

A Denotational Semantics for Polymorphic Effect Systems

Alexander Taylor

April 8, 2019

TODO: Be consistent in theorem names (Typing-environment substitution, etc) **TODO:** Effect-App -; Effect-Spec, Effect-Gen + other rule names **TODO:** Be consistent in N-T names (unit/point/pure, bind/join)

Abstract

To date, there has been limited work on the semantics of languages with polymorphic effect systems. The application, by Moggi [1], of strong monads to modelling the semantics of effects has become a mainstream concept in functional programming languages. This was improved upon by Katsumata ([2]) 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 [3] 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 introduce a modern definition of a lambda-calculus-based language with an explicit graded monad to handle a variety of effects. This calculus is then extended using polymorphic terms to yield a more general polymorphic effect calculus. Next, I 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 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 (**Reference Needed: (link)**), since due to the number of theorems and cases, the total size is well over 100 pages of definitions, theorems, and proofs.

Contents

1	Introduction	4
1.1	What is an Effect System?	4
1.2	What is Effect Polymorphism?	4
1.3	An Introduction to Categorical Semantics	5
2	Background	7
2.1	Required Category Theory	7
2.1.1	Cartesian Closed Category	7
2.1.2	Co-Product	8
2.1.3	Functors	8
2.1.4	Natural Transformations	9
2.1.5	Monad	9
2.1.6	Monoid	9
2.1.7	Graded Monad	10
2.1.8	Tensor Strength	10
2.1.9	Adjunction	10
2.1.10	Strictly Indexed Category	12
2.2	Language Features and Their Requirements	12
2.3	The Effect Calculus	15
2.4	Polymorphic Effect Calculus	16
2.4.1	Type System	18
3	The Semantics of PEC in an Strictly Indexed Category	22
3.1	Semantics for EC in an S-Category	22
3.2	Required Category Structure	23
3.3	Road Map	25
3.4	Denotations	25
3.5	Substitution and Weakening Theorems	27
3.6	Uniqueness of Denotations	45
3.7	Soundness	50

4	Instantiating a Model of PEC	54
5	Conclusion	63
	Bibliography	64

Chapter 1

Introduction

1.1 What is an Effect System?

Programs very rarely stand alone without interacting with their environment in some form. This interaction might be to receive input, such as from sensors or buttons, to write output to a file, or to halt with an error. In functional languages, manipulating mutable state can also be considered interacting with the environment. Language terms that produce these interactions are said to have *effects* as they have some observable effect in the environment other than simply the value that executing the term produces. It is important to be able to reason about the effects that a program term may produce, in order to ensure the soundness of compiler transformations and formal verification. An effect system, analogously to a type system, is a formal system of rules which infers abstract information about the side-effects that a program might have. This non-concrete data may then inform tools such as a compiler of the soundness of code transformations, such as removing redundant code or reordering statements. Languages such as Haskell and the languages introduced later in this dissertation do not allow implicit effects. Instead they require the programmer to explicitly use structures called monads [1] to encode effect analysis in the type of program fragments. This has the useful result of combining the effect system with the type system of the language. For example, a program of type `Int` cannot interact with its environment to produce its result and hence always yields the same return value. However a program of type `IO Int` may perform IO operations before returning its result and hence may not always behave in exactly the same fashion.

