Abstract

To date, there has been limited work on the semantics of languages with polymorphic effect systems. The application, by Moggi, of strong monads to modelling the semantics of effects has become a mainstream concept in functional programming languages. This was improved upon by Lucasson (?) using a graded monad to model languages with a range of independent and dependent effects at an operational level. A categorical semantics for parametric polymorphism in types was first published by Reynolds (?) allowing a denotational analysis of languages including type parameters. There has been some work on polymorphism over the exception effect (which paper). Despite these works, there has been no work to date on the denotational semantics of languages with general parametric polymorphism over effects.

In this dissertation, I present several pieces of work. Firstly, I shall introduce a modern definition of a lambda-calculus based language with an explicit graded monad to handle a variety of effects. This calculus shall then be extended using polymorphic terms to yield a more general polymorphic-effect-calculus. I shall then give an indexed-category-based denotational semantics for the language, along with an outline of a proof for the soundness of these semantics. Following this, I shall present a method of transforming a model of a non-polymorphic language into a model of the language with polymorphism over effects.

The full proofs can be found online on my github repository (link), since due to the number of theorems and cases, the total size is well over 100 pages of definitions, theorems, and proofs.

Chapter 1

Introduction

1.1 What is Effect Polymorphism?

Effect polymorphism is when the same function in a language can operate on values of similar types but with different effects. It allows the same piece of code to be used in multiple contexts with different type signatures. This manifests in a similar manner to type parameter polymorphism in system-F based languages. Consider the following Scala-style pseudo-code:

```
def check[E: Effect](
    action: Unit => (Unit;e)
): Unit; (IO, e) {
    val ok: Boolean = promptBool(
        "Are you sure you want to do this?"
    )
    If (ok) {
        action()
    } else {
       abort()
    }
}
```

```
check[IO, RealWorld](() => check[RealWorld](FireMissiles))
check[Transaction](SendMoney(Bob, 100, USD))
check[Exception](ThrowException("Not Aborted"))
```

In this case, we are reusing the same “check” function in three different cases with three different effects in a type safe manner. “Check” is polymorphic in the effect parameter it receives. To analyse this language, it would be useful to have an analysis tool that can precisely model these separate, though potentially interdependent effects. A denotational semantics that can account for the parametric polymorphism over effects would be a step towards building such tools.

1.2 An Introduction to Categorical Semantics

A denotational semantics for a language is a mapping, known as a denotation, $[\![-]\!]_M$, of concepts in the language, such as types and terms to mathematical objects in such a way that properties of the terms in the language correspond to other properties of the denotations of the terms.

When we specify a denotational semantics of a language in category theory, we look to find a mapping of types and typing environments to objects in a given category.

$$A : \texttt{Type} \mapsto [\![A]\!]_M \in \texttt{obj}\ \mathbb{C} \tag{1.1}$$

$$\Gamma \mapsto [\![\Gamma]\!]_M \in \texttt{obj}\ \mathbb{C} \tag{1.2}$$

Further more, instances of the type relation should be mapped to morphisms between the relevant objects.

$$\Gamma \vdash v \colon A \mapsto \mathbb{C}([\![\Gamma]\!]_M, [\![A]\!]_M) \tag{1.3}$$

This should occur in a sound manner. That is, for every instance of the $\beta\eta$-equivalence relation between two terms, the denotations of the terms should be equal in the category.

$$\Gamma \vdash v_1 =_{\beta\eta} v_2 \colon A \implies [\![\Gamma \vdash v_1 \colon A]\!]_M = [\![\Gamma \vdash v_2 \colon A]\!]_M \tag{1.4}$$

To prove soundness, we induct over derivation of the $\beta\eta$-equivalence relation. In doing so, we need to prove a couple of properties which help prove cases. These are substitution and weakening. **TODO: What do these do?**

1.2.1 Languages and Their Requirements

One of the simplest, while still interesting, languages to derive a denotational semantics for is the simply typed lambda calculus (STLC). STLC’s semantics require a cartesian closed category (CCC, see section 2.1).

Products in the CCC are used to denote the lists of variable types in the typing environment, exponential objects model functions, and the terminal object is used to derive ”points” (**TODO: Terminology**) which represent the ground terms, such as the unit term, (), as well as the empty typing environment.

