op}}}} \rangle\), i.e.,

\[
\text{handled with } \{\ldots\}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in}_C N_{\text{ret}}
\]

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

and

\[
\text{handled with } \{\ldots\}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in}_C N_{\text{ret}} = N_{\text{ret}}[V/y]
\]
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 – stream redirection, state, rollbacks, concurrency, ...

- Usually included in languages using the handling construct

\[ M \text{ handled with } \{ \text{op}_{x_v}(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \xrightarrow{W_{\text{eq}}} \text{to } y : A \text{ in}_C N_{\text{ret}} \]

interpreted using the homomorphism \( FA \xrightarrow{f_{N_{\text{op}}}} \langle UC, \xrightarrow{f} \rangle \), i.e.,

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

and

\[
(\text{return } V) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in}_C N_{\text{ret}} = N_{\text{ret}}[V/y]
\]
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 (has issues)
  - Take 2: A new type-level treatment of handlers
A core dependently typed effectful calculus

- Natural extension of Martin-Löf’s (intensional) type theory
  - clear distinction between values and computations (CBPV, EEC)

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

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

- Value terms \( (Γ ⊢ V : A) \)

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

- Computation terms \( (Γ ⊢ M : C) \)

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

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

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

- Natural extension of Martin-Löf’s (intensional) type theory
  - clear distinction between values and computations (CBPV, EEC)

- Value types \((\Gamma \vdash A)\) and computation types \((\Gamma \vdash C)\)

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

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

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

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

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

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

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

- Natural extension of Martin-Löf’s (intensional) 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 \((\Gamma \vdash V : A)\)

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

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

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

- Natural extension of Martin-Löf’s (intensional) type theory
  - clear distinction between values and computations (CBPV, EEC)

- Value types ($\Gamma \vdash A$) and computation types ($\Gamma \vdash C$)

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

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

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

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

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

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

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

- Natural extension of Martin-Löf’s (intensional) 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 \((\Gamma \vdash V : A)\)
  \[
  V, W ::= \ldots \mid \text{thunk } M
  \]
- 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 \((\Gamma \mid z:C \vdash K : D)\)
  \[
  K, L ::= z \mid K \text{ to } x:A \text{ in } C M \mid \ldots \quad \text{(stack terms, eval. ctxs.)}
  \]
The calculus we propose in this paper . . .

• . . . is a variant of the FoSSaCS’16 calculus, with

  • a Tarski-style value universe \( \mathcal{U} \)
  • with codes written as \( \hat{\Pi}, \hat{\Sigma}, \hat{0}, \hat{1}, \ldots \)
  • but thinking of them as \( \forall, \exists, \bot, \top, \ldots \)

  • fibred algebraic effects
    • dep. typed operation symbols \( \text{op} : (x_v : I) \rightarrow O \)
    • ops. determine computation terms \( \text{op}_V^C(y : O[V/x_v].M) \)
    • effect equations determine definitional equations

• a derivable “into-comps.” variant of handlers and handling
  \( M \) handled with \( (\{\text{op}^{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}} ; \overrightarrow{W_{\text{eq}}}} \) to \( y : A \) in \( \mathcal{C} \ N_{\text{ret}} \)

• a derivable “into-values” variant of handlers and handling
  \( M \) handled with \( (\{\text{op}^{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op} \in \mathcal{S}_{\text{eff}} ; \overrightarrow{W_{\text{eq}}}} \) to \( y : A \) in \( \mathcal{B} \ V_{\text{ret}} \)
The calculus we propose in this paper . . .

- . . . is a variant of the FoSSaCS’16 calculus, with
  - a Tarski-style **value universe** \( \mathcal{U} \)
    - with **codes** written as \( \hat{\Pi}, \hat{\Sigma}, \hat{0}, \hat{1}, \ldots \)
    - but thinking of them as \( \forall, \exists, \bot, \top, \ldots \)
  - fibred algebraic effects
    - dep. typed **operation symbols** \( \text{op} : (x_v : I) \rightarrow O \)
    - ops. determine **computation terms** \( \text{op}_V^C(y : O[V/x_v]. M) \)
  - effect equations determine **definitional equations**

- a derivable “into-comps.” variant of **handlers and handling**
  \( M \) handled with \( (\{ \text{op}_{x_v}(x_k) \mapsto N_{\text{op}} \}_\text{op} \in \mathcal{S}_\text{eff}; \mathcal{W}_{\text{eq}}) \) to \( y : A \) in \( \mathcal{C} \) \( \mathcal{N}_{\text{ret}} \)

- a derivable “into-values” variant of **handlers and handling**
  \( M \) handled with \( (\{ \text{op}_{x_v}(x_k) \mapsto V_{\text{op}} \}_\text{op} \in \mathcal{S}_\text{eff}; \mathcal{W}_{\text{eq}}) \) to \( y : A \) in \( \mathcal{B} \) \( \mathcal{V}_{\text{ret}} \)
The calculus we propose in this paper . . .