1.2 What is Effect Polymorphism?

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

```

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 situations 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 verifying and reasoning about such tools.

A key property of effect polymorphism that distinguishes it from type polymorphism is that effects are not impredicative. That is, types are polymorphic over effects, but effects do not range over themselves. This means that the models used to interpret effect-polymorphic languages can be simpler than the models required to interpret languages with impredicative polymorphism, such as System F [4].

1.3 An Introduction to Categorical Semantics

In this dissertation, I describe a denotational semantics using category theory. A denotational semantics for a language is a mapping, the image $\llbracket X \rrbracket$ of which is known as a denotation, of structures in the language, such as types and terms to mathematical objects in a compositional way. This means that the denotation of a term is defined entirely in terms of the denotations of its subterms.

When we specify a denotational semantics of a language in category theory, we look to find a categorical structure to which can construct a mapping of types and typing environments to objects in the category. That is, there should exist objects $\llbracket A \rrbracket, \llbracket \Gamma \rrbracket$ in the category \mathbb{C} such that:

$$\begin{aligned}
A : \text{Type} &\mapsto \llbracket A \rrbracket \in \text{obj } \mathbb{C} \\
\Gamma &\mapsto \llbracket \Gamma \rrbracket \in \text{obj } \mathbb{C}
\end{aligned}$$

Further more, assuming the language has a type system with a typing relation of the form $\Gamma \vdash v : A$, meaning “in environment Γ , language term v has type A ”, instances of the typing relation should be mapped to morphisms between the relevant objects in \mathbb{C} .

$$\Gamma \vdash v : A \mapsto \mathbb{C}(\llbracket \Gamma \rrbracket, \llbracket A \rrbracket)$$

This should occur in a sound manner with respect to some equational equivalence over the language. An equational equivalence is a relation of the form $\Gamma \vdash v_1 \cong v_2 : A$, which means “in environment Γ , terms v_1 and v_2 are equivalent under type A ”, which represents equality up to some particular property. A categorical semantics is sound with respect to an equational equivalence \cong if and only if $\Gamma \vdash v_1 \cong v_2 : A$ implies that the denotations $\llbracket \Gamma \vdash v_1 : A \rrbracket$ and $\llbracket \Gamma \vdash v_2 : A \rrbracket$ are equal morphisms in \mathbb{C} from $\llbracket \Gamma \rrbracket$ to $\llbracket A \rrbracket$.

Typically when working with lambda-calculus-based languages, we pick our equational equality to be an extension of $\beta\eta$ -equivalence to include other, language-specific, rules for equality under reduction. For example, we might want to include the execution of if-statements or effectful expressions.

An example of an inductive equivalence rule that we might want to be sound with respect to is that of the β -reduction of lambda terms. It should be the case that the equality defined in equation 1.1 implies that the denotations $\llbracket \Gamma \vdash (\lambda x : A. v_1) v_2 : B \rrbracket$ and $\llbracket \Gamma \vdash v_1 [v_2/x] : B \rrbracket$ are equal.

$$(\text{Lambda-Beta}) \frac{\Gamma, x : A \vdash v_1 : B \quad \Gamma \vdash v_2 : A}{\Gamma \vdash (\lambda x : A. v_1) v_2 \approx v_1 [v_2/x] : B} \tag{1.1}$$

In this dissertation, I shall introduce an effect-polymorphic language, describe a semantics for it, then prove that the semantics is sound with respect to a $\beta\eta$ -equivalence-based equational equivalence relation.

Chapter 2

Background

In this chapter I first introduce the required category theory to understand the rest of the dissertation. In addition, I shall explain briefly how the particular category theoretic structures can be used to model particular features of various programming languages, such as the simply typed lambda calculus and system F. Following this, I proceed to introduce the monadic, effectful language used in the rest of the dissertation, known from now on as the Effect Calculus (EC). In the final section, I extend the EC with polymorphic syntax to yield Polymorphic Effect Calculus (PEC).

2.1 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 to jog the memory of the reader, and briefly introduce some new concepts. For a more detailed treatment of these concepts, please see [5].

2.1.1 Cartesian Closed Category

A category is cartesian closed if it has a terminal object, products for all pairs of objects, and exponentials. These concepts will be explained below.

Terminal Object

An object, typically written 1 , is *terminal* in a category, \mathbb{C} if for all objects $A \in \text{obj } \mathbb{C}$, there exists exactly one morphism from A to 1 , written $\langle \rangle_A : A \rightarrow 1$.

Products

A *binary product* of a pair of objects $A, B \in \text{obj } \mathbb{C}$ consists of an object, written $A \times B$, with morphisms $\pi_1 : A \times B \rightarrow A$, $\pi_2 : A \times B \rightarrow B$ such that for any other object C and morphisms, $f : C \rightarrow A$, $g : C \rightarrow B$ there exists a unique morphism $\langle f, g \rangle : C \rightarrow (A \times B)$ such that the following commutes:

$$\begin{array}{ccccc} & & C & & \\ & f \swarrow & \downarrow \langle f, g \rangle & \searrow g & \\ A & \xleftarrow{\pi_1} & (A \times B) & \xrightarrow{\pi_2} & B \end{array}$$

A category is said to have binary products if for all objects A, B , $A \times B$ also exists.

Exponentials

An *exponential* for objects A, C is an object C^A with morphism $\mathbf{app} : C^A \times A \rightarrow C$ such that for any object B and morphism $f : B \times A \rightarrow C$, there exists a morphism $\mathbf{cur}(f) : B \rightarrow C^A$ such that the following commutes:

$$\begin{array}{ccc} C^A \times A & \xrightarrow{\mathbf{app}} & C \\ \uparrow \mathbf{cur}(f) \times \mathbf{Id}_B & \nearrow f & \\ B \times A & & \end{array}$$

A category has exponentials if for all pairs of objects A, C it has an exponential C^A .

Diagonal and Twist Morphisms

In the definition of the semantics of the if-expression, it is useful to make use of the following twist and diagonal shorthand morphisms.

$$\begin{aligned} \tau_{A,B} : (A \times B) &\rightarrow (B \times A) = \langle \pi_2, \pi_1 \rangle \\ \delta_A : A &\rightarrow (A \times A) = \langle \mathbf{Id}_A, \mathbf{Id}_A \rangle \end{aligned}$$

2.1.2 Co-Product

A *co-product* is the dual of a product. A category \mathbb{C} has a co-product for objects A, B if there exists an object $A + B$ in \mathbb{C} with morphisms $\mathbf{inl} : A \rightarrow (A + B)$, $\mathbf{inr} : B \rightarrow (A + B)$ such that for any other object C with morphisms $f : A \rightarrow C$, $g : B \rightarrow C$, there exists a unique morphism $[f, g] : (A + B) \rightarrow C$ such that the following commutes:

$$\begin{array}{ccccc} & & C & & \\ & \nearrow f & \uparrow [f,g] & \nwarrow g & \\ A & \xrightarrow{\mathbf{inl}} & (A + B) & \xleftarrow{\mathbf{inr}} & B \end{array}$$

2.1.3 Functors

A functor $F : \mathbb{C} \rightarrow \mathbb{D}$ is a mapping of objects and morphisms in \mathbb{C} to objects and morphisms respectively in \mathbb{D} that preserves composition and identities.

$$\begin{aligned} A \in \mathbf{obj} \ \mathbb{C} &\mapsto FA \in \mathbf{obj} \ \mathbb{D} \\ f : \mathbb{C}(A, B) &\mapsto F(f) : \mathbb{D}(FA, FB) \\ F(\mathbf{Id}_A) &= \mathbf{Id}_{FA} \\ F(g \circ f) &= F(g) \circ F(f) \end{aligned}$$

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

Figure 2.1: Naturality of a natural transformation

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

Figure 2.2: Monad Associativity Laws

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

Figure 2.3: Monad Left- and Right-Unit laws

2.1.4 Natural Transformations

A *natural transformation* θ between two functors $F, G : \mathbb{C} \rightarrow \mathbb{D}$ is a collection of morphisms in \mathbb{D} , indexed by objects in \mathbb{C} with $\theta_A : F(A) \rightarrow G(A)$ such that diagram in figure 2.1 commutes for each $f : A \rightarrow B \in \mathbb{C}$.

2.1.5 Monad

A *monad* consists of a functor, $T : \mathbb{C} \rightarrow \mathbb{C}$, from \mathbb{C} onto itself, also known as an *endofunctor*, which represents the collection of effectful values, and a pair of natural transformations. The first is the *unit* natural transformation $\eta_A : A \rightarrow T(A)$, which is used to treat pure values as an effectful expression. The second is the *join* (also known as *bind*) natural transformation, $\mu_A : T(T(A)) \rightarrow T(A)$, which is used to model the sequential composition of effectful subexpressions. In addition, there is the requirement that the diagrams in figures 2.2, 2.3 commute.