- Products are used to construct type environments. $[\![\Gamma]\!]_M = [\![\diamond, x : A, y : B, ...z : C]\!]_M = \mathtt{1} \times [\![A]\!]_M \times [\![B]\!]_M \times ... \times [\![C]\!]_M$
- Terminal objects are used in the denotation of constant terms $[\![\Gamma \vdash \mathtt{C}^A \colon A]\!]_M = [\![\mathtt{C}^A]\!]_M \circ \langle\rangle_{[\![\Gamma]\!]_M}$
- Exponentials are used in the denotations of functions. $[\![\Gamma \vdash \lambda x : A.v \colon A \to B]\!]_M = \mathtt{cur}([\![\Gamma, x : A \vdash v \colon B]\!]_M)$

From this, we can specify what structures categories need to have in order to model more complex languages.

Language Feature	Structure Required
STLC	CCC
Single Effect	Strong Monad
Multiple Effects	Strong Graded Monad
Polymorphism	Indexed Category

A single effect can be modelled by adding a strong monad to the category, as shown by Moggi **TODO: Reference**. The monad allows us to generate a unit effect and to compose multiple instances of the effect together in a way that intuitively matches the type system of a monadic language.

$$\text{(Return)}\frac{\Gamma \vdash v\colon A}{\Gamma \vdash \mathtt{return} v\colon \mathtt{M}A} \quad \text{(Bind)}\frac{\Gamma \vdash v_1 : \mathtt{M}A \quad \Gamma, x : A \vdash v_2 : \mathtt{M}B}{\Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}B} \tag{1.5}$$

These type rules can be modelled using the "unit" natural transformation and a combination of the "join" and tensor strength natural transformations respectively.

For more precise analysis of languages with multiple effects, we can look into the algebra on the effects. A simple example of such an algebra is a partially ordered monoid. The monoid operation defines how to compose effects, and the partial order gives a sub-typing relation to make programming more intuitive with respect to if statements. A category with a strong graded monad allows us to model this algebra in a category theoretic way. It also allows us to do some effect analysis in the type system, as seen in the type rules for return and bind in equation 1.6.

$$\text{(Return)}\frac{\Gamma \vdash v\colon A}{\Gamma \vdash \mathtt{return} v\colon \mathtt{M}_{\mathtt{1}} A} \quad \text{(Bind)}\frac{\Gamma \vdash v_1 : \mathtt{M}_{\epsilon_1} A \quad \Gamma, x : A \vdash v_2 : \mathtt{M}_{\epsilon_2} B}{\Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B} \tag{1.6}$$

To express polymorphism over a property P, the language's semantics are expanded to use a new environment specifying the variables ranging over P that are allowed in a given context. This can be seen in the augmented type rules in 1.7.

$$\text{(Gen)}\frac{\Phi, \alpha \mid v \vdash A\colon}{\Phi \mid \Gamma \vdash \Lambda\alpha.v\colon \forall\alpha.A} \quad \text{(Spec)}\frac{\Phi | \Gamma \vdash v : \forall\alpha.A \quad \Phi \vdash \epsilon}{\Phi | \Gamma \vdash v\ \epsilon : A[\epsilon/\alpha]} \tag{1.7}$$

To model these augmented type rules, we can create a category representing the semantics of the non-polymorphic language at each given context. This collection of non-polymorphic categories can be indexed by a base category which models the operations and relationships between the P-Environment. Morphisms in the base category between environments correspond to functors between the semantic categories for the relevant environments. These functors can then be used to construct the semantics of polymorphic terms. **TODO: Index diagram**

In this dissertation, I shall show how these category theoretic building blocks can be put together to give the class of categories that can model polymorphic effect systems.

Chapter 2

Required Category Theory

Before going further, it is assert a common level of category theory knowledge.

2.1 Cartesian Closed Category

Recall that a category is cartesian closed if it has a terminal object, products for all pairs of objects, and exponentials.

2.1.1 Terminal Object

An object, $\mathtt{1}$, is terminal in a category, $\mathbb{C}$ if for all objects $X \in \mathtt{obj}\ \mathbb{C}$, there exists exactly one morphism $\langle\rangle_X : X \to \mathtt{1}$.

