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

In this example, we are reusing the same "check" function in three different cases with three different

effects in a type safe manner. Hence, "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

In this dissertation, I shall be describing a denotational semantics using category theory. A denotational semantics for a language is a mapping, known as a denotation, $[\![-]\!]_M$, of structures in the language, such as types and terms to mathematical objects in such a way that non-trivial 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.

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

An example of $\beta\eta$-equivalence is that of the β-reduction of lambda terms. It should be the case that:

$$\text{(Lambda-Beta)}\frac{\Gamma, x : A \vdash v_1\colon B \quad \Gamma \vdash v_2\colon A}{\Gamma \vdash (\lambda x : A.v_1)\ v_2 =_{\beta\eta} v_1\,[v_2/x]\,\colon B} \tag{1.5}$$

Means that the denotations $[\![\Gamma \vdash (\lambda x : A.v_1)\ v_2\colon B]\!]_M$ and $[\![\Gamma \vdash v_1\,[v_2/x]\,\colon B]\!]_M$ are equal.

To prove soundness, we perform rule induction over the derivation of the $\beta\eta$-equivalence relation, such as the lambda-beta-reduction rule above.

Some of the inductive cases require us to quantify what the substitution of terms for variables, such as $[v_1/x]$ does to the denotations of the parent term. Substitution theorems, allow us to quantify this action on denotations in a category theoretic way.

If

$$\Gamma, x : A \vdash v_1\colon B \tag{1.6}$$

And

$$\Gamma \vdash v_2\colon A \tag{1.7}$$

Then

$$[\![\Gamma \vdash v_1\,[v_2/x]\,\colon B]\!]_M \tag{1.8}$$

Should be derivable from

$$[\![\Gamma, x : A \vdash v_1 : B]\!]_M \tag{1.9}$$

And

$$[\![\Gamma \vdash v_2 : A]\!]_M \tag{1.10}$$

A similar concept is that of environment weakening. $\Gamma, x : A$ can derive every typing relation that Γ can, if x is not already in the environment Γ. Hence, $\Gamma, x : A$ is an example of a typing environment that is *weaker* than Γ. A weakening theorem proves that there is a systematic way to generate the denotation of a typing-relation on a term in a weaker environment from the denotation of the same term in a stronger environment.

If

$$\Gamma' \leq_{\text{weaker}} \Gamma \tag{1.11}$$

Then

$$[\![\Gamma' \vdash v : A]\!]_M \tag{1.12}$$

should be derivable from

$$[\![\Gamma \vdash v : A]\!]_M \tag{1.13}$$

1.2.1 Languages and Their Requirements

Different languages require different structures to be present in a category for the category to be able to interpret terms in the language. Using the concepts defined in 2, I shall now give an introduction to which category-theoretic structures are required to interpret different language features.

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 representations of 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 $[\![\Gamma \vdash \mathtt{C}^A : A]\!]_M = [\![\mathtt{C}^A]\!]_M \circ \langle\rangle_{[\![\Gamma]\!]_M}$
- Exponentials are used in the denotations of functions. $[\![\Gamma \vdash \lambda x : A.v : A \to B]\!]_M = \mathtt{cur}([\![\Gamma, x : A \vdash v : B]\!]_M)$

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

Language Feature	Structure Required
STLC	CCC
If expressions and booleans	Co-product on the terminal object
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.

$$(\text{Return})\frac{\Gamma \vdash v\colon A}{\Gamma \vdash \texttt{return} v\colon \mathtt{M}A} \quad (\text{Bind})\frac{\Gamma\vdash v_1\colon \mathtt{M}A \quad \Gamma, x\colon A \vdash v_2\colon \mathtt{M}B}{\Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2\colon \mathtt{M}B} \tag{1.14}$$

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

For a 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 1.15.