2.1.6 Monoid

Another concept, though not strictly of category theory, is that of a monoidal algebra. A *monoid* is a set, operation, and identity $(M, \cdot, 1)$ where $\cdot : M \times M \rightarrow M$, is associative ($a \cdot (b \cdot c) = (a \cdot b) \cdot c$), and has an identity $a \cdot 1 = a = 1 \cdot a$. In this dissertation, I shall make use of a monoid with a partial-order. This means there is a transitive, reflexive relation \leq over the set M . In this case, the \cdot operator should also be monotone. That is if $a \leq a'$ and $b \leq b'$ then $a \cdot b \leq a' \cdot b'$. Such an algebra can be used to describe how effects induced by terms in a program interact with each other. For example, $a \cdot b$ is the effect induced by composing a subexpression which produces effect a with a subexpression that produces effect b . The partial-order is used to develop a notion of *subtyping* over effects.

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

Figure 2.4: Associativity of a graded monad

$$\begin{array}{ccc}
T_{\epsilon} A & \xrightarrow{T_{\epsilon} \eta_A} & T_{\epsilon} T_1 A \\
\downarrow \eta_{T_{\epsilon} A} & \searrow & \downarrow \mu_{\epsilon, 1, A} \\
T_1 T_{\epsilon} A & \xrightarrow{\mu_{1, \epsilon, A}} & T_{\epsilon} A
\end{array}$$

Figure 2.5: Left- and Right- Units of a graded monad

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

Figure 2.6: Left Naturality of Graded Tensor Strength

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

Figure 2.7: Right Naturality of Graded Tensor Strength

2.1.7 Graded Monad

A *graded monad* is a generalisation of a monad to be indexed by a monoidal algebra $(E, \cdot, 1)$. It consists of an endofunctor indexed by elements in the monoid, $T : (E, \cdot, 1) \rightarrow [\mathbb{C}, \mathbb{C}]$, and a pair of indexed natural transformations. Firstly, there is the unit natural transformation to the monad functor indexed by the identity $\eta : \text{Id} \rightarrow T_1$. The second natural transformation is a generalisation of join which composes nested instances of the indexed functor $\mu_{\epsilon_1, \epsilon_2} : T_{\epsilon_1} T_{\epsilon_2} \rightarrow T_{\epsilon_1 \cdot \epsilon_2}$. Furthermore there is a requirement that the diagrams in figures 2.4, 2.5 commute.

2.1.8 Tensor Strength

A slightly harder concept to motivate is that of tensorial (or tensor) *strength* for a graded monad. Tensor strength consists of a natural transformation: $\mathfrak{t}_{\epsilon, A, B} : A \times T_{\epsilon} B \rightarrow T_{\epsilon}(A \times B)$, which is required to have well defined interactions with the graded monad morphisms and the product-reordering natural transformation $\alpha_{A, B, C} = \langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle : ((A \times B) \times C) \rightarrow (A \times (B \times C))$, as seen in figures 2.6, 2.7, 2.8, 2.9, 2.10, 2.11. The reasoning behind this is that the tensor strength natural transformation allows us to model operations that are invisibly implicit in a programming language but not given by default in category theory. Tensor strength of a monad can be generalised to tensor strength in a graded monad by indexing by an element of the monoid algebra. A monad (or respectively a graded monad) is called *strong* if it has tensorial strength.

2.1.9 Adjunction

An important concept in category theory is that of an Adjunction. Given functors $F : C \rightarrow D$, and $G : D \rightarrow C$ and a pair natural transformations, known respectively as the unit and co-unit: $\eta_A : A \rightarrow G(FA)$ in \mathbb{C} and $\epsilon_B : F(GB) \rightarrow B$ in \mathbb{D} , such that $\epsilon_{FA} \circ F(\eta_A) = \text{Id}_{FA}$ and $G(\epsilon_B) \circ \eta_{FB} = \text{Id}_{GB}$. We can then use ϵ and η to form a natural isomorphism between morphisms in the two categories, as seen in figure 2.12. This natural isomorphism is called an adjunction.

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

Figure 2.8: Tensor Strength Unitor Law

$$\begin{array}{ccc}
A \times T_{\epsilon_1} T_{\epsilon_2} B & \xrightarrow{\mathfrak{t}_{\epsilon_1, A, T_{\epsilon_2} B}} T_{\epsilon_1}(A \times T_{\epsilon_2} B) & \xrightarrow{T_{\epsilon_1} \mathfrak{t}_{\epsilon_2, A, B}} T_{\epsilon_1} T_{\epsilon_2}(A \times B) \\
& \searrow \text{Id}_A \times \mu_{\epsilon_1, \epsilon_2, B} & \downarrow \mu_{\epsilon_1, \epsilon_2, A \times B} \\
& A \times T_{\epsilon_1, \epsilon_2} B & \xrightarrow{\mathfrak{t}_{\epsilon_1, \epsilon_2, A, B}} T_{\epsilon_1, \epsilon_2}(A \times B)
\end{array}$$

Figure 2.9: How the tensor strength natural transformation commutes with the join natural transformation.

$$\begin{array}{ccc}
A \times B & \xrightarrow{\text{Id}_A \times \eta_B} & A \times T_1 B \\
& \searrow \eta_{A \times B} & \downarrow \mathfrak{t}_{1, A, B} \\
& & T_1(A \times B)
\end{array}$$

Figure 2.10: How the tensor strength natural transformation commutes with the unit natural transformation

$$\begin{array}{ccc}
(A \times B) \times T_\epsilon C & \xrightarrow{\mathfrak{t}_{\epsilon, (A \times B), C}} & T_\epsilon((A \times B) \times C) \\
\downarrow \alpha_{A, B, T_\epsilon C} & & \downarrow T_\epsilon \alpha_{A, B, C} \\
A \times (B \times T_\epsilon C) & \xrightarrow{\text{Id}_A \times \mathfrak{t}_{\epsilon, B, C}} A \times T_\epsilon(B \times C) & \xrightarrow{\mathfrak{t}_{\epsilon, A, (B \times C)}} T_\epsilon(A \times (B \times C))
\end{array}$$

Figure 2.11: Tensorial strength commutes with the reordering natural transformation.

$$\begin{array}{ccc}
\overline{(-)} : \mathbb{C}(FA, B) & \leftrightarrow & \mathbb{D}(A, GB) : \widehat{(-)} \\
& f \mapsto G(f) \circ \eta_A & \\
& \epsilon \circ F(g) \leftarrow g &
\end{array}$$

Figure 2.12: How to construct the adjunction isomorphism from its unit and co-unit.

2.1.10 Strictly Indexed Category

The final piece of category theory required to understand this dissertation is the concept of a strictly indexed category. A strictly indexed category is a functor from a *base category* \mathbb{C} into a target (*indexed*) category of categories, where objects are categories and morphisms are functors. Objects, A , in the base category are mapped to categories $\mathbb{C}(A)$, known as *fibres* in the indexed category. Morphisms between objects in the base category, $f : B \rightarrow A$, are contravariantly mapped to functors, written $f^* : \mathbb{C}(A) \rightarrow \mathbb{C}(B)$ and known as *re-indexing functors*, between fibres in the indexed category. The strictly adverb indicates that the indexing in this construction is done by a functor as opposed to the weaker pseudofunctor (weak 2-functor) structure. Since pseudofunctors are not needed to explain anything in this project, I shall leave out their definition, though an interested reader may wish to research them further.¹

Due to the composition laws for functors, $\theta^* \circ \phi^* = (\phi \circ \theta)^*$ and $\text{Id}_A * (B) = B \in \text{obj } \mathbb{C}(A)$, and Id^* is the identity functor. For example, we may use the category of cartesian closed categories and cartesian closed functors, (CCCat) indexed by a partial-order, \mathbb{P} :

$$\begin{aligned} I : \mathbb{P} &\rightarrow \text{CCCat} && \text{The indexing functor} \\ A \in \text{obj } \mathbb{P} &\mapsto \mathbb{C} \in \text{CCCat} && \text{Objects are mapped to categories} \\ A \leq B &\mapsto (A \leq B)^* : \mathbb{C} \rightarrow \mathbb{D} && \text{Morphisms are mapped to functors preserving CCC properties.} \end{aligned}$$

2.2 Language Features 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 section 2.1, 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 2.1.1).

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. $\llbracket \Gamma \rrbracket = \llbracket \diamond, x : A, y : B, \dots, z : C \rrbracket = 1 \times \llbracket A \rrbracket \times \llbracket B \rrbracket \times \dots \times \llbracket C \rrbracket$
- Terminal objects are used in the denotation of constant terms $\llbracket \Gamma \vdash \mathbf{c}^A : A \rrbracket = \llbracket \mathbf{c}^A \rrbracket \circ \langle \rangle_{\llbracket \Gamma \rrbracket}$
- Exponentials are used in the denotations of functions. $\llbracket \Gamma \vdash \lambda x : A. v : A \rightarrow B \rrbracket = \text{cur}(\llbracket \Gamma, x : A \vdash v : B \rrbracket)$

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 of the terminal object with itself and subtyping
Single Effect	Strong Monad
Multiple Effects	Strong Graded Monad
Polymorphism	Indexed Category

¹<https://ncatlab.org/nlab/show/pseudofunctor>

Motivating the Tensor Strength Requirement

