An algebraic perspective on behavioral specifications in effectful languages

Danel Ahman
LFCS, University of Edinburgh

informatics

lfcs | Laboratory for Foundations of Computer Science

PhD work supervised by Gordon Plotkin and Alex Simpson

THE UNIVERSITY of EDINBURGH

Examples of reasoning about programs

- Side-effects - Hoare Logic, Separation Logic, Hoare Type Theory
- Network behavior - session types
- Permissions - ownership types
- Information flow security - security type systems
- ... - refinement types, contracts, dependent types

Examples of reasoning about programs

- Side-effects - Hoare Logic, Separation Logic, Hoare Type Theory - specific to state
- Network behavior - session types - specific to i/o
- Permissions - ownership types - specific to permissions
- Information flow security - security type systems
- ... - refinement types, contracts, dependent types - often specific to pure values or base types

Our long-term goals

- To develop a general theory of program specifications
  - accommodate various specification areas
    state side-effects, network behavior, permissions, ...
  - accommodate various notions of computation
    state, input/output, exceptions, probabilistic computation, handlers, ...
  - accommodate general logical specifications
- The tools we propose to use
  - Algebraic effects and their logics
  - Very fine-grained computational language (e.g., CBPV)
  - Refinement types (both on values and computations)

Computational effects
and
algebraic theories

Computational effects as monads

- Assume that we work in a cartesian-closed category $\mathbb{C}$
- Usual interpretation of λ-calculus
  - types as objects, terms as morphisms
- We model pure terms as morphisms $A \longrightarrow B$
- But how to model terms that can produce effects?
- Moggi's '91 answer to this question: use monads!
  - $T : [\mathbb{C}, \mathbb{C}]$
  - $\eta_X : X \longrightarrow TX$
  - $\mu_X : TTX \longrightarrow TX$
- or alternatively
  - $T : ob(\mathbb{C}) \to ob(\mathbb{C})$
  - $\eta_X : X \longrightarrow TX$
  - $f^* : TX \longrightarrow TY$ for every $f : X \longrightarrow TY$ in $\mathbb{C}$

Computational effects as monads

- Some of Moggi's example monads:
  - Global state: $TX = S \Rightarrow (S \times X)$
  - Exceptions: $TX = X + E$
  - Input/output: $TX = \mu Y.(V \Rightarrow Y) + (V \times Y) + X$
  - Non-determinism: $TX = \mathcal{P}(X)$
  - Continuations: $TX = (X \Rightarrow R) \Rightarrow R$
- But, once we have one such monad, what would be the right effectful program constructs to create elements of TX.
- And, once we have some effectful program constructs, what would be the right monad to model these effects?
- This is where algebraic effects will help us!

Computational effects as monads

- Some of Moggi's example monads:
  - Global state: $TX = S \Rightarrow (S \times X)$
  - Exceptions: $TX = X + E$
  - Input/output: $TX = \mu Y.(V \Rightarrow Y) + (V \times Y) + X$
  - Non-determinism: $TX = \mathcal{P}(X)$
  - Continuations: $TX = (X \Rightarrow R) \Rightarrow R$
- But, once we have one such monad, what would be the right effectful program constructs to create elements of TX.
- And, once we have some effectful program constructs, what would be the right monad to model these effects?
- This is where algebraic effects will help us!

Computational effects as monads

- Some of Moggi's example monads:
  - Global state: $TX = S \Rightarrow (S \times X)$
  - Exceptions: $TX = X + E$
  - Input/output: $TX = \mu Y.(V \Rightarrow Y) + (V \times Y) + X$
  - Non-determinism: $TX = \mathcal{P}(X)$
  - Continuations: $TX = (X \Rightarrow R) \Rightarrow R$
- But, once we have one such monad, what would be the right effectful program constructs to create elements of TX.
- And, once we have some effectful program constructs, what would be the right monad to model these effects?
- This is where algebraic effects will help us!

Algebraic effects

- A research programme started by Gordon Plotkin and John Power
- Key idea: use algebra to describe and represent effects!
- Use algebraic operations op : $\vec{\beta}; \vec{\alpha_1}, ..., \vec{\alpha_n}$ to present effects
- Use (conditional) equations to specify effectful behavior
- A model of effect theory $\mathbb{T}$ in a suitable category $\mathbb{C}$:
  - a carrier object X of $\mathbb{C}$
  - a morphism $[\![\mathrm{op}]\!]_X : [\![\beta]\!] \times ([\![\alpha]\!] \Rightarrow X) \longrightarrow X$ for every operation op : $\beta; \alpha$ (note: special case)
  - such that the equations are satisfied
- Such models form a category $\mathrm{Mod}(\mathbb{T}, \mathbb{C})$
- The monad $T = UF$ is induced by $F : \mathbb{C} \to \mathrm{Mod}(\mathbb{T}, \mathbb{C})$ and $U : \mathrm{Mod}(\mathbb{T}, \mathbb{C}) \to \mathbb{C}$ with $F \dashv U$

Algebraic effects

- A research programme started by Gordon Plotkin and John Power
- Key idea: use algebra to describe and represent effects!
- Use algebraic operations $\mathsf{op} : \vec{\beta}; \vec{\alpha_1}, \ldots, \vec{\alpha_n}$ to present effects
- Use (conditional) equations to specify effectful behavior
- A model of effect theory $\mathbb{T}$ in a suitable category $\mathbb{C}$:
  - a carrier object X of $\mathbb{C}$
  - a morphism $[\![\mathsf{op}]\!]_X : [\![\beta]\!] \times ([\![\alpha]\!] \Rightarrow X) \longrightarrow X$

    for every operation $\mathsf{op} : \beta; \alpha$ (note: special case)
  - such that the equations are satisfied
- Such models form a category $\mathsf{Mod}(\mathbb{T}, \mathbb{C})$
- The monad $T = UF$ is induced by $F : \mathbb{C} \to \mathsf{Mod}(\mathbb{T}, \mathbb{C})$ and $U : \mathsf{Mod}(\mathbb{T}, \mathbb{C}) \to \mathbb{C}$ with $F \dashv U$

Algebraic effects

- A research programme started by Gordon Plotkin and John Power
- Key idea: use algebra to describe and represent effects!