2.1.2 Products

There is a product for a pair of objects $X, Y \in \mathtt{obj}\ \mathbb{C}$ if there exists an object and morphisms in C: $X \xleftarrow{\pi_1} (X \times Y) \xrightarrow{\pi_2} Y$

Such that for any other object and morphisms,

$X \xleftarrow{f} Z \xrightarrow{g} Y$

There exists a unique morphism $\langle f, g\rangle : Z \to (X \times Y)$ such that the following commutes:

$$\begin{tikzcd}
 & Z \arrow[dl, "f"'] \arrow[d, "\langle f,g\rangle"] \arrow[dr, "g"] & \\
X & (X \times Y) \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & Y
\end{tikzcd}$$

2.1.3 Exponentials

A category has exponentials if for all objects A, B, it has an object B^A and a morphism $\mathtt{app} : \mathtt{Bool}^A \times A \to B$ and for each $f : (A \times B) \to C$ in $\mathbb{C}$ there exists a unique morphism $\mathtt{cur}(f) : A \to C^B$ such that the following diagram commutes.

$$\begin{tikzcd}C^B \times B \arrow[r, "\mathtt{app}"] & C \\ A \times B \arrow[u, "\mathtt{cur}(f)\times\mathtt{Id}_B"] \arrow[ur, "f"] & \end{tikzcd}$$

2.2 Co-Product

2.3 Functors

A functor $F : \mathbb{C} \to \mathbb{D}$ is a mapping of objects:

$$A \in \mathtt{obj}\ \mathbb{C} \mapsto FA \in \mathtt{obj}\ \mathbb{D} \tag{2.1}$$

And morphisms:

$$f : \mathbb{C}(A, B) \mapsto F(f) : \mathbb{D}(FA, FB) \tag{2.2}$$

that preserves the category properties of composition and identity.

$$F(\mathtt{Id}_A) = \mathtt{Id}_{FA} \tag{2.3}$$

$$F(g \circ f) = F(g) \circ F(f) \tag{2.4}$$

2.4 Natural Transformations

A natural transformation θ between to functors $F, G : \mathbb{C} \to \mathbb{D}$ is a collection of morphisms, indexed by objects in $\mathbb{C}$ with $\theta_A : F(A) \to G(A)$ such that following diagram commutes for each $f : A \to B \in \mathbb{C}$

$$\begin{tikzcd}F(A) \arrow[r, "\theta_A"] \arrow[d, "F(f)"] & G(A) \arrow[d, "G(f)"] \\ F(B) \arrow[r, "\theta_B"] & G(B)\end{tikzcd}$$

2.5 Monad

A monad is famously "a monoid on the category of endofunctors". In less opaque terms, a monad is:

- A functor from $\mathbb{C}$ onto itself. (An endofunctor) $T : \mathbb{C} \to C$
- A "unit" natural transformation $\eta_A : A \to T(A)$
- A "join" natural transformation $\mu_A : T(T(A)) \to T(A)$

Such that the following diagrams commute:

Associativity

$$\begin{tikzcd}
T(T(T(A)) \arrow[r, "\mu_{T(A)}"] \arrow[d, "T(\mu_A)"] & T(T(A)) \arrow[d, "\mu_A"] \\
T(T(A)) \arrow[r, "\mu_A"] & T(A)
\end{tikzcd}$$

Left and Right Unit

$$\begin{tikzcd}
T(A) \arrow[r, "\eta_{T(A)}"] \arrow[d, "T(\eta_A)"] \arrow[dr, "todo"] & T(T(A)) \arrow[d, "\mu_A"] \\
T(T(A)) \arrow[r, "\mu_A"] & T(A)
\end{tikzcd}$$

2.6 Graded Monad

A graded monad is a generalisation of a monad to be indexed by a monoidal algebra E. It is made up of:

- An endo-functor indexed by a monoid: $T : (\mathbb{E}, \cdot\, \mathtt{1}) \to [\mathbb{C}, \mathbb{C}]$
- A unit natural transformation: $\eta : \mathtt{Id} \to T_{\mathtt{1}}$
- A join natural transformation: $\mu_{\epsilon_1, \epsilon_2,} : T_{\epsilon_1} T_{\epsilon_2} \to T_{\epsilon_1 \cdot \epsilon_2}$

Such that the following diagrams commute.:

2.6.1 Left Unit

$$\begin{tikzcd}
T_\epsilon A \arrow[r, "T_\epsilon \eta_A"] \arrow[dr, "\mathtt{Id}_{T_\epsilon A}"] & T_\epsilon T_{\mathtt{1}} A \arrow[d, "\mu_{\epsilon, \mathtt{1}, A}"] \\
& T_\epsilon A
\end{tikzcd}$$

2.6.2 Right Unit

$$\begin{tikzcd}
T_\epsilon A \arrow[r, "\eta_{T_\epsilon A}"] \arrow[dr, "\mathtt{Id}_{T_\epsilon A}"] & T_{\mathtt{1}} T_{\mathtt{1}} A \arrow[d, "\mu_{\mathtt{1}, \epsilon, A}"] \\
& T_\epsilon A
\end{tikzcd}$$