```

let x = 5 in (
  do y <- readInt in (
    return x + y
  )
);

```

In the **return** clause, the program makes reference to variables x and y from the environment. Since x is defined inside the monadic **do** clause, it is not available to clauses in the body of the clause without a means to convert the type $(\Gamma \times \mathbf{M}A)$ to $\mathbf{M}(\Gamma \times A)$.

Figure 2.13: Program in a monadic effectful language that requires tensor strength of the effect’s monad to execute.

To model if-expressions of the form **if condition then if_true else if_false**, we need a way to combine morphisms of the form $\llbracket \Gamma \vdash \text{condition} : \text{Bool} \rrbracket : \Gamma \rightarrow \llbracket \text{Bool} \rrbracket$, $\llbracket \Gamma \vdash \text{if_true} : A \rrbracket : \Gamma \rightarrow A$, and $\llbracket \Gamma \vdash \text{if_false} : A \rrbracket : \Gamma \rightarrow A$ to form a morphism $\Gamma \rightarrow A$. If we have a co-product $\Gamma + \Gamma$, and a morphism mapping **Bools** to $\Gamma + \Gamma$, we could use the fold morphism $\llbracket \text{if_true} \rrbracket, \llbracket \text{if_false} \rrbracket$ to achieve the required. It turns out that using exponentials, as seen in the denotation of the (If) type rule in figure 3.5, we can factor out the Γ to instead use a co-product $1 + 1$ instead of $\Gamma + \Gamma$. It now a natural choice to use $1 + 1$ as $\llbracket \text{Bool} \rrbracket$. Hence we can model the boolean values **true**, **false** as the co-product constructors **inl**, **inr**.

A single effect can be modelled by adding a strong monad to the category, as shown by Moggi [1]. A language with an explicit monad in its type system requires two operations: *return* and *bind*, the type rules for which can be seen in equation 2.1. The $\mathbf{M}(-)$ type constructor represents values which have an instance of the effect associated with their computation. The *return* operator lifts pure values (with no associated effects) into the effectful type constructor. The *bind* operator (often supplemented with some **do ... in ...** syntax) allows us to compose together expressions which have effects. As shown by Moggi [1], these two operations can be modelled using the *unit* and *join* natural transformations of a strong graded monad. The monad needs to be strong in order to allow access to variables in the environment from within the monadic expression. An example of this can be seen in figure 2.13.

$$\begin{array}{c}
\text{(Return)} \frac{\Gamma \vdash v : A}{\Gamma \vdash \text{return } v : \mathbf{M}A} \quad \text{(Bind)} \frac{\Gamma \vdash v_1 : \mathbf{M}A \quad \Gamma, x : A \vdash v_2 : \mathbf{M}B}{\Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : \mathbf{M}B}
\end{array} \tag{2.1}$$

For a more precise analysis of languages with multiple effects, we can look into whether there is an algebra on the effects. For example, we might want to express what the composite effect of two sequential expressions with different effects. We could follow an expression that might throw an exception with one that accesses mutable state and a third that carries out IO transactions. We would also want some form of *unit* effect for pure expressions which does nothing when composed with other effects. Finally, to make it easier to analyse branched code, such as *if expressions*, some form of subtyping would be useful. This structure is modelled exactly by an appropriate partially-ordered monoidal algebra $(E, \cdot, \leq, 1)$. The set E gives the various effects that can be produced, 1 represents the unit effect for pure values, \cdot allows us to associatively compose multiple effects and the partial-order \leq gives us a subtyping of effects for an intuitive if-statement programming model.

In order to embed this algebra-based effect analysis in the type system, we can index Moggi’s monadic

type constructor \mathbf{M} with the effect ϵ that is produced when the corresponding expression is evaluated. Further more, when we use the *return* operation to lift pure values into the monadic type constructor, the resulting monadic type should have an index of 1 indicating that the effect produced is pure. Finally, when we *bind* together effectful expressions, the resulting effect should be the composition of the effects of the subexpressions. Putting together these requirements yields the new type rules given in equation 2.2. By suitably extending the monad laws to account for this new indexing, we find that we require a strong graded monad to model these features using category theory. This construct was first documented in the context of semantics by Katsumata ([2]).

$$\text{(Return)} \frac{\Gamma \vdash v : A}{\Gamma \vdash \text{return } v : \mathbf{M}_1 A} \quad \text{(Bind)} \frac{\Gamma \vdash v_1 : \mathbf{M}_{\epsilon_1} A \quad \Gamma, x : A \vdash v_2 : \mathbf{M}_{\epsilon_2} B}{\Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : \mathbf{M}_{\epsilon_1 \cdot \epsilon_2} B} \quad (2.2)$$

When we combine an effect system with if-expressions, we come across another, language-level, issue. When programming using the construct, we might want the true and false branches of the expressions to have different effects. For example, one branch of an expression may perform I/O operations, whilst the other has no side effects. Since effect analysis is done by the type system, this means that the two branches have different types. Hence the typical type rule for if-expressions, as given in equation 2.3, does not hold. A solution for this is to introduce a notion of subtyping. If $\Gamma \vdash v : A$ and $A \leq B$, then $\Gamma \vdash v : B$. Using the partial-order on effects to generate the subtyping relation, we can now unify compatible, though not equal, effects in an if-expression.

$$\text{(If)} \frac{\Gamma \vdash \text{condition} : \text{Bool} \quad \Gamma \vdash \text{if_true} : A \quad \Gamma \vdash \text{if_false} : A}{\Gamma \vdash \text{if condition then if_true else if_false} : A} \quad (2.3)$$

To model sub-typing, we need a way to convert morphisms $\llbracket \Gamma \vdash v : A \rrbracket : \Gamma \rightarrow A$ to $\llbracket \Gamma \vdash v : B \rrbracket : \Gamma \rightarrow B$ when $A \leq B$. This can be done if for each instance of $A \leq B$, we have a there is morphism $\llbracket A \leq B \rrbracket : A \rightarrow B$. These morphisms should respect the antisymmetry, transitivity, and reflexivity of the subtyping relation, meaning that $\llbracket B \leq C \rrbracket \circ \llbracket A \leq B \rrbracket = \llbracket A \leq C \rrbracket$ and $\llbracket A \leq A \rrbracket = \text{Id}_A$.

Polymorphism is a harder concept to explain intuitively. The particular flavour of polymorphism that this dissertation discusses is System-F-Style parametric polymorphism. For a given type-system feature F , such as types, effects, type constructors, or some other language specific feature, we can add parametric polymorphism over F by allowing instances of F to be replaced by F variables and introducing certain type expressions to modify the types. For example, let us look at system F [6]. System F has polymorphism over types. This means that we can replace a type expression with a type variable parameter. We also need to be able to syntactically generalise over type parameters and to be able to specify a particular type parameter's value. This requires us to extend the syntax of the simply typed lambda calculus with the appropriate specialisation and generalisation terms as well as parameterised types, as seen in figure 2.14.

To correctly account for these new terms and types in the type system, we need to ensure that all parameterisations are soundly constructed, in a similar way to how a closed term in the simply typed lambda calculus has no free variables. To do this, we introduce a new F -environment Φ to complement the typing environment Γ . A term or type is well formed (written $\Phi \vdash f$) only if it only contains F variables in Φ . An F expression, f , is also well formed only if it only contains F variables in Φ . We can now specify type rules for the generalisation (parameterisation of an expression over an F -variable) and specialisation (substituting a parameter for a value) of terms. As seen in equation 2.4.

$$\text{(Gen)} \frac{\Phi, \alpha \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \Lambda \alpha. v : \forall \alpha. A} \quad \text{(Spec)} \frac{\Phi \mid \Gamma \vdash v : \forall \alpha. A \quad \Phi \vdash f}{\Phi \mid \Gamma \vdash v f : A[f/\alpha]} \quad (2.4)$$

If we can model the language without the polymorphic terms, with each specific F environment, then we can instantiate a collection of categories, each of which models the language at a given F environment.

<p style="text-align: center;">System F</p> <p style="text-align: center;">Types are extended with variables and parameterisation terms.</p> $A ::= \dots \mid \alpha \mid \forall \alpha. A$ <p style="text-align: center;">Values are extended with generalisation and specialisation terms respectively.</p> $t ::= \dots \mid \Lambda \alpha. t \mid t A$
--

Figure 2.14: The extensions made to the simply typed lambda calculus which yield System F

This collection of categories (or fibres) can be indexed by a base category, the structure of which models the F environments and relationships between them. We can hence construct an indexed category, whereby each F environment in the base category is mapped to the fibre modelling the semantics at the specific environment, and morphisms between F environments correspond to functors between the respective fibres. Figure 2.15 demonstrates this construction. If we model an F environment Φ, α as a product, then the π_1 morphism represents removing α from the environment and the π_1^* functor conversely increases the size of the environment. As will be show later in this dissertation, if π_1^* has a right adjoint (see section 2.1.9), \forall , then this adjunction can be used to model generalisation and specialisation of polymorphic terms. This will be explained in the next chapter.

How the fibres are derived from objects in the base category depends on the polymorphic properties of the language being modelled. For example, in system F, types are impredicative. That is, types can quantify over any other types, including themselves. This means that there has to be a strong coupling between the base category, which represents type-variable environments and transformations upon them and objects in the fibres, which represent types. This typically manifests in the set of objects in each fibre being in bijection with the set of morphisms from the appropriate type-variable environment in the base category. Since, in effect-polymorphic languages, types quantify over effects, but effects do not quantify over themselves, we can conceptually decouple the objects in the fibres from the base category, meaning that effect-polymorphic models are simpler to define.

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

2.3 The Effect Calculus

The basic effect calculus is an extension of the simply typed lambda calculus to include constants, **if** expressions, effects, and subtyping. It has terms of the following form:

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

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

$$A, B, C ::= \gamma \mid A \rightarrow B \mid \mathbf{M}_\epsilon A$$

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

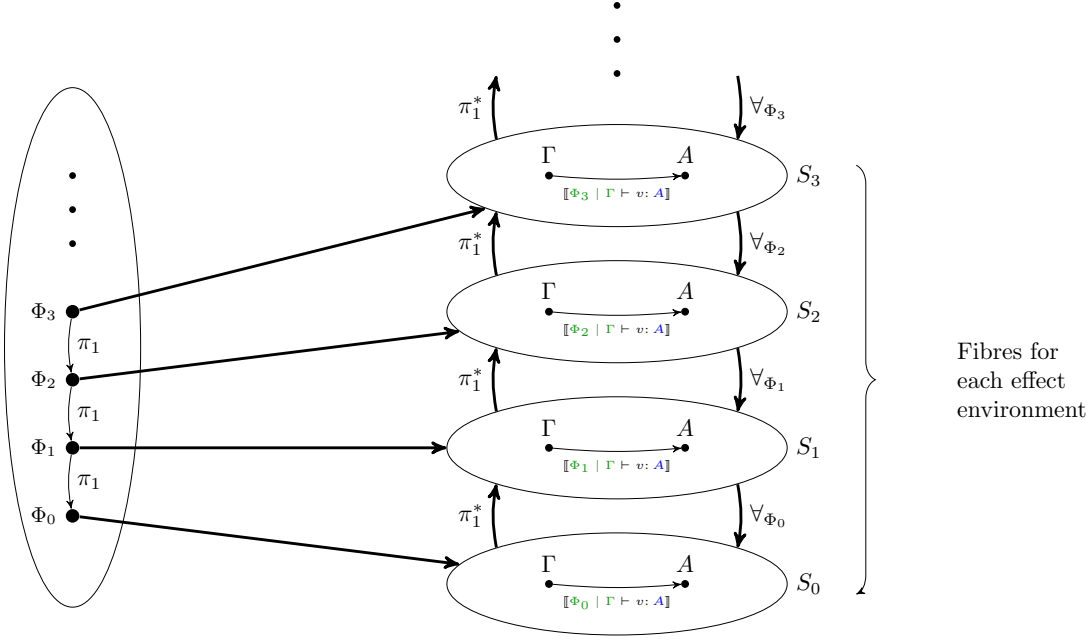


Figure 2.15: Diagram of the structure of an indexed category for modelling a polymorphic language. Thick arrows between categories represent functors and finer arrows within categories represent internal morphisms. The left hand category is the base category.

$$(\lambda x: A. v_1) v_2 \rightsquigarrow v_1 [v_2/x]$$

Figure 2.16: The β reduction lambda terms of lambda terms in the effect calculus.

The calculus has a simple, Haskell style semantics. Lambda terms when applied to a parameter substitute their bound variable for the parameter expression, as seen in figure 2.16, if statements pick a branch once their condition is decided to be true or false, figure 2.17, and finally the monadic effects behave in a similar way to Haskell’s monad type class. It is difficult to simply formalise an operational $\beta\eta$ -reduction semantics for a monadic language without a concrete instantiation of the graded monad. Instead, the monad should obey the equational laws given in figure 2.18.

As an example, by instantiating the language with the appropriate constants, ground effects, and ground types, we can write the programs in figure 2.19.

Already, we can see where having effect polymorphism in the language would be useful. We could make the first example in figure 2.19 into a more general procedure that prompts a user to confirm any I-O action, as can be seen in the extended example in figure 2.20.

2.4 Polymorphic Effect Calculus

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