- Use algebraic operations op : $\vec{\beta}; \vec{\alpha_1}, \ldots, \vec{\alpha_n}$ to present effects
- Use (conditional) equations to specify effectful behavior

- A model of effect theory $\mathbb{T}$ in a suitable category $\mathbb{C}$:
  - a carrier object X of $\mathbb{C}$
  - a morphism $[\![\mathsf{op}]\!]_X : [\![\beta]\!] \times ([\![\alpha]\!] \Rightarrow X) \longrightarrow X$
    for every operation op : $\beta; \alpha$ (note: special case)
  - such that the equations are satisfied

- Such models form a category $\mathsf{Mod}(\mathbb{T}, \mathbb{C})$

- The monad $T = UF$ is induced by $F : \mathbb{C} \to \mathsf{Mod}(\mathbb{T}, \mathbb{C})$ and $U : \mathsf{Mod}(\mathbb{T}, \mathbb{C}) \to \mathbb{C}$ with $F \dashv U$

Algebraic effects examples: State

- Base types: val, loc
- Operations
  - lookup : loc; val
  - update : loc, val; 1
- Equations
  - $M \equiv \mathsf{lookup}_l((x : \mathsf{val}).\mathsf{update}_{l,x}(M))$
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_l((x : \mathsf{val}).M)) \equiv \mathsf{update}_{l,v}(M[v/x])$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l,v'}(M)) \equiv \mathsf{update}_{l,v'}(M)$
  - $\mathsf{lookup}_l((x : \mathsf{val}).(\mathsf{lookup}_{l'}((y : \mathsf{val}).M))) \equiv$
    $\mathsf{lookup}_{l'}((y : \mathsf{val}).(\mathsf{lookup}_l((x : \mathsf{val}).M))) \qquad (l \neq l')$
  - $\mathsf{update}_{l,v}(\mathsf{update}_{l',v'}(M)) \equiv \mathsf{update}_{l',v'}(\mathsf{update}_{l,v}(M)) \qquad (l \neq l')$
  - $\mathsf{update}_{l,v}(\mathsf{lookup}_{l'}((x : \mathsf{val}).M)) \equiv$
    $\mathsf{lookup}_{l'}((x : \mathsf{val}).(\mathsf{update}_{l,v}(M))) \qquad (l \neq l')$

Algebraic effects examples: Input/output

- Base types: $\mathsf{val}, \mathsf{chan}$
- Operations
  - $\mathsf{receive} : \mathsf{chan}; \mathsf{val}$
  - $\mathsf{send} : \mathsf{chan}, \mathsf{val}; 1$
- Equations
  - none!

Algebraic effects examples: Non-determinism

- Base types: none
- Operations
  - $\oplus : 1, 1; 1$
- Equations
  - $M \oplus M \equiv M$
  - $M \oplus N \equiv N \oplus M$
  - $(M \oplus N) \oplus P \equiv M \oplus (N \oplus P)$

Logic of algebraic effects (and CBPV)

- Algebraic effects give us straightforward means for equational reasoning about effects
- Plotkin and Pretnar developed a suitable multi-sorted predicate logic

- $A ::= \alpha \mid 1 \mid A_1 \times A_2 \mid 0 \mid A_1 + A_2 \mid U\underline{C}$
- $\underline{C} ::= FA \mid \underline{C}_1 \times \underline{C}_2 \mid A \to \underline{C}$

- $V ::= x \mid f(\vec{V}) \mid \star \mid \langle V_1, V_2 \rangle \mid \mathsf{proj}_i\, V \mid \mathsf{thunk}\, M \mid \ldots$
- $M ::= \zeta \mid \mathsf{return}\, V \mid M_1\, \mathsf{to}\, x.\, M_2 \mid \mathsf{op}_V(x.M) \mid \ldots$

- $\varphi ::= V_1 \equiv V_2 \mid M_1 \equiv M_2 \mid R(\vec{V}) \mid \pi(\vec{V}, \vec{M}) \mid \bot$
  $\mid \varphi_1 \vee \varphi_2 \mid \exists x : A.\varphi \mid \exists \zeta : \underline{C}.\varphi$
- $\pi ::= X \mid (\vec{x} : \vec{A}, \vec{\zeta} : \underline{\vec{C}}).\varphi \mid \mu X : (\vec{A}, \vec{C}).\pi \mid \nu X : (\vec{A}, \vec{C}).\pi$

Logic of algebraic effects: modalities

- Can define various useful ("temporal") modalities in this logic
- Pureness modalities:
  - $[\downarrow](\pi) \stackrel{\text{def}}{=} (\zeta : FA).\forall x : A.\zeta \equiv \mathsf{return}\, x \implies \pi(x)$
  - $\langle\downarrow\rangle(\pi) \stackrel{\text{def}}{=} (\zeta : FA).\exists x : A.\zeta \equiv \mathsf{return}\, x \wedge \pi(x)$
- Operation modalities:
  - $[\mathsf{op}](\pi) \stackrel{\text{def}}{=} (\zeta : \underline{C}).$
    $$\forall y : \beta, \zeta' : \alpha \to \underline{C}.\zeta \equiv \mathsf{op}_y((x : \alpha).\zeta'(x)) \implies \pi(y, \zeta')$$
  - $\langle\mathsf{op}\rangle(\pi) \stackrel{\text{def}}{=} (\zeta : \underline{C}).$
    $$\exists y : \beta, \zeta' : \alpha \to \underline{C}.\zeta \equiv \mathsf{op}_y((x : \alpha).\zeta'(x)) \wedge \pi(y, \zeta')$$
- Can extend $[\mathsf{op}]$ and $\langle\mathsf{op}\rangle$ to $[-]$ and $\langle - \rangle$ by taking conjunctions and disjunctions over all operations op
- Can use μ and ν to extend the local modalities $[-]$ and $\langle - \rangle$ to global modalities