2.6.3 Associativity

$$\begin{tikzcd}
T_{\epsilon_1} T_{\epsilon_2} T_{\epsilon_3} A \arrow[r, "\mu_{\epsilon_1, \epsilon_2, T_{\epsilon_3} A}"] \arrow[d, "T_{\epsilon_1} \mu_{\epsilon_2, \epsilon_3, A}"] & T_{\epsilon_1 \cdot \epsilon_2} T_{\epsilon_3} A \arrow[d, "\mu_{\epsilon_1 \cdot \epsilon_2, \epsilon_3, A}"] \\
T_{\epsilon_1} T_{\epsilon_2 \cdot \epsilon_3} A \arrow[r, "\mu_{\epsilon_1, \epsilon_2 \cdot \epsilon_3, A}"] & T_{\epsilon_1 \cdot \epsilon_2 \cdot \epsilon_3} A
\end{tikzcd}$$

2.7 Tensor Strength

Tensorial strength over a graded monad gives us the tools necessary to manipulate monadic operations in an intuitive way. Tensorial strength consists of a natural transformation:

$$\mathtt{t}_{\epsilon,A,B} : A \times T_\epsilon B \to T_\epsilon(A \times B) \tag{2.5}$$

Such that the following diagrams commute:

2.7.1 Left Naturality

$$\begin{tikzcd}
A \times T_\epsilon B \arrow[r, "\mathtt{Id}_A \times T_\epsilon f"] \arrow[d, "\mathtt{t}_{\epsilon,A,B}"] & A \times T_\epsilon B' \arrow[d, "\mathtt{t}_{\epsilon,A,B'}"] \\
T_\epsilon(A \times B) \arrow[r, "T_\epsilon(\mathtt{Id}_A \times f)"] & T_\epsilon(A \times B')
\end{tikzcd}$$

2.7.2 Right Naturality

$$\begin{tikzcd}
A \times T_\epsilon B \arrow[r, "f \times \mathtt{Id}_{T_\epsilon B}"] \arrow[d, "\mathtt{t}_{\epsilon,A,B}"] & A' \times T_\epsilon B \arrow[d, "\mathtt{t}_{\epsilon,A',B}"] \\
T_\epsilon(A \times B) \arrow[r, "T_\epsilon(f \times \mathtt{Id}_B)"] & T_\epsilon(A' \times B)
\end{tikzcd}$$

2.7.3 Unitor Law

$$\begin{tikzcd}
\mathtt{1} \times T_\epsilon A \arrow[r, "\mathtt{t}_{\epsilon,\mathtt{1},A}"] \arrow[dr, "\lambda_{T_\epsilon A}"] & T_\epsilon(\mathtt{1} \times A) \arrow[d, "T_\epsilon(\lambda_A)"] \\
& T_\epsilon A
\end{tikzcd}$$

Where $\lambda : \mathtt{1} \times \mathtt{Id} \to \mathtt{Id}$ is the left-unitor. $(\lambda = \pi_2)$

Tensor Strength and Projection Due to the left-unitor law, we can develop a new law for the commutativity of π_2 with $\mathtt{t}_{,,}$

$$\pi_{2,A,B} = \pi_{2,1,B} \circ (\langle\rangle_A \times \mathtt{Id}_B)$$

And $\pi_{2,1}$ is the left unitor, so by tensorial strength:

$$\begin{aligned}
T_\epsilon \pi_2 \circ \mathtt{t}_{\epsilon,A,B} &= T_\epsilon \pi_{2,1,B} \circ T_\epsilon(\langle\rangle_A \times \mathtt{Id}_B) \circ \mathtt{t}_{\epsilon,A,B} \\
&= T_\epsilon \pi_{2,1,B} \circ \mathtt{t}_{\epsilon,\mathtt{1},B} \circ (\langle\rangle_A \times \mathtt{Id}_B) \\
&= \pi_{2,1,B} \circ (\langle\rangle_A \times \mathtt{Id}_B) \\
&= \pi_2
\end{aligned} \tag{2.6}$$

So the following commutes:

$$\begin{tikzcd}
A \times T_\epsilon B \arrow[r, "\mathtt{t}_{\epsilon,A,B}"] \arrow[dr, "\pi_2"] & T_\epsilon(A \times B) \arrow[d, "T_\epsilon \pi_2"] \\
& T_\epsilon B
\end{tikzcd}$$