```

ifA true then v1 else v2  $\rightsquigarrow$  v1
ifA true then v2 else v2  $\rightsquigarrow$  v2

```

Figure 2.17: The reduction of if expressions in the effect calculus.

```

do x  $\leftarrow$  return v1 in v2  $\rightsquigarrow$  v2 [v1/x]
do x  $\leftarrow$  v in return x  $\rightsquigarrow$  v
do x  $\leftarrow$  v1 in (do y  $\leftarrow$  v2 in v3)  $\rightsquigarrow$  do y  $\leftarrow$  (do y  $\leftarrow$  v1 in v2) in v3

```

Figure 2.18: The monad laws for the effect calculus.

```

do b <- Prompt("Are You Sure?") in (
  if b then
    FireMissiles
  else
    AbortMissiles
)

```

```

λ alice: Agent. (
  λ bob: Agent. (
    do amount <- AwaitPayment(alice)
    in SendPayment(bob, amount)
  )
)

```

Figure 2.19: A pair of examples of non-polymorphic programs

```

Λ Action: Effect.
λ ifConfirmed: Action.
λ ifAborted: Action.
do b <- Prompt("Are You Sure?") in (
  if b then
    ifConfirmed
  else
    ifAborted
)

```

Figure 2.20: A polymorphic version of the checking example

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

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

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

Where effects ϵ now range over the effect partially-ordered monoid augmented with effect variables from an environment $\Phi = \diamond, \alpha, \beta \dots$ (where \diamond represents the empty list), written as $(E_\Phi, \cdot_\Phi, \leq_\Phi, 1)$. The ground effects of E are now ranged over by e and ϵ ranges over effects including effect variables. The operator \cdot_Φ is the extension of the original \cdot operator to symbolically act on expressions including effect variables. For example, if for some ϵ , $\forall \epsilon'. \epsilon' \cdot \epsilon = \epsilon'$ then $\alpha \cdot_\Phi \epsilon = \alpha$ for some appropriate effect-variable environment Φ .

2.4.1 Type System

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.

$$\Phi ::= \diamond \mid \Phi, \alpha$$

Effects

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

$$(E_\Phi, \cdot_\Phi, 1, \leq_\Phi) \tag{2.5}$$

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

$$\frac{\epsilon_3 = \epsilon_1 \cdot \epsilon_2}{\epsilon_3 = \epsilon_1 \cdot_\Phi \epsilon_2} \tag{2.6}$$

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

$$\epsilon_1 \leq_\Phi \epsilon_2 \Leftrightarrow \forall \sigma \downarrow. \epsilon_1 [\sigma \downarrow] \leq \epsilon_2 [\sigma \downarrow] \tag{2.7}$$

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_Φ .

Types

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

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

Type Environments

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

Domain Function on Type Environments

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

Well-Formedness 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 Ok predicate on effect environments asserts that the effect environment does not contain any duplicated effect variables.

$$(\text{Empty}) \frac{}{\diamond \mathsf{Ok}} \quad (\text{Extend}) \frac{\Phi \mathsf{Ok}}{\Phi, \alpha \mathsf{Ok}} (\text{if } \alpha \notin \Phi)$$

Using this, we can define the well-formedness relation on effects, $\Phi \vdash \epsilon$. In short, this relation ensures that effects do not reference variables that are not in the effect environment. This is equivalent to saying that $\Phi \vdash \epsilon$ means that $\epsilon \in E_\Phi$.

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

The well-formedness 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.

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

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

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

Subtyping

We assume that the set of ground types with a subtyping partial-order relation \leq_γ . This subtyping relationship could be the trivial partial-order, which only includes $A \leq_\gamma B$ if $A = B$, or could be a more interesting relation. As this relationship is a partial-order, it is transitive and reflexive.

$$\text{(Reflexive)} \frac{}{A \leq_\gamma A} \quad \text{(Transitive)} \frac{A \leq_\gamma B \quad B \leq_\gamma C}{A \leq_\gamma C}$$

Using this ground type relation, we can then construct a subtyping relation between pairs of types A, B with respect to an effect environment Φ .

$$\text{(ground)} \frac{A \leq_\gamma B}{A \leq_\Phi B} \quad \text{(Fn)} \frac{A \leq_\Phi A' \quad B' \leq_\Phi B}{A' \rightarrow B' \leq_\Phi A \rightarrow B} \quad \text{(All)} \frac{A \leq_\Phi A'}{\forall \alpha. A \leq_\Phi, \alpha \forall \alpha. A'} \text{(if } \alpha \notin \Phi \text{)} \quad \text{(Effect)} \frac{A \leq_\Phi B \quad \epsilon_1 \leq_\Phi \epsilon_2}{M_{\epsilon_1} A \leq_\Phi M_{\epsilon_2} B}$$

Since the ground subtyping and subeffecting relations are partial-order, it is fairly simple to prove by induction that the new relation is also a partial-order. By induction, it is also easy to prove that if $A \leq_\Phi B$ and $\Phi \vdash A$ then $\Phi \vdash B$. This last property comes about because the labelled subeffecting relation \leq_Φ only holds between effects in E_Φ .

Type Rules

We define a fairly standard set of type rules on the language.

$$\begin{aligned} \text{(Const)} & \frac{\Phi \vdash \Gamma \text{ Ok} \quad \Phi \vdash A}{\Phi \mid \Gamma \vdash C^A : A} \quad \text{(Unit)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\Phi \mid \Gamma \vdash () : \text{Unit}} \quad \text{(True)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\Phi \mid \Gamma \vdash \text{true} : \text{Bool}} \quad \text{(False)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\Phi \mid \Gamma \vdash \text{false} : \text{Bool}} \\ \text{(Var)} & \frac{\Phi \vdash \Gamma, x : A \text{ Ok}}{\Phi \mid \Gamma, x : A \vdash x : A} \quad \text{(Weaken)} \frac{\Phi \mid \Gamma \vdash x : A \quad \Phi \vdash B}{\Phi \mid \Gamma, y : B \vdash x : A} \text{(if } x \neq y, y \notin \text{dom}(\Gamma) \text{)} \quad \text{(Fn)} \frac{\Phi \mid \Gamma, x : A \vdash v : B}{\Phi \mid \Gamma \vdash \lambda x : A. v : A \rightarrow B} \\ \text{(Sub)} & \frac{\Phi \mid \Gamma \vdash v : A \quad A \leq_\Phi B}{\Phi \mid \Gamma \vdash v : B} \quad \text{(Effect-Abs)} \frac{\Phi, \alpha \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \Lambda \alpha. v : \forall \alpha. A} \quad \text{(Effect-apply)} \frac{\Phi \mid \Gamma \vdash v : \forall \alpha. A \quad \Phi \vdash \epsilon}{\Phi \mid \Gamma \vdash v : A[\epsilon/\alpha]} \\ \text{(Return)} & \frac{\Phi \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \text{return } v : M_1 A} \quad \text{(Apply)} \frac{\Phi \mid \Gamma \vdash v_1 : A \rightarrow B \quad \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash v_1 v_2 : B} \\ \text{(If)} & \frac{\Phi \mid \Gamma \vdash v : \text{Bool} \quad \Phi \mid \Gamma \vdash v_1 : A \quad \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash \text{if } v \text{ then } v_1 \text{ else } v_2 : A} \quad \text{(Do)} \frac{\Phi \mid \Gamma \vdash v_1 : M_{\epsilon_1} A \quad \Phi \mid \Gamma, x : A \vdash v_2 : M_{\epsilon_2} B}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : M_{\epsilon_1 \cdot \epsilon_2} B} \end{aligned}$$

Ok Lemmas

Lemma 2.4.1 (Ok Lemma on Type Environments). *The first lemma used in this dissertation is: if $\Phi \mid \Gamma \vdash v : A$ then $\Phi \vdash \Gamma \text{ Ok}$.*

Proof: If $\Phi \vdash \Gamma, x : A \text{ Ok}$ then by the inversion (See figure 2.21) of the Ok rule, we have $\Phi \vdash \Gamma \text{ Ok}$

Only the type rule (*Weaken*) adds terms to the environment from its preconditions to its postcondition and it does so in an Ok preserving way. Any type derivation tree has at least one leaf. All leaves are axioms which require $\Phi \vdash \Gamma \text{ Ok}$. And all non-axiom derivations preserve the Ok property.

□

Lemma 2.4.2 (Ok Lemma on Effect Environments). *A second group of lemmas is that each of the judgments $\Phi \vdash \epsilon$, $\Phi \vdash A$, and $\Phi \vdash \Gamma \text{ Ok}$ implies $\Phi \text{ Ok}$.*

Inversion in Inductive Proofs

In proofs involving assumptions with multiple inductive rules, a useful property is that of inversion. This allows us to case split over the inductive rule that generated the assumption and can allow us to infer the structure of the term.

For example, consider a theorem which assumes $\Phi \mid \Gamma \vdash v : A$, then we can case split over the type rules. If $\Phi \mid \Gamma \vdash v : A$ was generated by the (Apply) rule (2.8), then by inspecting the rule, we can infer that there exist two subterms $\Phi \mid \Gamma \vdash v_1 : A \rightarrow B$, $\Phi \mid \Gamma \vdash v_2 : A$ such that $v = v_1 \ v_2$.

$$\text{(Apply)} \frac{\Phi \mid \Gamma \vdash v_1 : A \rightarrow B \quad \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash v_1 \ v_2 : B} \quad (2.8)$$

Then we might use the inferred subexpressions v_1, v_2 to prove the theorem in this case.

Figure 2.21: An explanation of the concept of inversion.

Proof: We shall prove each of the implications separately, but an over-arching theme of this proof is that the leaf rules of any $\Phi \vdash \text{Consequence}$ derivation tree require $\Phi \text{ Ok}$ and that any modifications to the effect environment in the assumptions to the effect environment in the consequence of a rule preserve the Ok relation. Hence, using induction and inversion, we can show that the effect environment is always well formed in any complete derivation tree.