Value and computation refinement types

The general picture

Value and computation refinement types

$\rightsquigarrow$ (upward)

CBPV

$+$

Algebraic effects

$+$

The logic of algebraic effects

Refinement types in general

- We use standard notation for refinement types: $\{x : \sigma \mid \varphi\}$
  - σ is the type we are refining
  - φ is the refinement proposition (i.e., logical specification)
  - x might appear free in φ

- As we use the CBPV paradigm, we can define two notions of refinement types:
  - Value refinement types: $\vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)$

    (these describe properties on values)

  - Computation refinement types: $\vdash \{\zeta : \underline{\tau} \mid \varphi\} : \underline{\mathsf{Ref}}(\underline{C})$

    (these describe properties of effectful behavior)

Value and computation refinement types

$$\frac{}{\vdash \alpha : \mathsf{Ref}(\alpha)} \qquad \frac{}{\vdash 1 : \mathsf{Ref}(1)} \qquad \frac{}{\vdash 0 : \mathsf{Ref}(0)}$$

$$\frac{\vdash \underline{\tau} : \underline{\mathsf{Ref}}(\underline{C})}{\vdash U\underline{\tau} : \mathsf{Ref}(U\underline{C})} \qquad \frac{\vdash \sigma : \mathsf{Ref}(A)}{\vdash F\sigma : \underline{\mathsf{Ref}}(FA)}$$

$$\frac{\vdash \sigma_i : \mathsf{Ref}(A_i) \quad (i \in \{1,2\})}{\vdash \sigma_1 + \sigma_2 : \mathsf{Ref}(A_1 + A_2)} \qquad \frac{\vdash \underline{\tau}_i : \underline{\mathsf{Ref}}(\underline{C}_i) \quad (i \in \{1,2\})}{\vdash \underline{\tau}_1 \times \underline{\tau}_2 : \underline{\mathsf{Ref}}(\underline{C}_1 \times \underline{C}_2)}$$

$$\frac{\vdash \sigma_1 : \mathsf{Ref}(A_1) \quad \vdash \sigma_2 : \mathsf{Ref}(A_2)}{\vdash \sigma_1 \times \sigma_2 : \mathsf{Ref}(A_1 \times A_2)} \qquad \frac{\vdash \sigma : \mathsf{Ref}(A) \quad \vdash \underline{\tau} : \underline{\mathsf{Ref}}(\underline{C})}{\vdash \sigma \to \underline{\tau} : \underline{\mathsf{Ref}}(A \to \underline{C})}$$

$$\frac{\vdash \sigma : \mathsf{Ref}(A) \quad x : A \vdash \varphi : \mathsf{prop}}{\vdash \{x : \sigma \mid \varphi\} : \mathsf{Ref}(A)} \qquad \frac{\vdash \underline{\tau} : \underline{\mathsf{Ref}}(\underline{C}) \quad \zeta : \underline{C} \vdash \varphi : \mathsf{prop}}{\vdash \{\zeta : \underline{\tau} \mid \varphi\} : \underline{\mathsf{Ref}}(\underline{C})}$$

Translations to underlying CBPV and the logic

- We can easily define a "forgetful" operation $\lceil - \rceil$
  - On value refinement types: $\lceil \vdash \sigma : \mathsf{Ref}(A) \rceil \overset{\mathsf{def}}{=} A$
  - On computation refinement types: $\lceil \vdash \underline{\tau} : \underline{\mathsf{Ref}}(\underline{C}) \rceil \overset{\mathsf{def}}{=} \underline{C}$
  - On terms, for example: $\lceil \mathsf{proj}_i\, V \rceil \overset{\mathsf{def}}{=} \mathsf{proj}_i\, \lceil V \rceil$

- There is also a translation $(-)^\bullet$ of refinement types to the logic of algebraic effects. For example:
  - $x : \lceil \{x' : \sigma \mid \varphi\} \rceil \vdash (\{x' : \sigma \mid \varphi\})^\bullet \quad \overset{\mathsf{def}}{=} \quad \varphi[x/x'] \wedge \sigma^\bullet$
  - $\zeta : \lceil F\sigma \rceil \vdash (F\sigma)^\bullet \quad \overset{\mathsf{def}}{=} \quad \Big(\mu X.((\zeta : \lceil F\sigma \rceil).$
    $(\exists x : \lceil \sigma \rceil . \zeta \equiv \mathsf{return}\, x \wedge \sigma^\bullet(x)) \vee (\langle\!\langle - \rangle\!\rangle(X)(\zeta)))\Big)(\zeta)$

Introduction and elimination of refinements

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \sigma \quad \lceil \Gamma \rceil \mid \Gamma^{\bullet} \vdash \varphi[\lceil V \rceil / x]}{\Gamma \vDash_{\mathsf{v}} V : \{x : \sigma \mid \varphi\}}$$

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \{x : \sigma \mid \varphi\}}{\Gamma \vDash_{\mathsf{v}} V : \sigma} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \{x : \sigma \mid \varphi\}}{\lceil \Gamma \rceil \mid \Gamma^{\bullet} \vdash \varphi[\lceil V \rceil / x]}$$

$$\frac{\Gamma \vDash_{\mathsf{c}} M : \underline{\tau} \quad \lceil \Gamma \rceil \mid \Gamma^{\bullet} \vdash \varphi[\lceil M \rceil / \zeta]}{\Gamma \vDash_{\mathsf{c}} M : \{\zeta : \underline{\tau} \mid \varphi\}}$$

$$\frac{\Gamma \vDash_{\mathsf{c}} M : \{\zeta : \underline{\tau} \mid \varphi\}}{\Gamma \vDash_{\mathsf{c}} M : \underline{\tau}} \qquad \frac{\Gamma \vDash_{\mathsf{c}} M : \{\zeta : \underline{\tau} \mid \varphi\}}{\lceil \Gamma \rceil \mid \Gamma^{\bullet} \vdash \varphi[\lceil M \rceil / \zeta]}$$