• . . . is a variant of the FoSSaCS’16 calculus, with
  • a Tarski-style **value universe** $\mathcal{U}$
    • with **codes** written as $\hat{\Pi}, \hat{\Sigma}, \hat{0}, \hat{1}, \ldots$
    • but thinking of them as $\forall, \exists, \bot, \top, \ldots$
  • **fibred algebraic effects**
    • dep. typed **operation symbols** $\text{op} : (x_v : I) \rightarrow O$
    • ops. determine **computation terms** $\text{op}_V^C(y : O[V/x_v]. M)$
    • effect equations determine **definitional equations**

• a derivable “into-comps.” variant of handlers and handling $M$ handled with $(\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op}\in S_{\text{eff}}; W_{\text{eq}}})$ to $y : A \in_{\mathcal{C}} N_{\text{ret}}$

• a derivable “into-values” variant of handlers and handling $M$ handled with $(\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op}\in S_{\text{eff}}; W_{\text{eq}}})$ to $y : A \in_{\mathcal{B}} V_{\text{ret}}$
The calculus we propose in this paper . . .

• . . . is a variant of the FoSSaCS’16 calculus, with
  • a Tarski-style value universe $\mathcal{U}$
    • with codes written as $\hat{\Pi}, \hat{\Sigma}, \hat{0}, \hat{1}, \ldots$
    • but thinking of them as $\forall, \exists, \bot, \top, \ldots$
  • fibred algebraic effects
    • dep. typed operation symbols $\text{op} : (x_v : I) \rightarrow O$
    • ops. determine computation terms $\text{op}^{C_V}_V(y : O[V/x_v] \cdot M)$
    • effect equations determine definitional equations
  • a derivable “into-comps.” variant of handlers and handling
    $M$ handled with $(\{\text{op}_{x_v}(x_k) \mapsto N_{\text{op}}\}_{\text{op} \in S_{\text{eff}}; W_{\text{eq}}} \rightarrow y : A \in _C N_{\text{ret}}$
  • a derivable “into-values” variant of handlers and handling
    $M$ handled with $(\{\text{op}_{x_v}(x_k) \mapsto V_{\text{op}}\}_{\text{op} \in S_{\text{eff}}; W_{\text{eq}}} \rightarrow y : A \in _B V_{\text{ret}}$
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 (has issues)
  • Take 2: A new **type-level treatment** of handlers
Handlers are useful for extrinsic reasoning!

- An alternative to using prop. eq. on thunks for *preds. on* $M : FA$

- With handlers we define *predicates* $P : UFA \rightarrow U$ by
  1) equipping $U$ (or a resp. type) with an *algebra* structure
  2) *handling* the given computation using that algebra

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

- We discuss *three examples* of such predicates

- Also, an alternative to monadic reification for *rel. reasoning*

  - E.g., relating *stateful comps.* $M, N : FA$ as functions $S \rightarrow A \times S$
  - Not investigated in this paper
  - See [Grimm et al.'18] for *reification-based* relational reasoning
Handlers are useful for extrinsic reasoning!

- An alternative to using prop. eq. on thunks for preds. on $M : FA$
  - With handlers we define predicates $P : UFA \rightarrow \mathcal{U}$ by
    1) equipping $\mathcal{U}$ (or a resp. type) with an algebra structure
    2) handling the given computation using that algebra
- Intuitively, $P \ (\text{thunk } M)$ computes a proof obligation for $M$
- We discuss three examples of such predicates

- Also, an alternative to monadic reification for rel. reasoning
  - E.g., relating stateful comps. $M,N : FA$ as functions $S \rightarrow A \times S$
  - Not investigated in this paper
- See [Grimm et al.'18] for reification-based relational reasoning
Handlers are useful for extrinsic reasoning!