$$(\text{Return})\frac{\Gamma \vdash v\colon A}{\Gamma \vdash \texttt{return} v\colon \mathtt{M}_{\mathtt{1}}A} \quad (\text{Bind})\frac{\Gamma\vdash v_1\colon \mathtt{M}_{\epsilon_1}A \quad \Gamma, x\colon A \vdash v_2\colon \mathtt{M}_{\epsilon_2}B}{\Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2\colon \mathtt{M}_{\epsilon_1\cdot\epsilon_2}B} \tag{1.15}$$

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 1.16.

$$(\text{Gen})\frac{\Phi, \alpha \mid \Gamma \vdash v\colon A}{\Phi \mid \Gamma \vdash \Lambda\alpha.v\colon \forall\alpha.A} \quad (\text{Spec})\frac{\Phi \mid \Gamma \vdash v\colon \forall\alpha.A \quad \Phi \vdash \epsilon}{\Phi \mid \Gamma \vdash v\ \epsilon\colon A[\epsilon/\alpha]} \tag{1.16}$$

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 necessary to assert a common level of category theory knowledge. This section is not intended as a tutorial but as a to jog the memory of the reader, and briefly introduce some new concepts.

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 $\langle\rangle_X : X \to \mathtt{1}$.

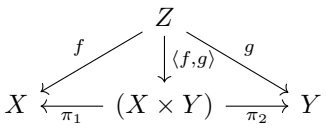
2.1.2 Products

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,

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

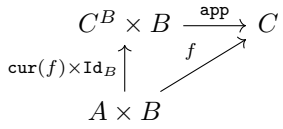
There exists a unique morphism $\langle f, g\rangle : Z \to (X \times Y)$ such that the following commutes:



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.



2.2 Initial Object

An initial object, I of $\mathbb{C}$ is one such that for every other object $X \in \mathtt{obj}\ \mathbb{C}$, there exists a unique morphism $\iota_X : I \to X$. It is the conceptual dual of a terminal objects.

2.3 Co-Product

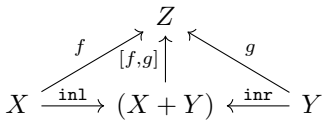
A co-product is the conceptual dual of a product.

There is a co-product for a pair of objects $X, Y \in \mathtt{obj}\ \mathbb{C}$ if there exists an object and morphisms in C: $X \xrightarrow[\mathtt{inl}]{} (X + Y) \xleftarrow[inr]{} Y$

Such that for any other object and morphisms,

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

There exists a unique morphism $[f, g] : X + Y \to Z$ such that the following commutes:



2.4 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.

$$F(\mathtt{Id}_A) = \mathtt{Id}_{FA} \tag{2.3}$$
$$F(g \circ f) = F(g) \circ F(f) \tag{2.4}$$

2.5 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 $f : A \to B \in \mathbb{C}$

$$\begin{array}{ccc} F(A) & \xrightarrow{\theta_A} & G(A) \\ \downarrow{\scriptstyle F(f)} & & \downarrow{\scriptstyle G(f)} \\ F(B) & \xrightarrow{\theta_B} & G(B) \end{array}$$

2.6 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:

2.6.1 Associativity

$$\begin{array}{ccc} T(T(T(A)) & \xrightarrow{\mu_{T(A)}} & T(T(A)) \\ \downarrow{\scriptstyle T(\mu_A)} & & \downarrow{\scriptstyle \mu_A} \\ T(T(A)) & \xrightarrow{\mu_A} & T(A) \end{array}$$

2.6.2 Left and Right Unit

$$\begin{array}{ccc} T(A) & \xrightarrow{\eta_{T(A)}} & T(T(A)) \\ \downarrow{\scriptstyle T(\eta_A)} & \searrow\!\!\searrow & \downarrow{\scriptstyle \mu_A} \\ T(T(A)) & \xrightarrow{\mu_A} & T(A) \end{array}$$

2.7 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.:

2.7.1 Left and Right Units

$$\begin{array}{ccc} T_\epsilon A & \xrightarrow{T_\epsilon \eta_A} & T_\epsilon T_{\mathtt{1}} A \\ \downarrow{\scriptstyle \eta_{T_\epsilon A}} & \searrow & \downarrow{\scriptstyle \mu_{\epsilon,\mathtt{1},A}} \\ T_{\mathtt{1}} T_\epsilon A & \xrightarrow{\mu_{\mathtt{1},\epsilon,A}} & T_\epsilon A \end{array}$$

2.7.2 Associativity

$$\begin{array}{ccc} T_{\epsilon_1} T_{\epsilon_2} T_{\epsilon_3} A & \xrightarrow{\mu_{\epsilon_1,\epsilon_2,T_{\epsilon_3}A}} & T_{\epsilon_1 \cdot \epsilon_2} T_{\epsilon_3} A \\ \downarrow{\scriptstyle T_{\epsilon_1}\mu_{\epsilon_2,\epsilon_3,A}} & & \downarrow{\scriptstyle \mu_{\epsilon_1\cdot\epsilon_2,\epsilon_3,A}} \\ T_{\epsilon_1} T_{\epsilon_2\cdot\epsilon_3} A & \xrightarrow{\mu_{\epsilon_1,\epsilon_2\cdot\epsilon_3,A}} & T_{\epsilon_1\cdot\epsilon_2\cdot\epsilon_3} A \end{array}$$

2.8 Tensor Strength

Tensorial strength over a graded monad gives us the tools necessary to manipulate monadic operations in an intuitive way. A monad with tensor strength is referred to as "strong". 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:

2.8.1 Left Naturality

$$\begin{array}{ccc} A \times T_\epsilon B & \xrightarrow{\mathtt{Id}_A \times T_\epsilon f} & A \times T_\epsilon B' \\ \downarrow{\scriptstyle \mathtt{t}_{\epsilon,A,B}} & & \downarrow{\scriptstyle \mathtt{t}_{\epsilon,A,B'}} \\ T_\epsilon (A \times B) & \xrightarrow{T_\epsilon(\mathtt{Id}_A \times f)} & T_\epsilon (A \times B') \end{array}$$

2.8.2 Right Naturality

$$\begin{array}{ccc} A \times T_\epsilon B & \xrightarrow{f \times \mathtt{Id}_{T_\epsilon B}} & A' \times T_\epsilon B \\ \downarrow{\scriptstyle \mathtt{t}_{\epsilon,A,B}} & & \downarrow{\scriptstyle \mathtt{t}_{\epsilon,A',B}} \\ T_\epsilon (A \times B) & \xrightarrow{T_\epsilon(f \times \mathtt{Id}_B)} & T_\epsilon (A' \times B) \end{array}$$

2.8.3 Unitor Law

$$\begin{array}{ccc} \mathtt{1} \times T_\epsilon A & \xrightarrow{\mathtt{t}_{\epsilon,\mathtt{1},A}} & T_\epsilon(\mathtt{1} \times A) \\ & \searrow^{\lambda_{T_\epsilon A}} & \downarrow{\scriptstyle T_\epsilon(\lambda_A)} \\ & & T_\epsilon A \end{array}$$

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:

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

So the following commutes:

$$\begin{array}{ccc} A \times T_\epsilon B & \xrightarrow{\mathtt{t}_{\epsilon,A,B}} & T_\epsilon(A \times B) \\ & \searrow^{\pi_2} & \downarrow{\scriptstyle T_\epsilon \pi_2} \\ & & T_\epsilon B \end{array}$$

2.8.4 Commutativity with Join

$$\begin{array}{ccccc} A \times T_{\epsilon_1} T_{\epsilon_2} B & \xrightarrow{\mathtt{t}_{\epsilon_1,A,T_{\epsilon_2}B}} & T_{\epsilon_1}(A \times T_{\epsilon_2} B) & \xrightarrow{T_{\epsilon_1}\mathtt{t}_{\epsilon_2,A,B}} & T_{\epsilon_1} T_{\epsilon_2}(A \times B) \\ & \searrow^{\mathtt{Id}_A \times \mu_{\epsilon_1,\epsilon_2,B}} & & & \downarrow{\scriptstyle \mu_{\epsilon_1,\epsilon_2,A\times B}} \\ & & A \times T_{\epsilon_1 \cdot \epsilon_2} B & \xrightarrow{\mathtt{t}_{\epsilon_1\cdot\epsilon_2,A,B}} & T_{\epsilon_1\cdot\epsilon_2}(A \times B) \end{array}$$