Some typing rules for value terms

- These typing rules resemble the standard CBPV rules

$$\frac{\vdash \Gamma, x : \sigma, \Gamma' \ \ \mathsf{wf}}{\Gamma, x : \sigma, \Gamma' \vDash_{\mathsf{v}} x : \sigma} \qquad \frac{\vdash \Gamma \ \ \mathsf{wf}}{\Gamma \vDash_{\mathsf{v}} \star : 1}$$

$$\frac{\Gamma \vDash_{\mathsf{v}} V_1 : \beta_1 \quad \ldots \quad \Gamma \vDash_{\mathsf{v}} V_n : \beta_n}{\Gamma \vDash_{\mathsf{v}} f(V_1, \ldots, V_n) : \beta} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : \sigma_1 \quad \Gamma \vDash_{\mathsf{v}} W : \sigma_2}{\Gamma \vDash_{\mathsf{v}} \langle V, W \rangle : \sigma_1 \times \sigma_2}$$

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \sigma_1 \times \sigma_2}{\Gamma \vDash_{\mathsf{v}} \mathsf{proj}_i\, V : \sigma_i} \qquad \frac{\Gamma \vDash_{\mathsf{c}} M : \underline{\tau}}{\Gamma \vDash_{\mathsf{v}} \mathsf{thunk}\, M : U\underline{\tau}}$$

Some typing rules for computation terms

- These typing rules resemble the standard CBPV rules

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \sigma}{\Gamma \vDash_{\mathsf{c}} \mathsf{return}\, V : F\sigma} \qquad \frac{\Gamma \vDash_{\mathsf{v}} V : U\underline{\tau}}{\Gamma \vDash_{\mathsf{c}} \mathsf{force}\, V : \underline{\tau}}$$

$$\frac{\Gamma, x : \sigma \vDash_{\mathsf{c}} M : \underline{\tau}}{\Gamma \vDash_{\mathsf{c}} \lambda x : \sigma . M : \sigma \to \underline{\tau}} \qquad \frac{\Gamma \vDash_{\mathsf{c}} M : \sigma \to \underline{\tau} \quad \Gamma \vDash_{\mathsf{v}} V : \sigma}{\Gamma \vDash_{\mathsf{c}} MV : \underline{\tau}}$$

$$\frac{\Gamma \vDash_{\mathsf{c}} M_1 : \underline{\tau}_1 \quad \Gamma \vDash_{\mathsf{c}} M_2 : \underline{\tau}_2}{\Gamma \vDash_{\mathsf{c}} \langle M_1, M_2 \rangle : \underline{\tau}_1 \times \underline{\tau}_2} \qquad \frac{\Gamma \vDash_{\mathsf{c}} M : \underline{\tau}_1 \times \underline{\tau}_2}{\Gamma \vDash_{\mathsf{c}} \mathsf{proj}_i\, M : \underline{\tau}_i}$$

Some typing rules for effect operations

- These typing rules differ from the standard CBPV rules

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \beta \quad \Gamma, x : \alpha \vDash_{\mathsf{c}} M : F\sigma}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).M) : F\sigma} \qquad \frac{\Gamma, y : \sigma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).(My)) : \underline{\tau}}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).M) : \sigma \to \underline{\tau}}$$

$$\frac{\Gamma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).(\mathsf{fst}\, M)) : \underline{\tau}_1 \quad \Gamma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).(\mathsf{snd}\, M)) : \underline{\tau}_2}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).M) : \underline{\tau}_1 \times \underline{\tau}_2}$$

- Compare these rules to the standard CBPV typing rule

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \beta \quad \Gamma, x : \alpha \vDash_{\mathsf{c}} M : \underline{C}}{\Gamma \vDash_{\mathsf{c}} \mathsf{op}_V((x : \alpha).M) : \underline{C}}$$

Some typing rules for sequencing

- These typing rules differ from the standard CBPV rules

$$\frac{\Gamma \vDash_c M : F\sigma_1 \quad \Gamma, x : \sigma_1 \vDash_c N : F\sigma_2}{\Gamma \vDash_c M \,\mathsf{to}\, x.\, N : F\sigma_2} \qquad \frac{\Gamma, y : \sigma \vDash_c M \,\mathsf{to}\, x.\, (N y) : \underline{\tau}}{\Gamma \vDash_c M \,\mathsf{to}\, x.\, N : \sigma \to \underline{\tau}}$$

$$\frac{\Gamma \vDash_c M \,\mathsf{to}\, x.\, (\mathsf{fst}\, N) : \underline{\tau}_1 \quad \Gamma \vDash_c M \,\mathsf{to}\, x.\, (\mathsf{snd}\, N) : \underline{\tau}_2}{\Gamma \vDash_c M \,\mathsf{to}\, x.\, N : \underline{\tau}_1 \times \underline{\tau}_2}$$

- Compare these rules to the standard CBPV typing rule

$$\frac{\Gamma \vDash_c M : FA \quad \Gamma, x : A \vDash_c N : \underline{C}}{\Gamma \vDash_c M \,\mathsf{to}\, x.\, N : \underline{C}}$$

Refinement relation and weakening

- We can give a straightforward structural definition for refinement relations

$$\Gamma \vDash_{\mathsf{r}} \sigma_2 \sqsubseteq \sigma_1 \qquad\qquad \Gamma \vDash_{\mathsf{r}} \underline{\tau}_2 \sqsubseteq \underline{\tau}_1$$

- And then show underlying type equality
  - $\Gamma \vDash_{\mathsf{r}} \sigma_2 \sqsubseteq \sigma_1 \implies \lceil \sigma_1 \rceil = \lceil \sigma_2 \rceil$
  - $\Gamma \vDash_{\mathsf{r}} \underline{\tau}_2 \sqsubseteq \underline{\tau}_1 \implies \lceil \underline{\tau}_1 \rceil = \lceil \underline{\tau}_2 \rceil$