- An alternative to using prop. eq. on thunks for \textbf{preds. on } \textit{M : FA}
  - With handlers we define \textbf{predicates } \textit{P : UFA \rightarrow U} by
    1) equipping \textit{U} (or a resp. type) with an \textbf{algebra} structure
    2) \textbf{handling} the given computation using that algebra
  - Intuitively, \textit{P (thunk M)} computes a \textbf{proof obligation} for \textit{M}
  - We discuss \textbf{three examples} of such predicates

- Also, an alternative to monadic reification for \textbf{rel. reasoning}
  - E.g., relating \textbf{stateful comps. } \textit{M,N : FA} as \textbf{functions } \textit{S \rightarrow A \times S}
  - Not investigated in this paper
  - See [Grimm et al.’18] for \textbf{reification-based} relational reasoning
Handlers are useful for extrinsic reasoning!

- An alternative to using prop. eq. on thunks for **preds. on** \( M : FA \)
  - With handlers, we define **predicates** \( P : UFA \rightarrow U \) by
    1) equipping \( U \) (or a resp. type) with an **algebra** structure
    2) **handling** the given computation using that algebra
  - Intuitively, \( P (\text{thunk } M) \) computes a **proof obligation** for \( M \)
  - We discuss **three examples** of such predicates

- Also, an alternative to monadic reification for **rel. reasoning**
  - E.g., relating **stateful comps.** \( M, N : FA \) as functions \( S \rightarrow A \times S \)
  - Not investigated in this paper
  - See [Grimm et al.'18] for **reification-based** relational reasoning
Ex1: Lifting predicates to effectful comps.

- Given a predicate $P : A \to U$ on return values,

  we define a predicate $\square P : UFA \to U$ on (I/O)-comps. as

  \[
  \square P \overset{\text{def}}{=} \lambda y:UFA. (\text{force } y) \text{ handled with } \{ \ldots \}_{\text{op} \in S_{I/O}} \text{ to } y':A \text{ in } u \ P y'
  \]

  using the handler given by

  \[
  \begin{align*}
  \text{read} (x_k) & \mapsto \hat{\Pi} y:El(\hat{\text{Chr}}). x_k \ y \\
  \text{write}_{x_v} (x_k) & \mapsto x_k \ast
  \end{align*}
  \]

  (where $x_k:\text{Chr} \to U$)

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

  \[
  \Gamma \vdash \square P \left( \text{thunk} \left( \text{read}(x.\text{write}_{e'}(\text{return } V)) \right) \right) = \hat{\Pi} x:El(\hat{\text{Chr}}). P \ V
  \]

- To get $\Diamond P$, we only have to replace $\hat{\Pi}$ with $\hat{\Sigma}$ in the handler
Ex1: Lifting predicates to effectful comps.

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

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

  $$\square P \overset{\text{def}}{=} \lambda y : UFA. \text{handled with } \{ \ldots \}_\text{op} \in S_{I/O} \text{ to } y' : A \text{ in } \mathcal{U} \ P \ y'$$

  using the handler given by

  $$\begin{align*}
  \text{read}(x_k) & \mapsto \hat{\Pi} y : \text{El}(\hat{\text{Chr}}). x_k \ y \\
  \text{write}_{x_v}(x_k) & \mapsto x_k \ \ast
  \end{align*}$$

  (where $x_k : \text{Chr} \to \mathcal{U}$)

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

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

- To get $\Diamond P$, we only have to replace $\hat{\Pi}$ with $\hat{\Sigma}$ in the handler.
**Ex1: Lifting predicates to effectful comps.**

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

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

  $\Box P \overset{\text{def}}{=} \lambda y : \text{UFA}. \left( \text{force } y \right)$

  handled with \{ ... \}$_{\text{op} \in S_{\text{I/O}}}$ to $y' : A$ in \mathcal{U} $P y'$

  using the handler given by

  \begin{align*}
  \text{read}(x_k) & \mapsto \hat{\Pi} y : \text{El}(\hat{\text{Chr}}). x_k \ y \\
  \text{write}_{x_v}(x_k) & \mapsto x_k \ \ast \\
  \end{align*}

  (where $x_k : \text{Chr} \rightarrow \mathcal{U}$)

  (where $x_v : \text{Chr}, \ x_k : 1 \rightarrow \mathcal{U}$)

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

  $\Gamma \vdash \Box P \left( \text{thunk \left( \text{read}(x. \text{write}_v (\text{return } V)) \right) } \right) = \hat{\Pi} x : \text{El}(\hat{\text{Chr}}). P \ V$

- To get $\Diamond P$, we only have to replace $\hat{\Pi}$ with $\hat{\Sigma}$ in the handler
Ex1: Lifting predicates to effectful comps.