Firstly in the case of the well-formedness relation $\Phi \vdash \epsilon$, the only leaf rules are the (*Ground*) and (*Var*) rules. By inversion, if $\Phi \vdash e$ or $\Phi, \alpha \vdash \alpha$ then $\Phi \text{ Ok}$ or $\Phi, \alpha \text{ Ok}$ respectively. Next, by inversion on the (*Monoid Op*) rule, if $\Phi \vdash \epsilon_1 \cdot \epsilon_2$ then $\Phi \vdash \epsilon_1$, so by induction $\Phi \text{ Ok}$. Finally, the weakening rule preserves the Ok property of the effect environment, so $\Phi, \beta \vdash \alpha \implies \Phi \vdash \alpha \implies \Phi \text{ Ok} \implies (\Phi, \beta) \text{ Ok}$.

Secondly, in the case of the well formedness relation on types, the only leaf rule is (*Ground*). If $\Phi \vdash \gamma$, then by inversion, $\Phi \text{ Ok}$. Similarly to the (*Monoid Op*) rule for effects, the (*Fn*) rule's preconditions have the same effect-environment, so $\Phi \text{ Ok}$ holds by inversion and induction. By inversion on the (*Effect*) rule, $\Phi \vdash \epsilon$ so $\Phi \text{ Ok}$. Finally the (*All*) rule preserves Ok in a similar way to the (*Weaken*) rule above.

Finally, if $\Phi \vdash \Gamma, x : A \text{ Ok}$ then, by inversion, $\Phi \vdash A$ and $\Phi \vdash \Gamma \text{ Ok}$. Since Γ is finite, $\Phi \vdash \diamond \text{ Ok}$. By inversion of the (*Nil*) rule, $\Phi \text{ Ok}$.

Chapter 3

The Semantics of PEC in an Strictly Indexed Category

In this chapter, I describe the category structure required to interpret an instance of the PEC. I then present denotations of each type of structure in the language, such as types, effects, terms, substitutions, and environment weakenings. Finally, I provide outlines and interesting cases of the proofs of the lemmas leading up to and including soundness of the semantics with respect to a $\beta\eta$ -conversion-based equational equivalence. However, firstly I shall give a brief treatment to the semantics of EC.

3.1 Semantics for EC in an S-Category

As suggested in section 2.2, since EC contains multiple effects, STLC terms, and if expressions, we should be able to interpret its semantics in a cartesian closed category with an appropriate strong graded monad and the co-product $1 + 1$. With the addition of some extra category structure to handle subtyping, which I shall explain shortly, it is indeed possible to interpret EC. This section contains a fairly high-level treatment of the semantics. This is because the concepts introduced are a subset of those required for the semantics of PEC, which I shall explain in more detail later. Semantics for similar languages to EC have been given using an S-category-style construct, such as by Katsumata [2].

Definition 3.1.1 (S-Category). In order to correctly model a given instantiation of EC, that is an EC with a collection of effects, constants, and ground types, there should be a collection of objects in the category to represent the ground types. There should also be a point morphism (a morphism from the terminal object to the appropriate type) for each constant. In addition to the previously stated requirements, we also require an S-category, \mathbb{C} , to be able to model subtyping and subeffecting. For each instance of the ground subtyping relation, $A \leq_\gamma B$, there should exist a morphism between the objects representing the ground types A and B . A further requirement is that \mathbb{C} has a collection of subeffecting natural transformations. For each instance of $\epsilon_1 \leq \epsilon_2$, there exists a natural transformation $\llbracket \epsilon_1 \leq \epsilon_2 \rrbracket : T_{\epsilon_1} \rightarrow T_{\epsilon_2}$ such that it has interactions with the graded monad as specified in figures 3.1, 3.2. We can now define morphisms for all forms of subtyping by constructions following the derivation tree for subtyping. For more detail, one can look at section 3.4 and below for the same construction on PEC. From this point onwards, I shall refer to a category that fulfills these properties of having a strong graded monad, CCC, ground objects and points, subeffect natural transformations, and a co-product $1 + 1$ as an *S-Category* (Semantic Category).

$$\begin{array}{ccc}
A \times T_{\epsilon_1} B & \xrightarrow{\text{Id}_{A \times \llbracket \epsilon_1 \leq \epsilon_2 \rrbracket_B}} & A \times T_{\epsilon_2} B \\
\downarrow \text{t}_{\epsilon_1, A, B} & & \downarrow \text{t}_{\epsilon_2, A, B} \\
T_{\epsilon_1}(A \times B) & \xrightarrow{\llbracket \epsilon_1 \leq \epsilon_2 \rrbracket_{A \times B}} & T_{\epsilon_2}(A \times B)
\end{array}$$

Figure 3.1: The interaction of the subeffect natural transformation with the tensor strength natural transformation.

$$\begin{array}{ccc}
T_{\epsilon_1} T_{\epsilon_2} & \xrightarrow{T_{\epsilon_1} \llbracket \epsilon_2 \leq \epsilon'_2 \rrbracket} & T_{\epsilon_1} T_{\epsilon'_2} \xrightarrow{\llbracket \epsilon_1 \leq \epsilon'_1 \rrbracket_{M, T_{\epsilon'_2}}} T_{\epsilon'_1} T_{\epsilon'_2} \\
\downarrow \mu_{\epsilon_1, \epsilon_2} & & \downarrow \mu_{\epsilon'_1, \epsilon'_2} \\
T_{\epsilon_1 \cdot \epsilon_2} & \xrightarrow{\llbracket \epsilon_1 \cdot \epsilon_2 \leq \epsilon'_1 \cdot \epsilon'_2 \rrbracket} & T_{\epsilon'_1 \cdot \epsilon'_2}
\end{array}$$

Figure 3.2: The interaction of the subeffect natural transformation with the graded monad bind natural transformation.

A full derivation and proof of soundness of the semantics of the Effect Calculus can be found online on my github repository¹ as it is too long to include here and many of the concepts are repeated later in the remainder of this dissertation anyway. The categorical semantics of the Effect Calculus requires an S-category not only to simply model all of the features of EC but also to use the various commutivity diagrams and rules to manipulate the expressions encountered when proving properties of the semantics. Again, as all these manipulations are also applied when proving the semantics of PEC, I shall not go into further detail here.

3.2 Required Category Structure

In order to model the polymorphism of PEC, we need to now look at an indexed category. This consists of a base category, \mathbb{C} in which we can interpret the possible effect environments in, and a mapping from objects in the base category to S-categories in the category of S-categories. This mapping is denoted from this point onwards as $\mathbb{C}(-)$ and the induced categories $\mathbb{C}(\llbracket \Phi \rrbracket)$ are called “fibres”. Furthermore, I shall shortly extend this mapping into a contravariant functor. The term “S-preserving” indicates that all functors derived from \mathbb{C} within this category preserve the properties of S-categories, which are explained in section 3.1. A functor F preserves the properties of S-categories if it preserves each of the features within an S-category. For example $F(A \times B) = (FA) \times (FB)$. For a full list of properties that an S-preserving functor should preserve, please see appendix **Reference Needed: (Link)**. To extend our mapping $\mathbb{C}(-)$ to a contravariant functor, it needs to map morphisms in the base category \mathbb{C} to functors between fibres. Thus, each morphism $\theta : \llbracket \Phi' \rrbracket \rightarrow \llbracket \Phi \rrbracket$ in \mathbb{C} should induce an S-preserving, re-indexing functor $\theta^* : \mathbb{C}(\llbracket \Phi \rrbracket) \rightarrow \mathbb{C}(\llbracket \Phi' \rrbracket)$ between the fibres.

The essential idea from this point on is that we have defined several relations of the form $\text{Env} \vdash \text{Conclusion}$, such as the typing relation $\Phi \mid \Gamma \vdash v : A$, and the well-formedness relation on effects $\Phi \vdash \epsilon$. Each instance of such a relation has a denotation that is an object or morphism in a category. For example, $\llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket$ is a morphism in the base category, $\llbracket \Phi \vdash A : \text{Type} \rrbracket$ is an object in the fibre (S-category) induced by Φ , and $\llbracket \Phi \mid \Gamma \vdash v : A \rrbracket$ is morphism between the objects which denote Γ and A in the fibre induced by Φ .

In order to form denotations of well formed effects, ϵ , we need specific objects to exist in the base category \mathbb{C} . Firstly, there should exist an object, U , indicating the kind of effects. To denote effect variable environments, which are lists of effect variables, we need finite products on U , that is \mathbb{C} should have a terminal object, 1 and binary products. We can form finite products as so: $U^0 = 1$ and $U^{n+1} = U^n \times U$. From now on, I use I to mean U^n for some n . Hence, the base category \mathbb{C} should contain 1 , U and finite products of U .

There is also a requirement that the indexed category can model ground effects, types, and terms. In order to do this, it should have a base-category morphism $\llbracket e \rrbracket : \mathbb{C}(1, U)$ for each ground effect e . Furthermore, each fibre should contain an object $\llbracket \gamma \rrbracket$ for each ground type γ . Finally, for each constant, \mathbb{C}^A , there should exist a morphism in each fibre: $\llbracket \mathbb{C}^A \rrbracket : 1 \rightarrow A$. These last two requirements are satisfied by the fibres all being S-categories.

