# A Denotational Semantics for Polymorphic Effect Systems

## Part III Project

Alexander Taylor, at736

University of Cambridge

April 29, 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 $[\![ - ]\!]$ : Language Structure $\rightarrow$ Mathematical Structure
- In particular want to define $[\![\Gamma \vdash t \colon A]\!]$
- Needs to be *compositional*, *sound*
- And *adequate* for our needs

# Denotational Semantics using Category Theory

- Interested in: Objects, Morphisms, and Functors
- $[\![A]\!], [\![\Gamma]\!] \in \texttt{obj}\ \mathbb{C}$
- $[\![\Gamma \vdash t : A]\!] : [\![\Gamma]\!] \to [\![A]\!]$

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

# Language features (2.B) - Graded Monads

A (strong) graded monad consists of:

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

# Language Features (3.A) - If-Expressions

- If expression example - Co-product diagram

- $\mathtt{if}_A\ b\ \mathtt{then}\ t\ \mathtt{else}\ f$
- If expressions modelled by co-products
- $$\begin{tikzcd} & A & \\ \Gamma \arrow[ur, "t"] \arrow[r, "\mathtt{inl}"] & (\Gamma + \Gamma) \arrow[u, "{[t,f]}"] & \Gamma \arrow[ul, "f"'] \arrow[l, "\mathtt{inr}"'] \end{tikzcd}$$

# 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

$$\text{(Subtype)}\frac{\Gamma \vdash t \colon A \qquad A \leq\colon B}{\Gamma \vdash t \colon B}$$

- This needs a denotation
- So introduce $[\![A \leq\colon B]\!]$
- $[\![\Gamma \vdash t \colon B]\!] = [\![A \leq\colon B]\!] \circ [\![\Gamma \vdash t \colon A]\!]$

# An Effectful Language

$$v ::= \mathtt{C}^{A} \mid x \mid \mathtt{true} \mid \mathtt{false} \mid () \mid \lambda x\colon A.v \mid v_1\ v_2 \mid \mathtt{return}\ v$$
$$\mid \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 \mid \mathtt{if}_A\ v\ \mathtt{then}\ v_1\ \mathtt{else}\ v_2$$

$$A, B, C ::= \gamma \mid A \to B \mid \mathtt{M}_\epsilon A$$

$$(\text{Return})\frac{\Gamma \vdash v\colon A}{\Gamma \vdash \mathtt{return}\ v : \mathtt{M}_1 A} \qquad (\text{Apply})\frac{\Gamma \vdash v_1\colon A \to B \qquad \Gamma \vdash v_2\colon A}{\Gamma \vdash v_1\ v_2\colon B}$$

# 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

$$(\text{Return})\frac{f = [\![\Gamma \vdash v \colon A]\!]}{[\![\Gamma \vdash \texttt{return}\ v : \texttt{M}_1 A]\!] = \eta_A \circ f} \qquad (\text{Fn})\frac{f = [\![\Gamma, x \colon A \vdash v \colon B]\!] : \Gamma \times A \to B}{[\![\Gamma \vdash \lambda x \colon A.v : A \to B]\!] = \texttt{cur}(f) : \Gamma \to B^A}$$

# An Ugly Example

```
let twiceIO = λ action: M_IO Unit. (
    do _ <- action in action
)

let twiceState = λ action: M_State Unit. (
    do _ <- action in action
)

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

# Let's Add Polymorphism

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

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

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

$$(\text{Effect-Gen})\frac{\Phi, \alpha \mid \Gamma \vdash v \colon A}{\Phi \mid \Gamma \vdash \Lambda\alpha.v \colon \forall\alpha.A} \qquad (\text{Effect-Spec})\frac{\Phi \mid \Gamma \vdash v \colon \forall\alpha.A \qquad \Phi \vdash \epsilon}{\Phi \mid \Gamma \vdash v\ \epsilon \colon A[\epsilon/\alpha]}$$

# An Ugly Example - With a Makeover