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

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

using the handler given by

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

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

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

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

- To get $\Diamond P$, we only have to replace $\hat{\Pi}$ with $\hat{\Sigma}$ in the handler
Ex2: Dijkstra’s weakest precondition sem.

- Given a postcondition on return values and final states
  \[ Q : A \rightarrow S \rightarrow U \]
  \( S \overset{\text{def}}{=} \prod \ell : \text{Loc}.\text{Val}(\ell) \)
  we define a precondition for stateful comps. on initial states
  \[ \wp_Q : UFA \rightarrow S \rightarrow U \]
  by
  1) handling the given comp. into a state-passing function using
     \[ V_{\text{get}}, V_{\text{put}} \text{ on } S \rightarrow U \times S \]
     and
     \[ V_{\text{ret}} \text{ “=}” Q \]
  2) feeding in the initial state; and
  3) projecting out the value of \( U \)

- Then, \( \wp_Q \) satisfies the expected properties, such as
  \[ \Gamma \vdash \wp_Q (\text{thunk}(\text{return } V)) = \lambda x_S : S . Q V x_S \]
  \[ \Gamma \vdash \wp_Q (\text{thunk}(\text{put}_{\langle \ell, V \rangle}(M))) = \lambda x_S : S . \wp_Q (\text{thunk } M) x_S[\ell \mapsto V] \]
Ex2: Dijkstra’s weakest precondition sem.

- Given a postcondition on return values and final states
  \[ Q : A \rightarrow S \rightarrow U \]
  
  we define a precondition for stateful comps. on initial states
  \[ \text{wp}_Q : UFA \rightarrow S \rightarrow U \]
  
  by
  
  1) handling the given comp. into a state-passing function using
     \[ V_{\text{get}}, V_{\text{put}} \text{ on } S \rightarrow U \times S \]
     and
     \[ V_{\text{ret}} \text{ "=} Q \]
  2) feeding in the initial state; and
  3) projecting out the value of \( U \)

- Then, \( \text{wp}_Q \) satisfies the expected properties, such as
  
  \[ \Gamma \vdash \text{wp}_Q (\text{thunk} (\text{return} V)) = \lambda x_S : S . Q \ V \ x_S \]
  
  \[ \Gamma \vdash \text{wp}_Q (\text{thunk} (\text{put} \langle \ell, V \rangle (M))) = \lambda x_S : S . \text{wp}_Q (\text{thunk} M) \ x_S[\ell \mapsto V] \]
Ex2: Dijkstra’s weakest precondition sem.

- Given a postcondition on return values and final states
  \[ Q : A \rightarrow S \rightarrow U \]
  \( S \overset{\text{def}}{=} \prod \ell : \text{Loc} . \text{Val}(\ell) \)

  we define a precondition for stateful comps. on initial states
  \[ \text{wp}_Q : UFA \rightarrow S \rightarrow U \]

  by
  1) handling the given comp. into a state-passing function using
  \[ V_{\text{get}}, V_{\text{put}} \text{ on } S \rightarrow U \times S \text{ and } V_{\text{ret}} \text{ “=}” Q \]
  2) feeding in the initial state; and
  3) projecting out the value of \( U \)

- Then, \( \text{wp}_Q \) satisfies the expected properties, such as
  \[ \Gamma \vdash \text{wp}_Q \left( \text{thunk} \left( \text{return } V \right) \right) = \lambda x_S : S . Q \left( V \right) x_S \]
  \[ \Gamma \vdash \text{wp}_Q \left( \text{thunk} \left( \text{put} \left( \ell, V \right)(M) \right) \right) = \lambda x_S : S . \text{wp}_Q \left( \text{thunk} M \right) x_S[\ell \mapsto V] \]
Ex2: Dijkstra’s weakest precondition sem.