¹<https://github.com/Al153/Part3Project/blob/master/diss/AppendixECSemantics.pdf>

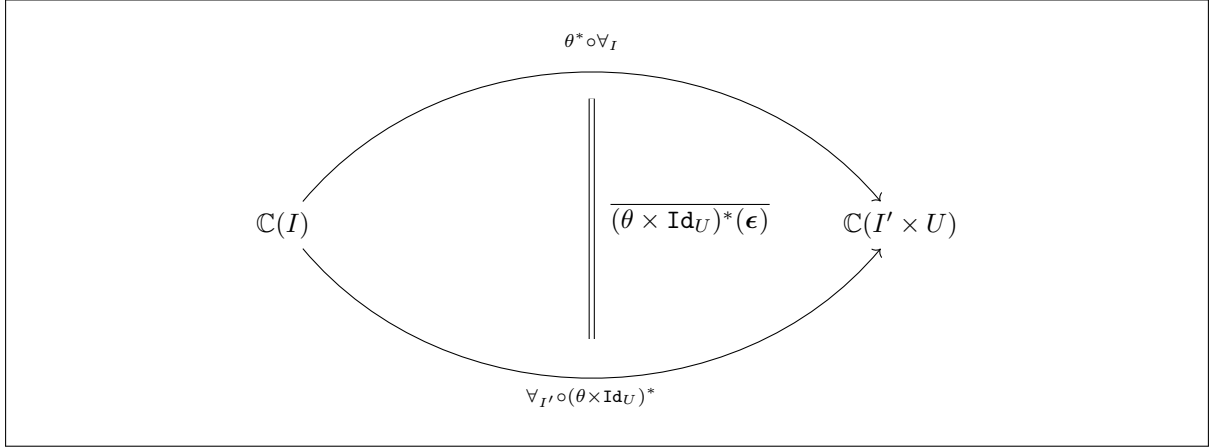


Figure 3.3: A functor diagram of the Beck-Chevalley condition

Next up, there needs to be a monoidal operator $\text{Mul}_I : \mathbb{C}(I, U) \times \mathbb{C}(I, U) \rightarrow \mathbb{C}(I, U)$. This means that Mul_I should be associative and have an identity element (which is $\llbracket 1 \rrbracket \circ \langle \rangle_I$). A corollary of this is that the hom-set $\mathbb{C}(I, U)$ forms a monoid. Furthermore Mul_I should be natural, which means: $\text{Mul}_{I'}(f, g) \circ \theta = \text{Mul}_I(f \circ \theta, g \circ \theta)$. Finally, Mul_1 should preserve the operation of the multiplication of ground effects. That is, $\text{Mul}_1(\llbracket e_1 \rrbracket, \llbracket e_2 \rrbracket) = \llbracket e_1 \cdot e_2 \rrbracket$ where e_1, e_2 are ground effects.

Our penultimate requirement is that the re-indexing functor induced by $\pi_1 : I \times U \rightarrow I$ (that is $\pi_1^* : \mathbb{C}(I) \rightarrow \mathbb{C}(I \times U)$) has a right adjoint, $\forall_I : \mathbb{C}(I \times U) \rightarrow \mathbb{C}(I)$. As the reader might be able to guess, this functor allows us to interpret quantification over effects. This right adjoint need not be S-preserving.

Finally, \forall_I should satisfy the Beck-Chevalley condition, as seen in figure 3.3. That is $\theta^* \circ \forall_I = \forall_{I'} \circ (\theta \times \text{Id}_U)^*$, and the natural transformation $(\theta \times \text{Id}_U)^*(\epsilon)$ between these functors is equal to the identity natural transformation. This allows us to commute the re-indexing functors with the quantification functor.

$$\overline{(\theta \times \text{Id}_U)^*(\epsilon)} = \text{Id} : \theta^* \circ \forall_I \rightarrow \forall_{I'} \circ (\theta \times \text{Id}_U)^* \in \mathbb{C}(I')$$

From the Beck-Chevalley condition, we can derive some more naturality conditions that will become useful later. Using the adjunction property, we have that: $f \circ \pi_1^*(n) = \bar{f} \circ n$. Secondly, using the Beck-Chevalley condition, we can derive some non-trivial interactions between re-indexing functors and the adjunction².

$$\begin{aligned} \theta^* \eta_A : \quad & \theta^* A \rightarrow \theta^* \circ \forall_I \circ \pi_1^* A \\ \theta^* \eta &= \overline{(\theta \times \text{Id}_U)^*(\epsilon_{\pi_1^*})} \circ \theta^* \eta \\ &= (\forall_{I'} \circ (\theta \times \text{Id}_U)^*)(\epsilon_{\pi_1^*}) \circ \eta_{(\forall_{I'} \circ (\theta \times \text{Id}_U)^*) \circ \pi_1^*} \circ \theta^* \eta \\ &= (\forall_{I'} \circ (\theta \times \text{Id}_U)^*)(\epsilon_{\pi_1^*}) \circ \eta_{\theta^* \circ \forall_I \circ \pi_1^*} \circ \theta^* \eta \\ &= (\forall_{I'} \circ (\theta \times \text{Id}_U)^*)(\epsilon_{\pi_1^*}) \circ (\theta^* \circ \forall_I \circ \pi_1^*) \eta \circ \eta_{(\theta \times \text{Id}_U)^*} \\ &= (\theta^* \circ \forall_I)(\epsilon_{\pi_1^*} \circ \pi_1^* \eta) \circ \eta_{(\theta \times \text{Id}_U)^*} \\ &= (\theta^* \circ \forall_I)(\text{Id}) \circ \eta_{(\theta \times \text{Id}_U)^*} \\ &= \eta_{(\theta \times \text{Id}_U)^*} \end{aligned}$$

²This part of the proof comes from a stack-overflow answer: <https://math.stackexchange.com/questions/188099/beck-chevalley-condition-and-maps-of-adjunctions>

Importantly, this gives us the useful naturality condition that allows us to push re-indexing functors into the adjunction terms.

$$\theta^*(\bar{f}) = \theta^*(\forall_I(f) \circ \eta_A) \quad (3.1)$$

$$= \theta^*(\forall_I(f)) \circ \theta^*(\eta_A) \quad (3.2)$$

$$= (\forall_{I'} \circ (\theta \times \text{Id}_U)^*)f \circ \eta_{(\theta \times \text{Id}_U)^*A} \quad (3.3)$$

$$= \overline{(\theta \times \text{Id}_U)^*f} \quad (3.4)$$

$$(3.5)$$

3.3 Road Map

In figure 3.4, one can see a diagram of the collection of theorems that need to be proved to establish the soundness of a semantics for PEC.

The first pair of theorems is made up of the effect substitution and weakening theorem on effects. These theorems show that substitutions of effects have a 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. From this, we can also look at the action of weakening and substituting effect environments on the subtyping between types.

The next step is to use these substitution theorems to formalise the action of substitution and weakening of the effect environments on terms. This then allows us to find denotations for the weakening of term substitutions and type environment weakening, which set us up to prove the typical weakening and substitution theorems upon term variables and type environments.

Separately, we prove that all derivable denotations for a typing relation instance, $\Phi \mid \Gamma \vdash v : A$ have the same denotation. This is important, since subtyping 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 prove that all typing derivations yield equal denotations.

This collection of theorems finally allows us to complete all cases of the equational-equivalence soundness theorem.

3.4 Denotations

We are now equipped to define the denotations of structures in the language. Firstly, we shall define the denotation of effect environments. In the base category, with the kind of effects indicated by U , the denotation of an effect environment Φ is given by a finite product.

$$\llbracket \diamond \rrbracket = 1 \quad \llbracket \Phi, \alpha \rrbracket = \llbracket \Phi \rrbracket \times U$$

Hence, the denotation of an effect-variable environment Φ is given by a finite product on U of width n where n is the length of Φ . In order to de-clutter the notation used going forwards, inheriting from R. L. Crole ([7]), I shall write I as shorthand for $\llbracket \Phi \rrbracket$.

As stated earlier, the denotation of a well-formed effect is a morphism $\llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket : I \rightarrow U$ in \mathbb{C} .

$$\llbracket \Phi \vdash e : \text{Effect} \rrbracket = \llbracket \epsilon \rrbracket \circ \langle \rangle_I : I \rightarrow U \quad \llbracket \Phi, \alpha \vdash \alpha : \text{Effect} \rrbracket = \pi_2 : I \times U \rightarrow U$$

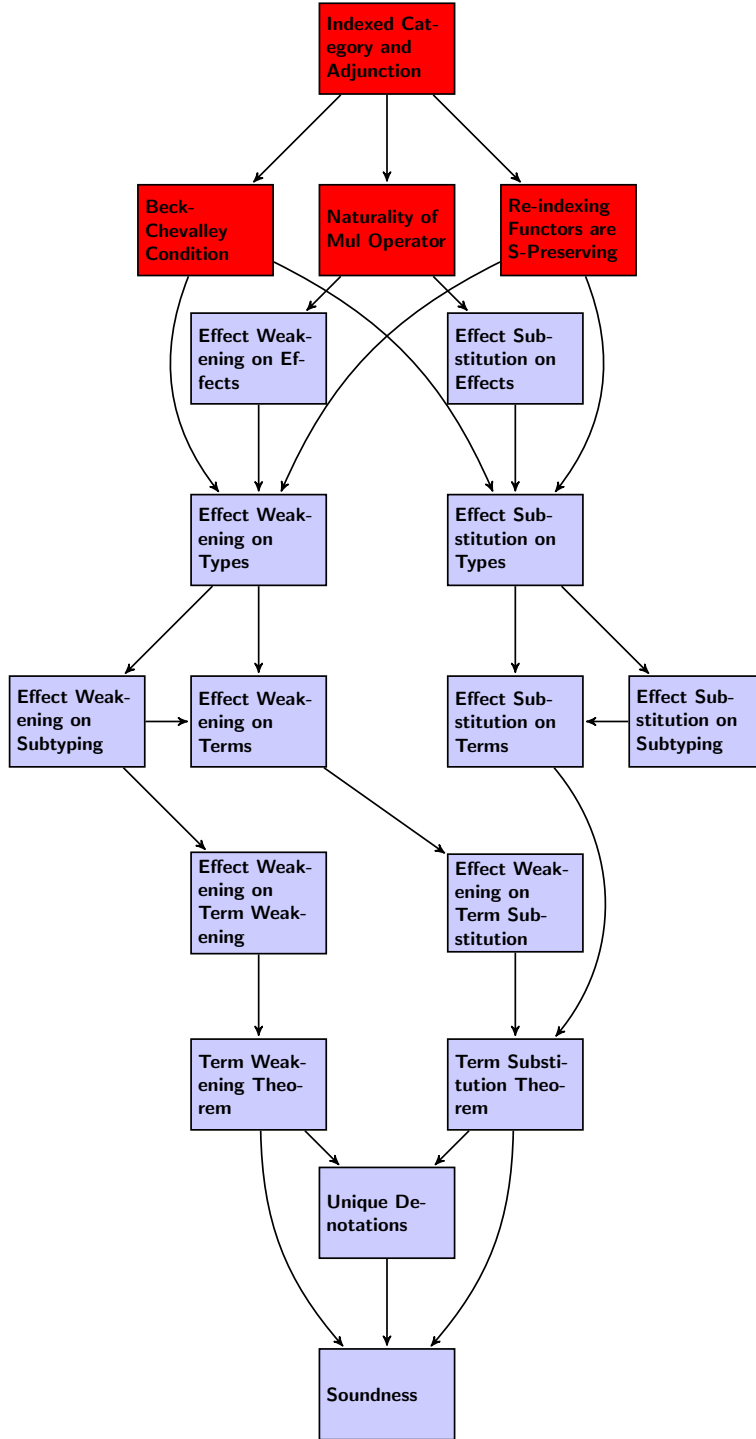


Figure 3.4: A road map of the proof dependencies. Assumptions in red, theorems in blue

$$\begin{aligned}
 \llbracket \Phi, \beta \vdash \alpha : \text{Effect} \rrbracket &= \llbracket \Phi \vdash \alpha : \text{Effect} \rrbracket \circ \pi_1 : I \times U \rightarrow U \\
 \llbracket \Phi \vdash \epsilon_1 \cdot \epsilon_2 : \text{Effect} \rrbracket &= \text{Mul}_I(\llbracket \Phi \vdash \epsilon_2 : \text{Effect} \rrbracket, \llbracket \Phi \vdash \epsilon_1 : \text{Effect} \rrbracket) : I \rightarrow U
 \end{aligned}$$

Using these denotations, we are now equipped to define the denotations of types. As stated above, types that are well formed in Φ are denoted by objects in the fibre category $\mathbb{C}(I)$ given by the denotation of Φ .

Since the fibre category $\mathbb{C}(I)$ is an S-category, it has objects for all ground types, a terminal object, graded monad T , exponentials, products, and co-product over $1 + 1$.

$$\llbracket \Phi \vdash \text{Unit} : \text{Type} \rrbracket = 1 \quad \llbracket \Phi \vdash \text{Bool} : \text{Type} \rrbracket = 1 + 1 \quad \llbracket \Phi \vdash \gamma : \text{Type} \rrbracket = \llbracket \gamma \rrbracket$$

$$\llbracket \Phi \vdash A \rightarrow B : \text{Type} \rrbracket = (\llbracket \Phi \vdash B : \text{Type} \rrbracket)^{(\llbracket \Phi \vdash A : \text{Type} \rrbracket)}$$

$$\llbracket \Phi \vdash M_\epsilon A : \text{Type} \rrbracket = T_{\llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket} \llbracket \Phi \vdash A : \text{Type} \rrbracket \quad \llbracket \Phi \vdash \forall \alpha. A : \text{Type} \rrbracket = \forall_I (\llbracket \Phi, \alpha \vdash A : \text{Type} \rrbracket)$$

By using the terminal objects and products present in each fibre, we can now derive denotations of type environments. $\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket$ should be an object in the fibre induced by Φ , $\mathbb{C}(I)$.

$$\llbracket \Phi \vdash \diamond \text{ Ok} \rrbracket = 1 \quad \llbracket \Phi \vdash \Gamma, x : A \text{ Ok} \rrbracket = (\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket \times \llbracket \Phi \vdash A : \text{Type} \rrbracket)$$

Another construction that is important is the denotation of subtyping. For each instance of the subtyping relation in Φ , $A \leq_\Phi B$, there exists a denotation in the fibre induced by Φ . $\llbracket A \leq_\Phi B \rrbracket \in \mathbb{C}(I)(A, B)$. Since the fibres are S-categories, the ground instances of the subtyping relation exist in each fibre anyway. In addition, for each instance of the subeffect relation $\epsilon_1 \leq_\Phi \epsilon_2$ each fibre contains the natural transformation $\llbracket \epsilon_1 \leq_\Phi \epsilon_2 \rrbracket : T_{\epsilon_1} \rightarrow T_{\epsilon_2}$

$$\llbracket \gamma_1 \leq_\Phi \gamma_2 \rrbracket = \llbracket \gamma_1 \leq_\gamma \gamma_2 \rrbracket \quad \llbracket A \rightarrow B \leq_\Phi A' \rightarrow B' \rrbracket = \llbracket B \leq_\Phi B' \rrbracket^{A'} \circ B^{\llbracket A' \leq_\Phi A \rrbracket}$$