2.9 Commutativity with Unit

$$\begin{array}{ccc} A \times B & \xrightarrow{\mathtt{Id}_A \times \eta_B} & A \times T_{\mathbf{1}} B \\ & \searrow^{\eta_{A\times B}} & \downarrow{\scriptstyle \mathtt{t}_{\mathbf{1},A,B}} \\ & & T_{\mathbf{1}}(A \times B) \end{array}$$

2.10 Commutativity with α

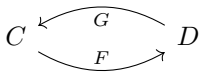
Let $\alpha_{A,B,C} = \langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle : ((A \times B) \times C) \to (A \times (B \times C))$

$$\begin{array}{ccccc} (A \times B) \times T_\epsilon C & & \xrightarrow{\qquad\mathtt{t}_{\epsilon,(A\times B),C}\qquad} & & T_\epsilon((A \times B) \times C) \\ \downarrow{\scriptstyle \alpha_{A,B,T_\epsilon C}} & & & & \downarrow{\scriptstyle T_\epsilon \alpha_{A,B,C}} \\ A \times (B \times T_\epsilon C) & \xrightarrow{\mathtt{Id}_A \times \mathtt{t}_{\epsilon,B,C}} & A \times T_\epsilon(B \times C) & \xrightarrow{\mathtt{t}_{\epsilon,A,(B\times C)}} & T_\epsilon(A \times (B \times C)) \end{array}$$

2.11 Adjunction

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

Given functors F, G:



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.

$$\overline{(-)} : \quad \mathbb{C}(FA, B) \leftrightarrow \mathbb{D}(A, GB) \quad : \widehat{(-)} \tag{2.9}$$
$$f \mapsto G(f) \circ \eta_A \tag{2.10}$$
$$\epsilon \circ F(g) \hookleftarrow g \tag{2.11}$$
$$\tag{2.12}$$

2.12 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.

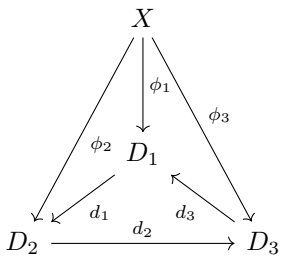
For example, we may use the the case of cartesian closed categories indexed by a pre-order:

$$I : \mathbb{P} \to \mathit{CCCat} \quad \text{The indexing functor} \tag{2.13}$$
$$A \in \mathtt{obj}\ \mathbb{P} \mapsto \mathbb{C} \in \mathit{CCCat} \quad \text{Objects are mapped to categories} \tag{2.14}$$
$$A \leq B \mapsto (A \leq B)^* : \mathbb{C} \to \mathbb{D} \quad \text{Morphisms are mapped to functors preserving CCC properties.} \tag{2.15}$$

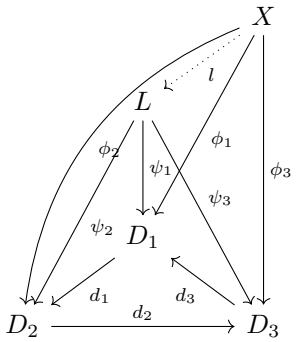
2.13 Limits

A limit is a generalisation of concepts such as terminal objects, products, and pullbacks. Co-products and initial objects are examples of co-limits.

For a diagram D in $\mathbb{C}$, we define a cone of D to be an instance of D, accompanied by another object X and a collection of morphisms $\vec{\phi}$ completing the diagram. If one imagines the diagram as a 2-dimensional shape in C with its morphisms as edges, the object X above the plane forms a cone shape.



