

A Denotational Semantics for Polymorphic Effect Systems

Part III Project

Alexander Taylor, at736

University of Cambridge

May 1, 2019

Motivating Polymorphic Effect Analysis

```
def logAction(  
  action: Unit => String  
): Unit {  
  log.info(action())  
}
```

```
logAction(() => FireMissiles(); "Launched Missiles)
```

```
logAction(() => throwError("My Error"))
```

```
logAction(() => readEnvironmentVariables)
```

What is denotational Semantics?

- A mapping $\llbracket - \rrbracket : \text{Language Structure} \rightarrow \text{Mathematical Structure}$
- In particular want to define $\llbracket \Gamma \vdash t : A \rrbracket$
- Needs to be *compositional*, *sound*
- And *adequate* for our needs

Denotational Semantics using Category Theory

- Interested in: Objects, Morphisms, and Functors
- $\llbracket A \rrbracket, \llbracket \Gamma \rrbracket \in \text{obj } \mathbb{C}$
- $\llbracket \Gamma \vdash t : A \rrbracket : \llbracket \Gamma \rrbracket \rightarrow \llbracket A \rrbracket$

Language features (1) - Lambda Calculus

A cartesian closed category (CCC) consists of:

- Products $A \times B$ - models tuples
- A terminal object 1 - models the `Unit` type
- Exponential objects B^A - models functions as first-class objects

Language features (2.A) - Monads

A (strong) monad consists of:

- A functor $T : \mathbb{C} \rightarrow \mathbb{C}$
- Join and Unit natural transformations
 - ▶ $\mu_A : TTA \rightarrow TA$
 - ▶ $\eta_A : A \rightarrow TA$
- Tensor strength natural transformation $\mathfrak{t}_{A,B} : A \times TB \rightarrow T(A \times B)$

Language features (2.B) - Graded Monads

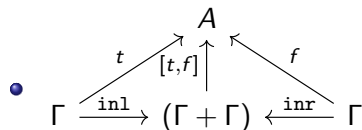
A (strong) graded monad consists of:

- An indexed functor $T_\epsilon : \mathbb{C} \rightarrow \mathbb{C}$
- Indexed Join and Unit natural transformations
 - ▶ $\mu_{\epsilon_1, \epsilon_2, A} : T_{\epsilon_1} T_{\epsilon_2} A \rightarrow T_{\epsilon_1 \cdot \epsilon_2} A$
 - ▶ $\eta_A : A \rightarrow T_1 A$
- Tensor strength natural transformation $\mathfrak{t}_{\epsilon, A, B} : A \times T_\epsilon B \rightarrow T_\epsilon(A \times B)$

Language Features (3.A) - If-Expressions

- If expression example - Co-product diagram

- if b then t else f
- If expressions modelled by co-products



Language Features 3.B - An Issue

Consider this:

```
if (UserConfirms) then Save() else pass;
```

- Branches have different effects
- So have different types!
- This doesn't type correctly

Language Features 3.C - Subtyping

$$\text{(Subtype)} \frac{\Gamma \vdash t: A \quad A \leq B}{\Gamma \vdash t: B}$$

- This needs a denotation
- So introduce $\llbracket A \leq B \rrbracket$
- $\llbracket \Gamma \vdash t: B \rrbracket = \llbracket A \leq B \rrbracket \circ \llbracket \Gamma \vdash t: A \rrbracket$

An Effectful Language

$$v ::= k^A \mid x \mid \text{true} \mid \text{false} \mid () \mid \lambda x: A. v \mid v_1 v_2 \mid \text{return } v \\ \mid \text{do } x \leftarrow v_1 \text{ in } v_2 \mid \text{if}_A v \text{ then } v_1 \text{ else } v_2$$
$$A, B, C ::= \gamma \mid A \rightarrow B \mid M_e A$$
$$\text{(Return)} \frac{\Gamma \vdash v: A}{\Gamma \vdash \text{return } v: M_1 A} \quad \text{(Apply)} \frac{\Gamma \vdash v_1: A \rightarrow B \quad \Gamma \vdash v_2: A}{\Gamma \vdash v_1 v_2: B}$$

Semantics of EC

- Can build a model of EC when we have
 - ▶ CCC
 - ▶ Co-product
 - ▶ Strong Graded Monad
 - ▶ Subtyping morphisms
- We'll call this an S-category

$$\text{(Return)} \frac{f = \llbracket \Gamma \vdash v : A \rrbracket}{\llbracket \Gamma \vdash \text{return } v : M_1 A \rrbracket = \eta_A \circ f}$$

$$\text{(Fn)} \frac{f = \llbracket \Gamma, x : A \vdash v : B \rrbracket : \Gamma \times A \rightarrow B}{\llbracket \Gamma \vdash \lambda x : A. v : A \rightarrow B \rrbracket = \text{cur}(f) : \Gamma \rightarrow B^A}$$

An Ugly Example

```
let twiceIO =  $\lambda$  action:  $M_{IO}Unit$ . (  
  do _ <- action in action  
)
```

```
let twiceState =  $\lambda$  action:  $M_{State}Unit$ . (  
  do _ <- action in action  
)
```

```
do _ <- twiceState(increment) in twiceIO(writeLog)
```

Let's Add Polymorphism

$$v ::= .. \mid \Lambda \alpha. v \mid v \epsilon$$

$$A, B, C ::= ... \mid \forall \alpha. A$$

$$\epsilon ::= e \mid \alpha \mid \epsilon \cdot \epsilon$$

$$\text{(Effect-Gen)} \frac{\Phi, \alpha \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \Lambda \alpha. v : \forall \alpha. A}$$

$$\text{(Effect-Spec)} \frac{\Phi \mid \Gamma \vdash v : \forall \alpha. A \quad \Phi \vdash \epsilon}{\Phi \mid \Gamma \vdash v \epsilon : A[\epsilon/\alpha]}$$

An Ugly Example - With a Makeover

```
let twice =  $\Lambda$  eff.(  
   $\lambda$  action:  $M_{eff}Unit$ . (  
    do _ <- action in action  
  )  
)  
  
do _ <- (twice State increment) in (twice IO writeLog)
```

How do we Model the Semantics of a Polymorphic Language?