2.7.4 Commutativity with Join

$$\begin{tikzcd}
A \times T_{\epsilon_1} T_{\epsilon_2} B \arrow[r, "\mathtt{t}_{\epsilon_1, A, T_{\epsilon_2} B}"] \arrow[dr, "\mathtt{Id}_A \times \mu_{\epsilon_1, \epsilon_2, B}"] & T_{\epsilon_1}(A \times T_{\epsilon_2} B) \arrow[r, "T_{\epsilon_1} \mathtt{t}_{\epsilon_2, A, B}"] & T_{\epsilon_1} T_{\epsilon_2}(A \times B) \arrow[d, "\mu_{\epsilon_1, \epsilon_2, A \times B}"] \\
 & A \times T_{\epsilon_1 \cdot \epsilon_2} B \arrow[r, "\mathtt{t}_{\epsilon_1 \cdot \epsilon_2, A, B}"] & T_{\epsilon_1 \cdot \epsilon_2}(A \times B)
\end{tikzcd}$$

2.8 Commutativity with Unit

$$\begin{tikzcd}
A \times B \arrow[r, "\mathtt{Id}_A \times \eta_B"] \arrow[dr, "\eta_{A \times B}"] & A \times T_{\mathtt{1}} B \arrow[d, "\mathtt{t}_{\mathtt{1}, A, B}"] \\
 & T_{\mathtt{1}}(A \times B)
\end{tikzcd}$$

2.9 Commutativity with α

Let $\alpha_{A,B,C} = \langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle : ((A \times B) \times C) \to (A \times (B \times C))$

$$\begin{tikzcd}
(A \times B) \times T_\epsilon C \arrow[rr, "\mathtt{t}_{\epsilon, (A \times B), C}"] \arrow[d, "\alpha_{A, B, T_\epsilon C}"] & & T_\epsilon((A \times B) \times C) \arrow[d, "T_\epsilon \alpha_{A, B, C}"] \\
A \times (B \times T_\epsilon C) \arrow[r, "\mathtt{Id}_A \times \mathtt{t}_{\epsilon, B, C}"] & A \times T_\epsilon (B \times C) \arrow[r, "\mathtt{t}_{\epsilon, A, (B \times C)}"] & T_\epsilon(A \times (B \times C))
\end{tikzcd}$$

2.10 Adjunction

An important concept in category theory is that of an Adjunction.

Given functors F, G:

$$\begin{tikzcd}
C \arrow[r, bend right, "F"] & D \arrow[l, bend right, "G"]
\end{tikzcd}$$

And natural transformations:

- Unit: $\eta_A : A \to G(FA)$ in $\mathbb{C}$
- Co-unit $\epsilon_B : F(GB) \to B$ in $\mathbb{D}$

Such that

$$\epsilon_{FA} \circ F(\eta_A) = \mathtt{Id}_{FA} \tag{2.7}$$

$$G(\epsilon_B) \circ \eta_{FB} = \mathtt{Id}_{GB} \tag{2.8}$$

We can then use ϵ and η to form a natural isomorphism between morphisms in the two categories.

$$\overline{(-)} : \quad \mathbb{C}(FA, B) \leftrightarrow \mathbb{D}(A, GB) \quad : \widehat{(-)} \tag{2.9}$$

$$f \mapsto G(f) \circ \eta_A \tag{2.10}$$

$$\epsilon \circ F(g) \leftarrow\!\shortmid g \tag{2.11}$$

$$\tag{2.12}$$

2.11 Strict Indexed Category

The final piece of category theory required to understand this dissertation is the concept of a strictly indexed Category.

A strict indexed category is a functor from a base category into a target category of categories, such as the category of cartesian closed categories.

Objects in the base category are mapped to categories in the target category. Morphisms between objects in the base category are mapped to functors between categories in the target category.

For example, we may use the the case of cartesian closed categories indexed by a pre-order:

$$I : \mathbb{P} \to CCCat \quad \text{The indexing functor} \tag{2.13}$$

$$A \in \texttt{obj}\ \mathbb{P} \mapsto \mathbb{C} \in CCCat \quad \text{Objects are mapped to categories} \tag{2.14}$$

$$A \leq B \mapsto (A \leq B)^* : \mathbb{C} \to \mathbb{D} \quad \text{Morphisms are mapped to functors preserving CCC properties.} \tag{2.15}$$