# A Denotational Semantics for Polymorphic Effect Systems

## Part III Project

Alexander Taylor, at736

University of Cambridge

May 15, 2019

# Motivating Polymorphic Effect Analysis

**TODO: Syntax highlight**

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

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

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

logAction(() => readEnvironmentVariables)
```

# What is a Denotational Semantics?

- A *compositional* mapping
  $[\![ - ]\!]$ : Language Structure $\rightarrow$ Mathematical Structure
- In particular want to define $[\![\Gamma \vdash t \colon A]\!]$
- Needs to be *sound* $t_1 \approx t_2 \implies [\![t_1]\!] = [\![t_2]\!]$
- And *adequate* for our needs $[\![t_1]\!] = [\![d_2]\!] \implies t_1 \approx t_2$

# Contributions

- A sound set of requirements for denotational semantics of effect-polymorphic languages.
- A method to construct models for effect-polymorphic languages in $\mathtt{Set}$.
- A proof of adequacy of such a model.

# Denotational Semantics using Category Theory

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

# Language features (1) - Lambda Calculus

A cartesian closed category (CCC) consists of:

- Products $A \times B$ - models tuples
- A terminal object $\mathtt{1}$ - 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 $\mathtt{t}_{\epsilon,A,B} : A \times T_\epsilon B \to T_\epsilon(A \times B)$

# An Effectful Language

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

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

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

# Semantics of EC

- Can build a model of EC when we have
  - CCC
  - Strong Graded Monad
  - Co-product and Subtyping (morphisms for if-statements)

- 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

**TODO: syntax highlight this**

let $\mathit{twiceIO} = \lambda\ \mathit{action}: \mathtt{M}_{\mathtt{IO}}\mathtt{Unit}.$ (
    do _ <- *action* in *action*
)

let $\mathit{twiceState} = \lambda\ \mathit{action}: \mathtt{M}_{\mathtt{State}}\mathtt{Unit}.$ (
    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

**TODO: Syntax highlighting**

let $\mathit{twice} = \Lambda\, \mathit{eff}.($
    $\lambda\, \mathit{action}$: $\texttt{M}_{\mathit{eff}}\texttt{Unit}$. (
        do _ <- *action* in *action*
    )
)

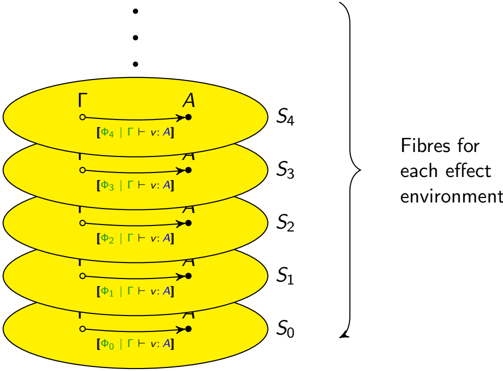
do _ <- (*twice* State increment) in (*twice* IO writeLog)

# 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?

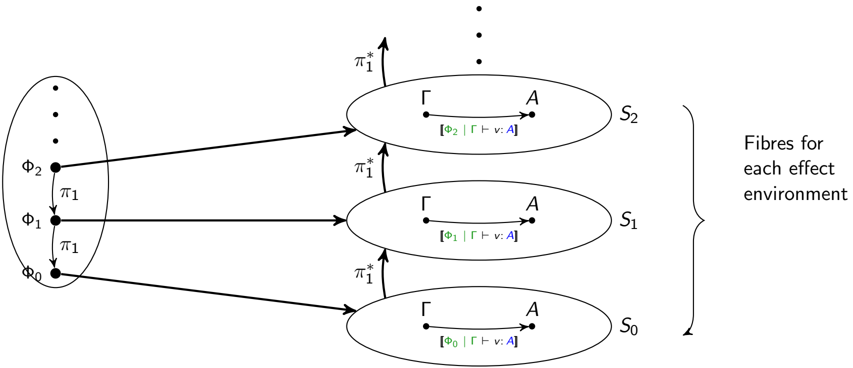
- So we instantiate an S-category for each environment.
- The type rule for quantification requires us to move between categories **TODO: Type rule here**.
- Functors are required.



# Base Category

- Need to be able to reason about the effect environment categorically
- Model effects and their environments using category `Eff`:
  - Objects are products of the set of ground effects $\{*\}, E, E \times E, \ldots E^n$..
  - Morphisms are monotone functions
- Represent $\Phi$ as an object $E^n$
- Transformations (e.g. substitutions) between environments become functions

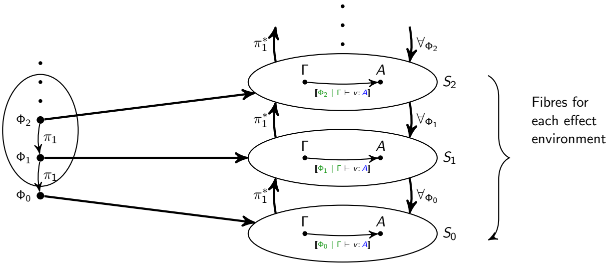
# Indexed Category



# 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) - Fibres

- The fibre $\mathbb{C}(n)$ is the category of functors $[E^n, \texttt{Set}]$

- I.E. objects are functions that take a vector of ground effects and return a set $[\![\Phi \vdash A\colon \texttt{Type}]\!] : E^n \to \texttt{obj}\ \ (\texttt{Set})$.

- Morphisms are dependent functions that return functions in $f : (\vec{\epsilon}\colon E^n) \to A\vec{\epsilon} \to B\vec{\epsilon}$

- These fibres have 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

- Sound: Proved for all indexed S-Categories ✓
- Compositional: By the definition of denotations ✓
- Adequate: Proved for an instantiation in $\mathtt{Set}$ ✓

**TODO: Dissertation and github links**