A limit of the diagram is a least-cone. That is, it is a cone object L with morphisms $\vec{\psi}$ such that any other cone X can be factored into a unique morphism l to L and the cone formed by L.



Chapter 3

The Polymorphic-Effect-Calculus

In this chapter I'll be introducing the monadic, effect-ful language used in the rest of the dissertation, known from now on as the Effect Calculus (EC). Then I shall introduce polymorphic terms to the EC which yield the Polymorphic Effect Calculus (PEC).

3.1 Effect Calculus

The basic effect calculus is an extension of the simply typed lambda calculus to include constants, if-statements, effects, and sub-typing.

It has terms of the following form:

$$v ::= \mathtt{C}^A \mid x \mid \mathtt{true} \mid \mathtt{false} \mid () \mid \lambda x : A.v \mid v_1\ v_2 \mid \mathtt{return} v \mid \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 \mid \mathtt{if}_A\ v\ \mathtt{then}\ v_1\ \mathtt{else}\ v_2 \tag{3.1}$$

Where $\mathtt{C}^A$ is one of collection of ground constants, and A ranges over the types:

$$A, B, C ::= \gamma \mid A \to B \mid \mathtt{M}_\epsilon A \tag{3.2}$$

Where γ is from a collection of ground types, including $\mathtt{Unit}, \mathtt{Bool}$, and ϵ ranges over pre-ordered monoid of effects: $(E, \cdot, \leq, 1)$

TODO: This is very vague so far. I don't want to get too bogged down in the semantics of the Effect Calculus though A full derivation and proof of soundness of the semantics of the Effect Calculus can be found online **TODO: Link** as it is too long to include here and many of the concepts will be repeated in the rest of this dissertation anyway. The categorical semantics of the Effect Calculus require a CCC with a strong graded monad, sub-typing morphisms, sub-effecting natural transformations, and a co-product on the terminal object. These features allow us to prove the soundness of the effect-calculus semantics.

3.2 Polymorphic Effect Calculus

Next, we shall consider the Effect Calculus extended with terms to allow System-F style polymorphism over effects.

$$v ::= .. \mid \Lambda\alpha.v \mid v\ \epsilon \tag{3.3}$$

$$A, B, C ::= ... \mid \forall\alpha.A \tag{3.4}$$

Where effects ϵ now range over the effect pre-ordered monoid augmented with effect variables from an environment $\Phi = \diamond, \alpha, \beta...$, written as $(E_\Phi, \cdot_\Phi, \leq_\Phi, \mathtt{1})$.

3.3 Type System

3.3.1 Environments

As mentioned before, effects can now include effect variables. These are managed in the type system using a well-formed effect-variable-environment Φ, which is a snoc-list.

$$\Phi ::= \diamond \mid \Phi, \alpha \tag{3.5}$$

3.3.2 Effects

The ground effects form the same monotonous, pre-ordered monoid $(E, \cdot, \mathtt{1}, \leq)$ over ground elements e. For each effect environment Φ, we define a new, symbolic pre-ordered monoid:

$$(E_\Phi, \cdot_\Phi, \mathtt{1}, \leq_\Phi) \tag{3.6}$$

Where E_Φ is the closure of $E \cup \{\alpha \mid \alpha \in \Phi\}$ under $\cdot_\Phi$, which is defined as:

$$()\frac{\epsilon_3 = \epsilon_1 \cdot \epsilon_2}{\epsilon_3 = \epsilon_1 \cdot_\Phi \epsilon_2} \tag{3.7}$$

For variable-free terms and is defined symbolically for variable containing terms. Further more, we also define the sub-effecting relation in terms of its variables and the ground relation.

$$\epsilon_1 \leq_\Phi \epsilon_2 \Leftrightarrow \forall\sigma \downarrow .\epsilon_1\,[\sigma \downarrow] \leq \epsilon_2\,[\sigma \downarrow] \tag{3.8}$$

Where $\sigma \downarrow$ denotes any ground-effect-substitution of Φ. That is any substitution of all effect-variables in Φ to ground effects. Where it is obvious from the context, I shall use $\leq$ instead of $\leq_\Phi$.