- Using refinement relations, we also define weakening principles

$$\frac{\Gamma \vDash_{\mathsf{v}} V : \sigma_2 \quad \Gamma \vDash_{\mathsf{r}} \sigma_2 \sqsubseteq \sigma_1}{\Gamma \vDash_{\mathsf{v}} V : \sigma_1} \qquad \frac{\Gamma \vDash_{\mathsf{c}} M : \underline{\tau}_2 \quad \Gamma \vDash_{\mathsf{r}} \underline{\tau}_2 \sqsubseteq \underline{\tau}_1}{\Gamma \vDash_{\mathsf{c}} M : \underline{\tau}_1}$$

Some elements of our denotational semantics

Categorical semantics of CBPV and logic

- Semantics of CBPV:
  - Based on adjunction $F \dashv U$
  - We use a specific adjunction suitable for alg. effects:
    $F : \mathbf{Set} \to \mathsf{Mod}(\mathbb{T}, \mathbf{Set})$, $U : \mathsf{Mod}(\mathbb{T}, \mathbf{Set}) \to \mathbf{Set}$
  - Value types A are interpreted as objects $[\![A]\!]$ in $\mathbf{Set}$
  - Computation types $\underline{C}$ are interpreted as $[\![\underline{C}]\!]$ in $\mathsf{Mod}(\mathbb{T}, \mathbf{Set})$
  - Terms are interpreted as morphisms in $\mathbf{Set}$
    - $[\![\Gamma \vDash_{\mathsf{v}} V : A]\!] : [\![\Gamma]\!] \longrightarrow [\![A]\!]$
    - $[\![\Gamma \vDash_{\mathsf{c}} M : \underline{C}]\!] : [\![\Gamma]\!] \longrightarrow U[\![\underline{C}]\!]$
- Semantics of the logic of algebraic effects:
  - Propositions as: $[\![\Gamma \mid \Delta \mid \Theta \vdash \varphi]\!] \subseteq [\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\Theta]\!]$
  - Predicates as:
    $[\![\Gamma \mid \Delta \mid \Theta \vdash \pi]\!] : [\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\Theta]\!] \longrightarrow \mathcal{P}([\![\vec{A}]\!] \times U[\![\vec{\underline{C}}]\!])$
  - Sequents as: $\Gamma \mid \Delta \mid \Theta \mid \vec{\varphi} \vdash \varphi$ as $[\![\vec{\varphi}]\!] \subseteq [\![\varphi]\!]$

Categorical semantics of our type system

- Our current concrete categorical semantics is based on the $\mathbf{Set}$-based semantics of CBPV and the logic
- We abstract a little bit and work explicitly with the subobject fibration

$$
\begin{array}{ccc}
\mathrm{Sub}(\mathbf{Set}) & \qquad\qquad & \Gamma \mid \Delta \mid \Theta \vdash \varphi \\
\Big\downarrow{\scriptstyle p} & & \Big\downarrow{\scriptstyle p} \\
\mathbf{Set} & & \Gamma \mid \Delta \mid \Theta
\end{array}
$$

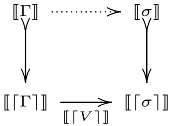
- Base category $\mathbf{Set}$ models contexts (and the language)
- Fibres $\mathrm{Sub}(\mathbf{Set})_{(\Gamma \mid \Delta \mid \Theta)}$ model the logic
- We give our semantics in the total category $\mathrm{Sub}(\mathbf{Set})$

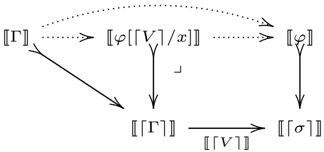
Categorical semantics of our type system ctd.

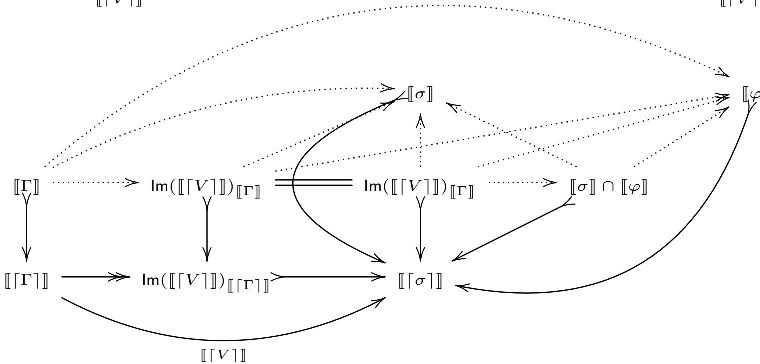
- So we start with $p : \mathsf{Sub}(\mathbf{Set}) \to \mathbf{Set}$
- Value refinement types $\vdash \sigma : \mathsf{Ref}(A)$ are interpreted as objects $[\![\sigma]\!] \rightarrowtail [\![A]\!]$ in $\mathsf{Sub}(\mathbf{Set})$
- Computation refinement types $\vdash \underline{\tau} : \underline{\mathsf{Ref}}(\underline{C})$ are interpreted as objects $[\![\underline{\tau}]\!] \rightarrowtail U[\![\underline{C}]\!]$ in $\mathsf{Sub}(\mathbf{Set})$
  - The interpretation of $\vdash F\sigma : \underline{\mathsf{Ref}}(FA)$ relies on the fact that all monads on $\mathbf{Set}$ preserve subobjects.
- Value and computation terms will be interpreted as morphisms in $\mathsf{Sub}(\mathbf{Set})$
  - Value terms $\Gamma \Vdash^{\mathsf{r}}_{\mathsf{v}} V : \sigma$ as morphisms from $[\![\Gamma]\!] \rightarrowtail [\![\lceil\Gamma\rceil]\!]$ to $[\![\sigma]\!] \rightarrowtail [\![\lceil\sigma\rceil]\!]$
  - Computation terms $\Gamma \Vdash^{\mathsf{r}}_{\mathsf{c}} M : \underline{\tau}$ as morphisms from $[\![\Gamma]\!] \rightarrowtail [\![\lceil\Gamma\rceil]\!]$ to $[\![\underline{\tau}]\!] \rightarrowtail U[\![\lceil\underline{\tau}\rceil]\!]$