- Given a postcondition on return values and final states

\[ Q : A \to S \to U \]

we define a precondition for stateful comps. on initial states

\[ wp_Q : UFA \to S \to U \]

by

1) handling the given comp. into a state-passing function using \( V_{get}, V_{put} \) on \( S \to U \times S \) and \( V_{ret} "=" Q \)

2) feeding in the initial state; and

3) projecting out the value of \( U \)

- Then, \( wp_Q \) satisfies the expected properties, such as

\[ \Gamma \vdash wp_Q (\text{thunk} (\text{return} V)) = \lambda x_S : S . Q V x_S \]

\[ \Gamma \vdash wp_Q (\text{thunk} (\text{put}_{\langle \ell, V \rangle}(M))) = \lambda x_S : S . wp_Q (\text{thunk} M) x_S[\ell \mapsto V] \]
Ex3: Allowed patterns of (I/O)-effects

- Assuming an inductive type of I/O-protocols, given by
  \[ \text{e : Protocol} \quad \text{r : (Chr \rightarrow Protocol)} \rightarrow \text{Protocol} \]
  \[ \text{w : (Chr \rightarrow U) \times Protocol} \rightarrow \text{Protocol} \]

- We can define a relation between comps. and protocols
  \[ \text{Allowed : UFA \rightarrow Protocol \rightarrow U} \]
  by handling the given computation using a handler on
  \[ \text{Protocol \rightarrow U} \]
given by (using pattern-matching lambda notation)

  \[ \text{read}(x_k) \mapsto \lambda \{(r \ x_{pr}) \rightarrow \hat{\Pi} \ y : \text{El(Chr)}. \ x_k \ y \ (x_{pr} \ y) ; \}
  \]
  \[- \rightarrow \hat{0} \} \]

  \[ \text{write}_{x_v}(x_k) \mapsto \lambda \{(w \ P \ x_{pr}) \rightarrow \hat{\Sigma} \ y : \text{El(P \ x_v)}. \ x_k \ * \ x_{pr} ; \}
  \]
  \[- \rightarrow \hat{0} \} \]
Ex3: Allowed patterns of (I/O)-effects

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

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

- We can define a relation between comps. and protocols

  \[ \text{Allowed} : UFA \rightarrow \text{Protocol} \rightarrow U \]

  by handling the given computation using a handler on

  \[ \text{Protocol} \rightarrow U \]

  given by (using pattern-matching lambda notation)

  \[ \text{read}(x_k) \mapsto \lambda \{ (r \ x_{pr}) \rightarrow \hat{\Pi} y : \text{El(Chr)}. x_k \ y \ (x_{pr} \ y) ; \}
  \[ - \rightarrow \hat{0} \} \]

  \[ \text{write}_{x_v}(x_k) \mapsto \lambda \{ (w \ P \ x_{pr}) \rightarrow \hat{\Sigma} y : \text{El(P x_v)}. x_k \ * \ x_{pr} ; \}
  \[ - \rightarrow \hat{0} \} \]}
Ex3: Allowed patterns of (I/O)-effects

- Assuming an inductive type of I/O-protocols, given by
  \[ e : \text{Protocol} \hspace{1cm} r : (\text{Chr} \rightarrow \text{Protocol}) \rightarrow \text{Protocol} \]
  \[ w : (\text{Chr} \rightarrow U) \times \text{Protocol} \rightarrow \text{Protocol} \]

- We can define a relation between comps. and protocols

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

by handling the given computation using a \textbf{handler} on

\[ \text{Protocol} \rightarrow U \]

given by (using pattern-matching lambda notation)

\[
\text{read}(x_k) \mapsto \lambda \{ (r \ x_{pr}) \rightarrow \hat{\Pi} \ y : \text{El}(\text{Chr}). x_k \ y \ (x_{pr} \ y) ;
- \rightarrow \hat{0} \}
\]

\[
\text{write}_{x_v}(x_k) \mapsto \lambda \{ (w \ P \ x_{pr}) \rightarrow \hat{\Sigma} \ y : \text{El}(P \ x_v). x_k \ * \ x_{pr} ;
- \rightarrow \hat{0} \}
\]
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 (has issues)
  - Take 2: A new type-level treatment of handlers
Extending the FoSSaCS’16 calculus