3.3.3 Types

As stated, types are now generated by the following grammar.

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

3.3.4 Type Environments

As is often the case in similar type systems, a type environment is a snoc-list of term-variable, type pairs, $G ::= \diamond \mid \Gamma, x : A$.

Domain Function on Type Environments

$$\texttt{dom}(\diamond) = \emptyset \qquad \texttt{dom}(\Gamma, x : A) = \texttt{dom}(\Gamma) \cup \{x\}$$

3.3.5 Well-Formed-Ness Predicates

To formalise properties of the type system, it will be useful to have a collection of predicates ensuring that structures in the language are well-behaved with respect to their use of effect variables.

Informally, $\alpha \in \Phi$ if α appears in the list represented by Φ.

The $\texttt{Ok}$ predicate on effect environments asserts that the effect environment does not contain any duplicate effect-variables.

$$(\text{Atom})\frac{}{\diamond \texttt{Ok}} \quad (\text{A})\frac{\Phi \texttt{Ok} \;\; \alpha \notin \Phi}{\Phi, \alpha \texttt{Ok}}$$

Using this, we can define the well-formed-ness relation on effects, $\Phi \vdash \epsilon$. In short, this relation ensures that effects don't reference variables that are not in the effect environment.

$$(\text{Ground})\frac{\Phi \texttt{Ok}}{\Phi \vdash e} \quad (\text{Var})\frac{\Phi, \alpha \texttt{Ok}}{\Phi, \alpha \vdash \alpha} \quad (\text{Weaken})\frac{\Phi \vdash \alpha}{\Phi, \beta \vdash \alpha}(\text{if } \alpha \neq \beta) \quad (\text{Monoid Op})\frac{\Phi \vdash \epsilon_1 \;\; \Phi \vdash \epsilon_2}{\Phi \vdash \epsilon_1 \cdot \epsilon_2}$$

The well-formed-ness of effects can be used to a similar well-typed-relation on types, $\Phi \vdash A$, which asserts that all effects in the type are well-formed.

$$(\text{Ground})\frac{}{\Phi \vdash \gamma} \quad (\text{Lambda})\frac{\Phi \vdash A \;\; \Phi \vdash B}{\Phi \vdash A \to B} \quad (\text{Computation})\frac{\Phi \vdash A \;\; \Phi \vdash \epsilon}{\Phi \vdash \mathtt{M}_\epsilon A} \quad (\text{For-All})\frac{\Phi, \alpha \vdash A}{\Phi \vdash \forall \alpha . A}$$

Finally, we can derive the a well-formed-ness of type-environments, $\Phi \vdash \Gamma \texttt{Ok}$, which ensures that all types in the environment are well formed.

$$(\text{Nil})\frac{}{\Phi \vdash \diamond \texttt{Ok}} \quad (\text{Var})\frac{\Phi \vdash \Gamma \texttt{Ok} \;\; x \notin \texttt{dom}(\Gamma) \;\; \Phi \vdash A}{\Phi \vdash \Gamma, x : A \texttt{Ok}}$$

3.3.6 Sub-typing

There exists a sub-typing pre-order relation $\leq:_\gamma$ over ground types. That is:

$$(\text{Reflexive})\frac{}{A \leq:_\gamma A} \quad (\text{Transitive})\frac{A \leq:_\gamma B \;\; B \leq:_\gamma C}{A \leq:_\gamma C}$$

We extend this relation with the function, effect, and effect-lambda sub-typing rules to yield the full sub-typing relation under an effect environment, Φ, $\leq:_\Phi$

$$(\text{ground})\frac{A \leq:_\gamma B}{A \leq:_\Phi B} \quad (\text{Fn})\frac{A \leq:_\Phi A' \;\; B' \leq:_\Phi B}{A' \to B' \leq:_\Phi A \to B} \quad (\text{All})\frac{A \leq:_\Phi A'}{\forall \alpha . A \leq:_\Phi \forall a . A'} \quad (\text{Effect})\frac{A \leq:_\Phi B \;\; \epsilon_1 \leq_\Phi \epsilon_2}{\mathtt{M}_{\epsilon_1} A \leq:_\Phi \mathtt{M}_{\epsilon_2} B}$$