Example: interpreting refinement introduction

$$\frac{\Gamma \mathrel{\Vdash\!\!\!\!\!\nabla} V : \sigma \qquad \lceil\Gamma\rceil \mid \Gamma^{\bullet} \vdash \varphi[\lceil V\rceil/x]}{\Gamma \mathrel{\Vdash\!\!\!\!\!\nabla} V : \{x : \sigma \mid \varphi\}}$$







Some examples of specifications

Pre- and post-condition specifications on state

- Based on the theory of state and follows ideas from:
  - Hoare triples $\{P\}\,\mathbf{C}\,\{Q\}$
  - Hoare types $\{P\}x : A\{Q\}$
- Pre- and post-conditions as computation refinement types:
  - $\vdash \{\zeta : FA \mid P \blacktriangleright_{x:A} Q\} : \underline{\mathsf{Ref}}(FA)$
- The Hoare refinement has the following definition

$$\zeta : FA \vdash P \blacktriangleright_{x:A} Q \stackrel{\text{def}}{=}$$

$$\forall x_{l_1} : \mathsf{int}, ..., x_{l_n} : \mathsf{int}, y_{l_1} : \mathsf{int}, ..., y_{l_n} : \mathsf{int}, z : A\,.$$

$$P^{\blacktriangleright}[x_{l_1}/x_1, ..., x_{l_n}/x_n] \wedge$$

$$\exists \zeta' : F1\,.\,\mathsf{update}_{l_1, x_{l_1}}(\quad ... \quad (\mathsf{update}_{l_n, x_{l_n}}(\zeta))) \equiv$$

$$\zeta' \,\mathsf{to}\, x.\,\mathsf{update}_{l_1, y_{l_1}}(\quad ... \quad (\mathsf{update}_{l_n, y_{l_n}}(\mathsf{return}\, z)))$$

$$\Longrightarrow \; Q^{\blacktriangleright}[y_{l_1}/x_1, ..., y_{l_n}/x_n, z/x]$$

Pre- and post-condition specifications on state

- Based on the theory of state and follows ideas from:
  - Hoare triples $\{P\}\,\mathbf{C}\,\{Q\}$
  - Hoare types $\{P\}x : A\{Q\}$
- Pre- and post-conditions as computation refinement types:
  - $\vdash \{\zeta : FA \mid P \blacktriangleright_{x:A} Q\} : \underline{\mathsf{Ref}}(FA)$
- The Hoare refinement has the following definition

$$\zeta : FA \vdash P \blacktriangleright_{x:A} Q \stackrel{\text{def}}{=}$$

$$\forall x_{l_1} : \mathsf{int}, \ldots, x_{l_n} : \mathsf{int}, y_{l_1} : \mathsf{int}, \ldots, y_{l_n} : \mathsf{int}, z : A\,.$$

$$P^{\blacktriangleright}[x_{l_1}/x_1, \ldots, x_{l_n}/x_n] \wedge$$

$$\exists \zeta' : F1\,.\,\mathsf{update}_{l_1,x_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n,x_{l_n}}(\zeta))) \equiv$$

$$\zeta'\,\mathsf{to}\,x.\,\mathsf{update}_{l_1,y_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n,y_{l_n}}(\mathsf{return}\,z)))$$

$$\Longrightarrow \; Q^{\blacktriangleright}[y_{l_1}/x_1, \ldots, y_{l_n}/x_n, z/x]$$

Pre- and post-condition specifications on state

- Based on the theory of state and follows ideas from:
  - Hoare triples $\{P\}\,\mathbf{C}\,\{Q\}$
  - Hoare types $\{P\}x : A\{Q\}$
- Pre- and post-conditions as computation refinement types:
  - $\vdash \{\zeta : FA \mid P \blacktriangleright_{x:A} Q\} : \underline{\mathsf{Ref}}(FA)$
- The Hoare refinement has the following definition

$$\zeta : FA \vdash P \blacktriangleright_{x:A} Q \stackrel{\text{def}}{=}$$

$$\forall x_{l_1} : \mathsf{int}, \ldots, x_{l_n} : \mathsf{int}, y_{l_1} : \mathsf{int}, \ldots, y_{l_n} : \mathsf{int}, z : A\,.$$

$$P^{\blacktriangleright}[x_{l_1}/x_1, \ldots, x_{l_n}/x_n] \wedge$$

$$\exists \zeta' : F1\,.\,\mathsf{update}_{l_1,x_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n,x_{l_n}}(\zeta))) \equiv$$

$$\zeta'\,\mathsf{to}\,x.\,\mathsf{update}_{l_1,y_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n,y_{l_n}}(\mathsf{return}\;z)))$$

$$\Longrightarrow \; Q^{\blacktriangleright}[y_{l_1}/x_1, \ldots, y_{l_n}/x_n, z/x]$$

Pre- and post-condition specifications on state

- Based on the theory of state and follows ideas from:
  - Hoare triples $\{P\}\,\mathbf{C}\,\{Q\}$
  - Hoare types $\{P\}x : A\{Q\}$
- Pre- and post-conditions as computation refinement types:
  - $\vdash \{\zeta : FA \mid P \blacktriangleright_{x:A} Q\} : \underline{\mathsf{Ref}}(FA)$
- The Hoare refinement has the following definition

$$\zeta : FA \vdash P \blacktriangleright_{x:A} Q \stackrel{\text{def}}{=}$$

$$\forall x_{l_1} : \mathsf{int}, \ldots, x_{l_n} : \mathsf{int}, y_{l_1} : \mathsf{int}, \ldots, y_{l_n} : \mathsf{int}, z : A\,.$$

$$P^{\blacktriangleright}[x_{l_1}/x_1, \ldots, x_{l_n}/x_n] \wedge$$

$$\exists \zeta' : F1\,.\, \mathsf{update}_{l_1, x_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n, x_{l_n}}(\zeta))) \equiv$$

$$\zeta' \,\mathsf{to}\, x.\, \mathsf{update}_{l_1, y_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n, y_{l_n}}(\mathsf{return}\; z)))$$

$$\Longrightarrow \; Q^{\blacktriangleright}[y_{l_1}/x_1, \ldots, y_{l_n}/x_n, z/x]$$