- For a fixed effect variable environment Φ and terms with no polymorphic sub-terms, we have EC
- Effect-variable environments of length n are isomorphic by α -equivalence

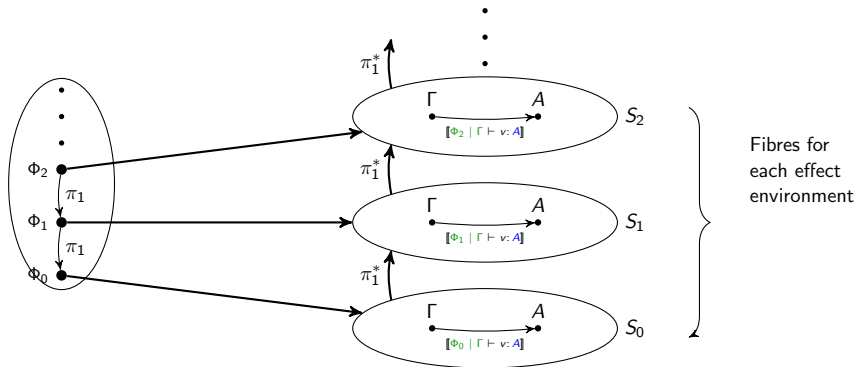
How do we Model the Semantics of a Polymorphic Language?

- Stack of S -categories and their morphisms
- type rule for generalisation - "Need functors"

Base Category

- We need a way of reasoning about effect-variable environment categorically
- We can model effects and environments in new category.
- Objects: $1, U, U^n$ (write I for U^n) - Morphisms: $\llbracket e \rrbracket : 1 \rightarrow U$ - Monoidal operator $\text{Mul} : \mathbb{C}(I, U) \times \mathbb{C}(I, U) \rightarrow \mathbb{C}(I, U)$ - Can represent each effect environment as an object I , and common transformations between environments, such as weakening and substitutions, are morphisms between effect environments.

Indexed Category



Quantification

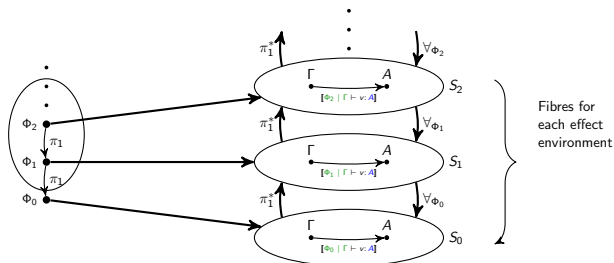
- What about effect-generalisation?

- (Effect-Gen)
$$\frac{\Phi, \alpha \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \Lambda \alpha. v : \forall \alpha. A}$$

- Need to map $\llbracket \Phi, \alpha \mid \Gamma \vdash v : A \rrbracket$ to $\llbracket \Phi \mid \Gamma \vdash \Lambda \alpha. v : \forall \alpha. A \rrbracket$

- For specialisation to work, needs: $\pi_1^* \dashv \forall_I$

Instantiating a Model (1)



- Can we actually instantiate a category with the required structure?
- Starting point a model of EC in Set

Instantiating a Model (2) - Base Category

- Use \mathbf{Eff} - category of monotone functions of tuples of ground effects to ground effects
- $\llbracket \diamond, \alpha, \beta \vdash \beta \cdot (\alpha \cdot \mathbf{IO}) : \mathbf{Effect} \rrbracket = (e_1, e_2) \mapsto e_2 \cdot (e_1 \cdot \mathbf{IO})$
- $\mathbf{Mul}(f, g)\vec{\epsilon} = (f\vec{\epsilon}) \cdot (g\vec{\epsilon})$

Instantiating a Model (3) - Fibres

- The fibre $\mathbb{C}(n)$ is the category of functors $[E^n, \text{Set}]$
- I.E. objects are functions that take a vector of ground effects and return a set.
- Morphisms are functions that return functions in Set
- S-Category features

Instantiating a Model (4) - Functors and Adjunctions

Re-indexing functors act by pre-composition

$$\begin{aligned} A &\in [E^n, \mathbf{Set}] \\ \theta^*(A)\epsilon_m^{\rightarrow} &= A(\theta(\epsilon_m^{\rightarrow})) \\ \theta^*(f)\epsilon_m^{\rightarrow} &= f(\theta(\epsilon_m^{\rightarrow})) : \theta^*(A) \rightarrow \theta^*(B) \end{aligned}$$

The quantification functor takes a product over all ground effects

$$\forall_{E^n}(A)\epsilon_n^{\rightarrow} = \prod_{\epsilon \in E} A(\epsilon_n^{\rightarrow}, \epsilon)$$

The End

- Dissertation and github links