3.3.7 Type Rules

We define a fairly standard set of type rules on the language. **TODO: stack these horizontally to reduce the vertical space.**

$$(\text{Const})\frac{\Phi \vdash \Gamma \texttt{Ok} \;\; \Phi \vdash A}{\Phi \mid \Gamma \vdash \texttt{C}^A : A} \quad (\text{Unit})\frac{\Phi \vdash \Gamma \texttt{Ok}}{\Phi \mid \Gamma \vdash \texttt{()} : \texttt{Unit}} \quad (\text{True})\frac{\Phi \vdash \Gamma \texttt{Ok}}{\Phi \mid \Gamma \vdash \texttt{true} : \texttt{Bool}} \quad (\text{False})\frac{\Phi \vdash \Gamma \texttt{Ok}}{\Phi \mid \Gamma \vdash \texttt{false} : \texttt{Bool}}$$

$$(\text{Var})\frac{\Phi \vdash \Gamma, x : A \texttt{Ok}}{\Phi \mid \Gamma, x : A \vdash x : A} \quad (\text{Weaken})\frac{\Phi \mid \Gamma \vdash x : A \;\; \Phi \vdash B}{\Phi \mid \Gamma, y : B \vdash x : A}(\text{if } x \neq y) \quad (\text{Fn})\frac{\Phi \mid \Gamma, x : A \vdash v : \beta}{\Phi \mid \Gamma \vdash \lambda x : A.v : A \to B}$$

$$(\text{Sub})\frac{\Phi \mid \Gamma \vdash v : A \;\; A \leq_{:\Phi} B}{\Phi \mid \Gamma \vdash v : B} \quad (\text{Effect-Abs})\frac{\Phi, \alpha \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \Lambda\alpha.v : \forall\alpha.A} \quad (\text{Effect-apply})\frac{\Phi \mid \Gamma \vdash v : \forall\alpha.A \;\; \Phi \vdash \epsilon}{\Phi \mid \Gamma \vdash v\ \epsilon : A\,[\epsilon/\alpha]}$$

$$(\text{Return})\frac{\Phi \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \texttt{return} v : \texttt{M}_{\texttt{1}} A} \quad (\text{Apply})\frac{\Phi \mid \Gamma \vdash v_1 : A \to \texttt{M}_\epsilon B \;\; \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash v_1\ v_2 : \texttt{M}_\epsilon B}$$

$$(\text{If})\frac{\Phi \mid \Gamma \vdash v : \texttt{Bool} \;\; \Phi \mid \Gamma \vdash v_1 : A \;\; \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash \texttt{if}_A\ V\ \texttt{then}\ v_1\ \texttt{else}\ v_2 : A} \quad (\text{Do})\frac{\Phi \mid \Gamma \vdash v_1 : \texttt{M}_{\epsilon_1} A \;\; \Phi \mid \Gamma, x : A \vdash v_2 : \texttt{M}_{\epsilon_2} B}{\Phi \mid \Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2 : \texttt{M}_{\epsilon_1 \cdot \epsilon_2} B}$$

3.3.8 Ok Lemma

The first lemma used in this dissertation is that: If $\Phi \mid \Gamma \vdash v : A$ then $\Phi \vdash \Gamma\texttt{Ok}$.

Proof If $\Gamma, x : A\texttt{Ok}$ then by inversion $\Gamma\texttt{Ok}$ Only the type rule `Weaken` adds terms to the environment from its preconditions to its post-condition and it does so in an `Ok` preserving way. Any type derivation tree has at least one leaf. All leaves are axioms which require $\Phi \vdash \Gamma\texttt{Ok}$. And all non-axiom derivations preserve the `Ok` property.