Pre- and post-condition specifications on state

- Based on the theory of state and follows ideas from:
  - Hoare triples $\{P\}\,\mathbf{C}\,\{Q\}$
  - Hoare types $\{P\}x : A\{Q\}$
- Pre- and post-conditions as computation refinement types:
  - $\vdash \{\zeta : FA \mid P \blacktriangleright_{x:A} Q\} : \underline{\mathsf{Ref}}(FA)$
- The Hoare refinement has the following definition

$$\zeta : FA \vdash P \blacktriangleright_{x:A} Q \stackrel{\text{def}}{=}$$

$$\forall x_{l_1} : \mathsf{int}, \ldots, x_{l_n} : \mathsf{int}, y_{l_1} : \mathsf{int}, \ldots, y_{l_n} : \mathsf{int}, z : A\,.$$

$$P^{\blacktriangleright}[x_{l_1}/x_1, \ldots, x_{l_n}/x_n] \wedge$$

$$\exists \zeta' : F1\,.\,\mathsf{update}_{l_1,x_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n,x_{l_n}}(\zeta))) \equiv$$

$$\zeta' \,\mathsf{to}\, x.\,\mathsf{update}_{l_1,y_{l_1}}(\quad \ldots \quad (\mathsf{update}_{l_n,y_{l_n}}(\mathsf{return}\, z)))$$

$$\implies Q^{\blacktriangleright}[y_{l_1}/x_1, \ldots, y_{l_n}/x_n, z/x]$$

Session refinements on network communication

- Based on the theory of I/O and follows ideas from:
  - Session types: $S ::= end \mid !A.S \mid ?A.S$
- We consider simple communication of bit-valued data over some fixed set of global channels $i \in \{a, b, c\}$
- We can define a grammar for session refinements
  - $S_i ::= end_i \mid !\mathsf{bit}.S_i \mid ?\mathsf{bit}.S_i$
  - $end_i \stackrel{\mathsf{def}}{=} \zeta.\neg(\Diamond_{(\zeta:\underline{T}).(\exists\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta')\vee\zeta\equiv\mathsf{send}_{i,1}(\zeta')))\vee(\exists\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta''))})(\zeta)$
  - $!\mathsf{bit}.S_i \stackrel{\mathsf{def}}{=} \zeta.\Box_{(\zeta:\underline{T}).\forall\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta') \implies S_i(\zeta')\wedge\zeta\equiv\mathsf{send}_{i,1}(\zeta') \implies S_i(\zeta'))\wedge}$
    $(\forall\zeta',\zeta''.\zeta \equiv \mathsf{receive}_i(\zeta',\zeta'') \implies \bot)(\zeta)$
  - $?\mathsf{bit}.S_i \stackrel{\mathsf{def}}{=} \zeta.\Box_{(\forall\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta'') \implies (S_i(\zeta')\wedge S_i(\zeta'')))\wedge}$
    $(\forall\zeta'.\zeta \equiv \mathsf{send}_{i,0}(\zeta') \implies \bot \wedge \zeta \equiv \mathsf{send}_{i,1}(\zeta') \implies \bot)(\zeta)$
- Unfortunately no true concurrency in algebraic effects yet
  - Plotkin's CONCUR'12 work on CCS-style parallel interleaving

Session refinements on network communication

- Based on the theory of I/O and follows ideas from:
  - Session types: $S ::= end \mid !A.S \mid ?A.S$
- We consider simple communication of bit-valued data over some fixed set of global channels $i \in \{a, b, c\}$
- We can define a grammar for session refinements
  - $S_i ::= end_i \mid !\mathsf{bit}.S_i \mid ?\mathsf{bit}.S_i$
  - $end_i \stackrel{\text{def}}{=} \zeta.\neg(\Diamond_{(\zeta:\mathcal{I})}.(\exists\zeta'.(\zeta{\equiv}\mathsf{send}_{i,0}(\zeta'){\vee}\zeta{\equiv}\mathsf{send}_{i,1}(\zeta'))){\vee}(\exists\zeta',\zeta''.\zeta{\equiv}\mathsf{receive}_i(\zeta',\zeta'')))(\zeta)$
  - $!\mathsf{bit}.S_i \stackrel{\text{def}}{=} \zeta.\Box_{(\zeta:\mathcal{I}).\forall\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta') \implies S_i(\zeta')\wedge\zeta\equiv\mathsf{send}_{i,1}(\zeta') \implies S_i(\zeta'))\wedge} (\forall\zeta',\zeta''.\zeta \equiv \mathsf{receive}_i(\zeta',\zeta'') \implies \bot)(\zeta)$
  - $?\mathsf{bit}.S_i \stackrel{\text{def}}{=} \zeta.\Box_{(\forall\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta'') \implies (S_i(\zeta')\wedge S_i(\zeta'')))\wedge} (\forall\zeta'.\zeta \equiv \mathsf{send}_{i,0}(\zeta') \implies \bot \wedge \zeta \equiv \mathsf{send}_{i,1}(\zeta') \implies \bot)(\zeta)$
- Unfortunately no true concurrency in algebraic effects yet
  - Plotkin's CONCUR'12 work on CCS-style parallel interleaving