```
let twice = Λ eff.(
    λ 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 $\Phi$ and terms with no polymorphic sub-terms, we have EC
- Effect-variable environments of length $n$ are isomorphic by $\alpha$-equivalence

# How do we Model the Semantics of a Polymorphic Language?

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

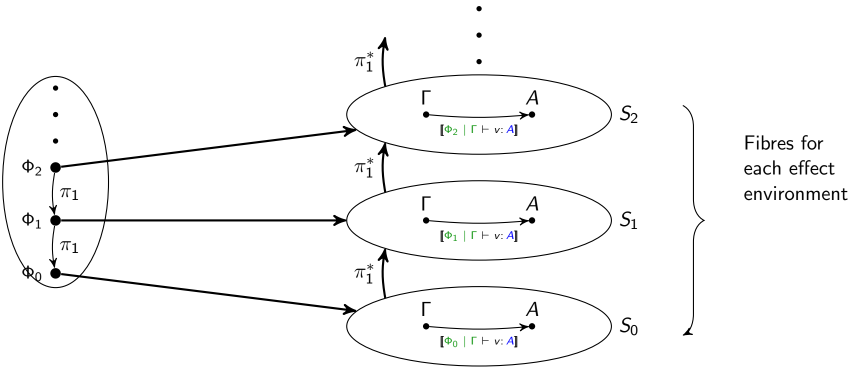
# Base Category

- We need a way of reasoning about effect-variable environment categorically

- We can model effects and environments in new category.

- Objects: $\mathtt{1}$, $U$, $U^n$ (write I for $U^n$) - Morphisms: $[\![e]\!] : \mathtt{1} \to U$ - Monoidal operator $\mathtt{Mul} : \mathbb{C}(I, U) \times \mathbb{C}(I, U) \to \mathbb{C}(I, U)$ - Can represent each effect environment as an object $I$, and common transformations between environments, such as weakening and substitutions, are morphisms between effect environments.

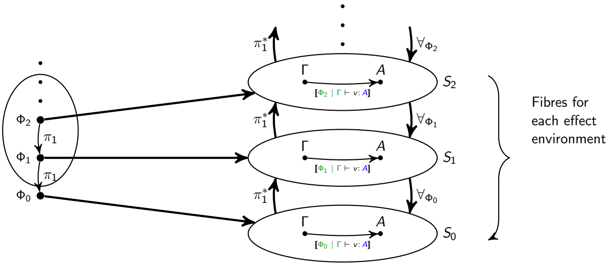
# Indexed Category



# Quantification

- What about effect-generalisation?
- (Effect-Gen) $\dfrac{\Phi, \alpha \mid \Gamma \vdash v \colon A}{\Phi \mid \Gamma \vdash \Lambda\alpha.v \colon \forall\alpha.A}$
- Need to map $[\![\Phi, \alpha \mid \Gamma \vdash v \colon A]\!]$ to $[\![\Phi \mid \Gamma \vdash \Lambda\alpha.v \colon \forall\alpha.A]\!]$
- 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 $\mathtt{Set}$

# Instantiating a Model (2) - Base Category

- Use `Eff` - category of monotone functions of tuples of ground effects to ground effects
- $[\![\diamond, \alpha, \beta \vdash \beta \cdot (\alpha \cdot \mathtt{IO})\colon \mathtt{Effect}]\!] = (e_1, e_2) \mapsto e_2 \cdot (e_1 \cdot \mathtt{IO})$
- $\mathtt{Mul}(f, g)\vec{\epsilon} = (f\vec{\epsilon}) \cdot (g\vec{\epsilon})$

# Instantiating a Model (3) - Fibres

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

# Instantiating a Model (4) - Functors and Adjunctions

Re-indexing functors act by pre-composition

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

The quantification functor takes a product over all ground effects

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

# The End

- Dissertation and github links