Chapter 4

Semantics for EC in an S-Category

As hinted at previously, we can interpret the Effect Calculus in a CCC with a strong graded monad and co-products. A further requirement is the appropriate sub-effecting natural transformations. For each instance of $\epsilon_1 \leq \epsilon_2$, there exists a natural transformation $[\![\epsilon_1 \leq \epsilon 2]\!] : T_{\epsilon_1} \to T_{\epsilon_2}$ such that it has the following interactions with the graded monad:

TODO: Put these horizontal to each other.

4.0.1 Sub-Effecting and Tensor Strength

$$\begin{array}{ccc} A \times T_{\epsilon_1} B & \xrightarrow{\mathtt{Id}_A \times [\![\epsilon_1 \leq \epsilon_2]\!]_B} & A \times T_{\epsilon_2} B \\ \downarrow{\scriptstyle \mathtt{t}_{\epsilon_1,A,B}} & & \downarrow{\scriptstyle \mathtt{t}_{\epsilon_2,A,B}} \\ T_{\epsilon_1}(A \times B) & \xrightarrow{[\![\epsilon_1 \leq \epsilon_2]\!]_{A \times B}} & T_{\epsilon_2}(A \times B) \end{array}$$

$$\begin{array}{ccccc} T_{\epsilon_1} T_{\epsilon_2} & \xrightarrow{T_{\epsilon_1}[\![\epsilon_2 \leq \epsilon_2']\!]_M} & T_{\epsilon_1} T_{\epsilon_2'} & \xrightarrow{[\![\epsilon_1 \leq \epsilon_1']\!]_{M,T_{\epsilon_2'}}} & T_{\epsilon_1'} T_{\epsilon_2'} \\ \downarrow{\scriptstyle \mu_{\epsilon_1,\epsilon_2,}} & & & & \downarrow{\scriptstyle \mu_{\epsilon_1',\epsilon_2',}} \\ T_{\epsilon_1 \cdot \epsilon_2} & & \xrightarrow{\qquad [\![\epsilon_1 \cdot \epsilon_2 \leq \epsilon_1' \cdot \epsilon_2']\!]_M \qquad} & & T_{\epsilon_1' \cdot \epsilon_2'} \end{array}$$

We shall call a category fulfilling these properties an S-Category (Semantic Category).

TODO: This has been very vague as I want to save words on the non-polymorphic stuff.

Chapter 5

Semantics For PEC in an Indexed Category

In this chapter, I shall describe the category structure required to interpret an instance of the PEC. I shall then present denotations of each type of structure in the language, such as types, effects, terms, substitutions, and environment weakenings. Finally, I shall provide outlines and interesting cases of the proofs of the lemmas leading up to and including soundness of $\beta\eta$-conversion.

5.1 Required Category Structure

5.2 Road Map

TODO: Diagram. The first pair of theorems is effect-substitution theorem on effects. These theorems show that substitutions of effects have well-behaved and easily defined action upon the denotations of effects.

Using these theorems, we can then move on to characterize the action of effect-substitutions and effect-environment-weakening on the denotations of types and type-environments.

Using the above, we can characterize the action of effect-substitutions and effect-environment weakening on the denotations of terms.

From this we can characterize the action of effect-weakening on term-substitutions and term weakenings. This gives us the means to prove the substitution and weakening theorems for typing environments.

The substitution and weakening theorems allow us to easily find the denotation of a substituted term, simply by pre-composing the term before substitution with the denotation of the substitution or the weakening.

Separately, we prove that all derivable denotations for a typing relation instance, $\Phi \mid \Gamma \vdash v\colon A$ have the same denotation. This is important, since sub-typing allows us to find multiple distinct typing derivations for terms, which initially look like they may have distinct denotations. Using a reduction function to transform typing derivations into a unique form, I shall prove that all typing derivations yield equal denotations.

This collection of theorems finally allows us to complete all cases of the $\beta\eta$-equivalence soundness theorem.

5.3 Denotations

5.4 Substitution and Weakening Theorems