Session refinements on network communication

- Based on the theory of I/O and follows ideas from:
  - Session types: $S ::= end \mid !A.S \mid ?A.S$
- We consider simple communication of bit-valued data over some fixed set of global channels $i \in \{a, b, c\}$
- We can define a grammar for session refinements
  - $S_i ::= end_i \mid !\text{bit}.S_i \mid ?\text{bit}.S_i$
  - $end_i \overset{\text{def}}{=} \zeta.\neg(\Diamond_{(\zeta:\mathcal{T}).(\exists\zeta'.(\zeta\equiv\text{send}_{i,0}(\zeta')\vee\zeta\equiv\text{send}_{i,1}(\zeta')))\vee(\exists\zeta',\zeta''.\zeta\equiv\text{receive}_i(\zeta',\zeta''))})(\zeta)$
  - $!\text{bit}.S_i \overset{\text{def}}{=} \zeta.\Box_{(\zeta:\mathcal{T}).\forall\zeta'.(\zeta\equiv\text{send}_{i,0}(\zeta') \implies S_i(\zeta')\wedge\zeta\equiv\text{send}_{i,1}(\zeta') \implies S_i(\zeta'))\wedge}$ $(\forall\zeta',\zeta''.\zeta \equiv \text{receive}_i(\zeta',\zeta'') \implies \bot)(\zeta)$
  - $?\text{bit}.S_i \overset{\text{def}}{=} \zeta.\Box_{(\forall\zeta',\zeta''.\zeta\equiv\text{receive}_i(\zeta',\zeta'') \implies (S_i(\zeta')\wedge S_i(\zeta'')))\wedge}$ $(\forall\zeta'.\zeta \equiv \text{send}_{i,0}(\zeta') \implies \bot \wedge \zeta \equiv \text{send}_{i,1}(\zeta') \implies \bot)(\zeta)$
- Unfortunately no true concurrency in algebraic effects yet
  - Plotkin's CONCUR'12 work on CCS-style parallel interleaving

Session refinements on network communication

- Based on the theory of I/O and follows ideas from:
  - Session types: $S ::= end \mid\ !A.S \mid\ ?A.S$
- We consider simple communication of bit-valued data over some fixed set of global channels $i \in \{a, b, c\}$
- We can define a grammar for session refinements
  - $S_i ::= end_i \mid\ !\mathsf{bit}.S_i \mid\ ?\mathsf{bit}.S_i$
  - $end_i \stackrel{\mathsf{def}}{=} \zeta.\neg(\Diamond_{(\zeta:\mathcal{I}).(\exists\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta')\vee\zeta\equiv\mathsf{send}_{i,1}(\zeta')))\vee(\exists\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta''))})(\zeta)$
  - $!\mathsf{bit}.S_i \stackrel{\mathsf{def}}{=} \zeta.\Box_{(\zeta:\mathcal{I}).\forall\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta') \implies S_i(\zeta')\wedge\zeta\equiv\mathsf{send}_{i,1}(\zeta') \implies S_i(\zeta'))\wedge}$ $(\forall\zeta',\zeta''.\zeta \equiv \mathsf{receive}_i(\zeta',\zeta'') \implies \bot)(\zeta)$
  - $?\mathsf{bit}.S_i \stackrel{\mathsf{def}}{=} \zeta.\Box_{(\forall\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta'') \implies (S_i(\zeta')\wedge S_i(\zeta'')))\wedge}$ $(\forall\zeta'.\zeta \equiv \mathsf{send}_{i,0}(\zeta') \implies \bot \wedge \zeta \equiv \mathsf{send}_{i,1}(\zeta') \implies \bot)(\zeta)$
- Unfortunately no true concurrency in algebraic effects yet
  - Plotkin's CONCUR'12 work on CCS-style parallel interleaving

Session refinements on network communication

- Based on the theory of I/O and follows ideas from:
  - Session types: $S ::= end \mid !A.S \mid ?A.S$
- We consider simple communication of bit-valued data over some fixed set of global channels $i \in \{a, b, c\}$
- We can define a grammar for session refinements
  - $S_i ::= end_i \mid !\mathsf{bit}.S_i \mid ?\mathsf{bit}.S_i$
  - $end_i \stackrel{\text{def}}{=} \zeta.\neg(\Diamond_{(\zeta:\mathcal{T})}.(\exists\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta')\vee\zeta\equiv\mathsf{send}_{i,1}(\zeta')))\vee(\exists\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta'')))(\zeta)$
  - $!\mathsf{bit}.S_i \stackrel{\text{def}}{=} \zeta.\Box_{(\zeta:\mathcal{T})}.\forall\zeta'.(\zeta\equiv\mathsf{send}_{i,0}(\zeta') \implies S_i(\zeta')\wedge\zeta\equiv\mathsf{send}_{i,1}(\zeta') \implies S_i(\zeta'))\wedge$ $(\forall\zeta',\zeta''.\zeta \equiv \mathsf{receive}_i(\zeta',\zeta'') \implies \bot)(\zeta)$
  - $?\mathsf{bit}.S_i \stackrel{\text{def}}{=} \zeta.\Box_{(\forall\zeta',\zeta''.\zeta\equiv\mathsf{receive}_i(\zeta',\zeta'') \implies (S_i(\zeta')\wedge S_i(\zeta'')))\wedge}$ $(\forall\zeta'.\zeta \equiv \mathsf{send}_{i,0}(\zeta') \implies \bot \wedge \zeta \equiv \mathsf{send}_{i,1}(\zeta') \implies \bot)(\zeta)$
- Unfortunately no true concurrency in algebraic effects yet
  - Plotkin's CONCUR'12 work on CCS-style parallel interleaving

Conclusions

- Showed our preliminary work towards combining the following:
  - algebraic effects
  - refinement types
  - program specifications
- Still a lot of work to be done
  - Generalizing the semantics of this simply-typed refinement type system to a context-dependent calculus
  - Accommodate local effects and instances of effects
  - Extend effect theories with built in behavioral refinements
  - Extend effect theories with cost measures and label algebras
  - Combinations of specifications induced by combinations of effect theories
  - Find more practical applications