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

- First, we extend the calculus with algebraic effects as follows:
  - we extend the computation terms with
    $$M, N ::= \ldots \mid \text{op}_V^\mathcal{C}(y : O[V/x_v].M) \quad (\text{op} : (x_v : I) \rightarrow O \in S)$$
  - we extend the equational theory with equations given in $\mathcal{E}$
  - we capture the interaction of comp. terms and ops. with the eq.
    $$\Gamma \vdash V : I, \Gamma, x : O[V/x_v] \vdash M : \mathcal{C} \quad \Gamma, z : \mathcal{C} \vdash K : D \quad (\text{op} : (x_v : I) \rightarrow O \in S)$$
    $$\Gamma \vdash K[\text{op}_V^\mathcal{C}(x.M)/z] = \text{op}_V^\mathcal{D}(x.K[M/z]) : D$$

- Second, we extend the calculus with a support for handlers \ldots
Extending the FoSSaCS’16 calculus

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

- First, we extend the calculus with algebraic effects as follows:
  - we extend the computation terms with
    \[
    M, N ::= \ldots \mid \text{op}_V^C(y : O[V/x_v].M) \quad (\text{op} : (x_v : I) \rightarrow O \in S)
    \]
  - we extend the equational theory with equations given in \( \mathcal{E} \)
  - we capture the interaction of comp. terms and ops. with the eq.
    \[
    \frac{\Gamma \vdash V : I \quad \Gamma, x : O[V/x_v] \vdash M : C \quad \Gamma | z : C \vdash K : D}{\Gamma \vdash \text{op}_V^C(x.M)/z = \text{op}_V^D(x.K[M/z]) : D} \quad (\text{op} : (x_v : I) \rightarrow O \in S)
    \]

- Second, we extend the calculus with a support for handlers \ldots
We assume given a fibred effect theory $\mathcal{T} = (S, \mathcal{E})$.

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

- we extend the computation terms with

\[
M, N ::= \ldots \mid \text{op}_V^C(y : O[V/x_v] \cdot M) \quad \text{(op : (x_v : I) $\rightarrow$ O $\in$ S)}
\]

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

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

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

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

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

\[ M, N ::= \ldots \mid M \text{ handled with } \{ \text{op}_{x_k}(x_k) \mapsto N_{op} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \to y : A \text{ in } \mathcal{C} \subseteq N_{\text{ret}} \]

- But as handling denotes a homomorphism, then perhaps also

\[ K, L ::= \ldots \mid K \text{ handled with } \{ \text{op}_{x_k}(x_k) \mapsto N_{op} \}_{\text{op} \in \mathcal{S}_{\text{eff}}} \to y : A \text{ in } \mathcal{C} \subseteq N_{\text{ret}} \]

- However, this leads to an unsound calculus, e.g.,

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

- At a very high-level, the problem is (see the paper for details)
  - interaction between \( K \)s and ops. is governed by comp. types
  - but the type of handled with does not mention the handler
Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS’16 computation terms with

\[ M, N ::= \ldots \mid M \text{ handled with } \{ \text{op}_x(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C \subseteq N_{\text{ret}} \]

- But as handling denotes a homomorphism, then perhaps also

\[ K, L ::= \ldots \mid K \text{ handled with } \{ \text{op}_x(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C \subseteq N_{\text{ret}} \]

- However, this leads to an unsound calculus, e.g.,

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

- At a very high-level, the problem is (see the paper for details)
  - interaction between \( K \)s and ops. is governed by comp. types
  - but the type of handled with does not mention the handler
Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS’16 computation terms with
  \[
  M, N ::= \ldots \mid M \text{ handled with } \{ op_{x^v}(x_k) \mapsto N_{op} \}_{op \in S_{eff}} \text{ to } y : A \text{ in } C \subseteq N_{ret}
  \]

- But as handling denotes a homomorphism, then perhaps also
  \[
  K, L ::= \ldots \mid K \text{ handled with } \{ op_{x^v}(x_k) \mapsto N_{op} \}_{op \in S_{eff}} \text{ to } y : A \text{ in } C \subseteq N_{ret}
  \]

- However, this leads to an unsound calculus, e.g.,
  \[
  \Gamma \vdash \text{write}_a(\text{return} \star) = \text{write}_z(\text{return} \star) : F1
  \]

- At a very high-level, the problem is (see the paper for details)
  - interaction between \(K\)s and ops. is governed by comp. types
  - but the type of handled with does not mention the handler
Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS’16 computation terms with
  \[ M, N ::= \ldots \mid M \text{ handled with } \{ \text{op}_x(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C \subseteq N_{\text{ret}} \]

- But as handling denotes a **homomorphism**, then perhaps also
  \[ K, L ::= \ldots \mid K \text{ handled with } \{ \text{op}_x(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C \subseteq N_{\text{ret}} \]

- However, this leads to an **unsound** calculus, e.g.,
  \[ \Gamma \vdash \text{write}_{a}(\text{return} \star) = \text{write}_{z}(\text{return} \star) : F1 \]

- At a very high-level, the problem is (see the paper for details)
  - interaction between \( K \)'s and ops. is governed by comp. types
  - but the type of handled with does not mention the handler
Take 1: Term-level definition of handlers

- Begin by extending the FoSSaCS’16 computation terms with

  \[ M, N ::= \ldots \mid M \text{ handled with } \{ \text{op}_x(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C \subseteq N_{\text{ret}} \]

- But as handling denotes a homomorphism, then perhaps also

  \[ K, L ::= \ldots \mid K \text{ handled with } \{ \text{op}_x(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}} \text{ to } y : A \text{ in } C \subseteq N_{\text{ret}} \]

- However, this leads to an unsound calculus, e.g.,

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

- At a very high-level, the problem is (see the paper for details)
  
  - interaction between \( K \)'s and ops. is governed by comp. types
  
  - but the type of \text{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

\[ C, D ::= \ldots | \langle A; \overrightarrow{V_{op}}; \overrightarrow{W_{eq}} \rangle \]

where

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

• As a result, we can derive the "handing construct" as

\[ M \text{ handled with } \{ \{ \text{op}_x (x_k) \mapsto N_{op} \} \}_{\text{op} \in S_{\text{eff}}; \overrightarrow{W_{eq}}} \text{ to } y : A \text{ in } C \text{ N}_{\text{ret}} \]

\[ \text{def} = \]

\[ \text{force}_C (\text{thunk} (M \text{ to } y : A \text{ in force} \langle U_C; V_{N_{op}}; W_{eq} \rangle \text{ (thunk N}_{\text{ret}}))) \]

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

\[ C, D ::= \ldots \mid \langle A; \overrightarrow{V_{\text{op}}}; \overrightarrow{W_{\text{eq}}} \rangle \]

where

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

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

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

\[
\text{def} = \text{force}_C(\text{thunk}(M \text{ to } y : A \text{ in force } \langle UC; V_{\text{Noop}}; W_{\text{eq}} \rangle \text{ (thunk } N_{\text{ret}})))
\]

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

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

- As a result, we can derive the **handing construct** as
  $$M \text{ handled with } \{ \text{op}_{xv}(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in S_{\text{eff}}; W_{\text{eq}}} \text{ to } y : A \text{ in } C N_{\text{ret}}$$

  $$\overset{\text{def}}{=} \text{force}_C(\text{thunk}(M \text{ to } y : A \text{ in force } \langle UC; V_{\text{Nop}}; W_{\text{eq}} \rangle (\text{thunk } N_{\text{ret}})))$$

  temporarily working at type $\langle UC; V_{\text{Nop}}; W_{\text{eq}} \rangle$

  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

\[ C, D ::= \ldots \mid \langle A; V_{\text{op}}; W_{\text{eq}} \rangle \]

where
  - \( A \) is the carrier value type
  - \( V_{\text{op}} \) is a set of user-defined operations
  - \( W_{\text{eq}} \) is a set of witnesses of equational proof obligations

- As a result, we can derive the handling construct as

\[
M \text{ handled with } \{ \text{op}_{xv}(x_k) \mapsto N_{\text{op}} \}_{\text{op} \in \mathcal{S}_\text{eff}; W_{\text{eq}}} \text{ to } y : A \text{ in } C \text{ } N_{\text{ret}} \equiv \]

\[
\text{force}_C(\text{thunk}(M \text{ to } y : A \text{ in force } \langle UC; V_{N_{\text{op}}}; W_{\text{eq}} \rangle (\text{thunk } N_{\text{ret}})))
\]

(temporarily working at type \( \langle UC; V_{N_{\text{op}}}; W_{\text{eq}} \rangle \))

and similarly for the “into-values” variant of it
Conclusion

In conclusion

- handlers are natural for **extrinsic reasoning** about 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 $P : UFA \to U$
Thank you!

Questions?