$$\llbracket M_{\epsilon_1} A \leq_\Phi M_{\epsilon_2} B \rrbracket = \llbracket \epsilon_1 \leq_\Phi \epsilon_2 \rrbracket \circ T_{\epsilon_1} \llbracket A \leq_\Phi B \rrbracket \quad \llbracket \forall \alpha. A \leq_\Phi \forall \alpha. B \rrbracket = \forall_I \llbracket A \leq_{\Phi, \alpha} B \rrbracket$$

This finally gives us the ability to express the denotations of well-typed terms in an effect environment, Φ as morphisms in the fibre induced by Φ , $\mathbb{C}(I)$. Writing Γ_I and A_I for $\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket$ and $\llbracket \Phi \vdash A : \text{Type} \rrbracket$, we can derive $\llbracket \Phi \mid \Gamma \vdash v : A \rrbracket$ as a morphism in $\mathbb{C}(I)(\Gamma_I, A_I)$. Since each fibre is an S-category, for each ground constant, \mathbb{C}^A , there exists $\llbracket \mathbb{C}^A \rrbracket : 1 \rightarrow A_I$ in $\mathbb{C}(I)$. These denotations can be seen in figure 3.5

The least intuitive of these denotations is that of effect application (specialisation) in equation 3.6. As explained later in section 3.5 the re-indexing functor represents application of the substitution (**TODO: I don't think this is circular logic, but would like another pair of eyes if possible**) $[\epsilon/\beta]$. By doing type analysis on the co-unit morphism $\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}$, as in figure 3.6 we can see that before the application of the substitution, the co-unit natural transformation takes a quantified type to a substituted type under an extended effect environment. By applying the substitution re-indexing functor in figure 3.7, we substitute ϵ for the extra effect variable, β . In the case of the quantified type, as β is not used, the substitution has no effect, but in the case of the substituted type, the substitutions compose to yield the substitution $[\epsilon/\alpha]$. This composes with the expression denotation g to give the applied denotation in figure 3.8.

3.5 Substitution and Weakening Theorems

In this section, I introduce and prove a series of utility theorems, which will help us prove cases in future theorems. These weakening and substitution theorems are concerned with a change in environment of

$$\begin{array}{c}
\text{(Unit)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\llbracket \Phi \mid \Gamma \vdash () : \mathbf{Unit} \rrbracket = \langle \rangle_{\Gamma} : \Gamma_I \rightarrow 1} \quad \text{(Const)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\llbracket \Phi \mid \Gamma \vdash \mathbf{c}^A : A \rrbracket = \llbracket \mathbf{c}^A \rrbracket \circ \langle \rangle_{\Gamma} : \Gamma \rightarrow \llbracket A \rrbracket} \\
\\
\text{(True)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\llbracket \Phi \mid \Gamma \vdash \mathbf{true} : \mathbf{Bool} \rrbracket = \mathbf{inl} \circ \langle \rangle_{\Gamma} : \Gamma \rightarrow \llbracket \mathbf{Bool} \rrbracket = 1 + 1} \\
\\
\text{(False)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\llbracket \Phi \mid \Gamma \vdash \mathbf{false} : \mathbf{Bool} \rrbracket = \mathbf{inr} \circ \langle \rangle_{\Gamma} : \Gamma \rightarrow \llbracket \mathbf{Bool} \rrbracket = 1 + 1} \\
\\
\text{(Var)} \frac{\Phi \vdash \Gamma \text{ Ok}}{\llbracket \Phi \mid \Gamma, x : A \vdash x : A \rrbracket = \pi_2 : \Gamma \times A \rightarrow A} \quad \text{(Weaken)} \frac{f = \llbracket \Phi \mid \Gamma \vdash x : A \rrbracket : \Gamma \rightarrow A}{\llbracket \Phi \mid \Gamma, y : B \vdash x : A \rrbracket = f \circ \pi_1 : \Gamma \times B \rightarrow A} \\
\\
\text{(Lambda)} \frac{f = \llbracket \Phi \mid \Gamma, x : A \vdash v : B \rrbracket : \Gamma \times A \rightarrow B}{\llbracket \Phi \mid \Gamma \vdash \lambda x : A. v : A \rightarrow B \rrbracket = \mathbf{cur}(f) : \Gamma \rightarrow (B)^A} \\
\\
\text{(Subtype)} \frac{f = \llbracket \Phi \mid \Gamma \vdash v : A \rrbracket : \Gamma \rightarrow A \quad g = \llbracket A \leq_{\Phi} B \rrbracket}{\llbracket \Phi \mid \Gamma \vdash v : B \rrbracket = g \circ f : \Gamma \rightarrow B} \quad \text{(Return)} \frac{f = \llbracket \Phi \mid \Gamma \vdash v : A \rrbracket}{\llbracket \Phi \mid \Gamma \vdash \mathbf{return} v : \mathbf{M}_1 A \rrbracket = \eta_A \circ f} \\
\\
\text{(If)} \frac{f = \llbracket \Phi \mid \Gamma \vdash v : \mathbf{Bool} \rrbracket : \Gamma \rightarrow 1 + 1 \quad g = \llbracket \Phi \mid \Gamma \vdash v_1 : \mathbf{M}_{\epsilon} A \rrbracket \quad h = \llbracket \Phi \mid \Gamma \vdash v_2 : \mathbf{M}_{\epsilon} A \rrbracket}{\llbracket \Phi \mid \Gamma \vdash \mathbf{if}_{\epsilon, A} v \text{ then } v_1 \text{ else } v_2 : \mathbf{M}_{\epsilon} A \rrbracket = \mathbf{app} \circ ((\llbracket \mathbf{cur}(g \circ \pi_2), \mathbf{cur}(h \circ \pi_2) \rrbracket \circ f) \times \mathbf{Id}_{\Gamma}) \circ \delta_{\Gamma} : \Gamma \rightarrow T_{\epsilon} A} \\
\\
\text{(Bind)} \frac{f = \llbracket \Phi \mid \Gamma \vdash v_1 : \mathbf{M}_{\epsilon_1} A : \Gamma \rightarrow T_{\epsilon_1} A \rrbracket \quad g = \llbracket \Phi \mid \Gamma, x : A \vdash v_2 : \mathbf{M}_{\epsilon_2} B \rrbracket : \Gamma \times A \rightarrow T_{\epsilon_2} B}{\llbracket \Phi \mid \Gamma \vdash \mathbf{do} x \leftarrow v_1 \text{ in } v_2 : \mathbf{M}_{\epsilon_1, \epsilon_2} B \rrbracket = \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} g \circ \mathbf{t}_{\Gamma, A, \epsilon_1} \circ \langle \mathbf{Id}_{\Gamma}, f \rangle : \Gamma \rightarrow T_{\epsilon_1, \epsilon_2} B} \\
\\
\text{(Apply)} \frac{f = \llbracket \Phi \mid \Gamma \vdash v_1 : A \rightarrow B \rrbracket : \Gamma \rightarrow (B)^A \quad g = \llbracket \Phi \mid \Gamma \vdash v_2 : A \rrbracket : \Gamma \rightarrow A}{\llbracket \Phi \mid \Gamma \vdash v_1 v_2 : B \rrbracket = \mathbf{app} \circ \langle f, g \rangle : \Gamma \rightarrow B} \\
\\
\text{(Effect-Lambda)} \frac{f = \llbracket \Phi, \alpha \mid \Gamma \vdash v : A \rrbracket : \mathbb{C}(I \times U, W)(\Gamma, A)}{\llbracket \Phi \mid \Gamma \vdash \Lambda \alpha. A : \forall \epsilon. A \rrbracket = \bar{f} : \mathbb{C}(I)(\Gamma, \forall_I(A))} \\
\\
\text{(Effect-App)} \frac{g = \llbracket \Phi \mid \Gamma \vdash v : \forall \alpha. A \rrbracket : \mathbb{C}(I)(\Gamma, \forall_I(A)) \quad h = \llbracket \Phi \vdash \epsilon : \mathbf{Effect} \rrbracket : \mathbb{C}(I, U)}{\llbracket \Phi \mid \Gamma \vdash v \epsilon : A[\epsilon/\alpha] \rrbracket = \langle \mathbf{Id}_I, h \rangle^* (\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket}) \circ g : \mathbb{C}(I)(\Gamma, A[\epsilon/\alpha])} \quad (3.6)
\end{array}$$

Figure 3.5: The denotations of terms in PEC

$$\begin{aligned}
\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket} &= \overline{\text{Id}_{\forall_I}(\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket)} \\
&: \pi_1^* \forall_I(\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket) \rightarrow \llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket \\
&: \pi_1^* \llbracket \Phi \vdash \forall \beta. A[\beta/\alpha] : \text{Type} \rrbracket \rightarrow \llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket \\
&: \pi_1^* \llbracket \Phi \vdash \forall \alpha. A : \text{Type} \rrbracket \rightarrow \llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket \\
&: \llbracket \Phi, \beta \vdash \forall \alpha. A : \text{Type} \rrbracket \rightarrow \llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket
\end{aligned}$$

Figure 3.6: Analysis of the type of the co-unit natural transformation

$$\begin{aligned}
\langle \text{Id}_I, h \rangle^*(\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}) &: \llbracket \Phi \vdash \forall \alpha. A : \text{Type} \rrbracket \rightarrow \llbracket \Phi \vdash A[\beta/\alpha] [\epsilon/\beta] : \text{Type} \rrbracket \\
&: \llbracket \Phi \vdash \forall \alpha. A : \text{Type} \rrbracket \rightarrow \llbracket \Phi \vdash A[\epsilon/\alpha] : \text{Type} \rrbracket
\end{aligned}$$

Figure 3.7: Analysis of the type of the re-indexed co-unit

$$\begin{array}{ccc}
\llbracket \Phi, \beta \vdash \forall \alpha. A : \text{Type} \rrbracket & \xrightarrow{\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}} & \llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket \\
\Gamma \xrightarrow{g} \forall_I(A) & \xrightarrow{\langle \text{Id}_I, h \rangle^*(\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket})} & A[\epsilon/\alpha]
\end{array}$$

Figure 3.8: Composition of the effect specialisation denotation

typing derivations and their associated denotations. If $\Phi \mid \Gamma \vdash v : A$, then it should be the case that $\Phi \mid \Gamma, x : A \vdash v : A$ if x is fresh in Γ . We also want to know what happens to the denotation when we change the type environment in this way. In this section, I introduce the tools for manipulating the type and effect environments in this way.

Substitutions and weakenings are two distinct ways of manipulating an effect or typing environment. Weakening acts as a kind of subtyping of the environment. If we insert fresh variables into an environment, then any expression that was typeable under the previous environment should also be typeable under the new environment. This change of environment should also have a predictable effect on the denotations of any expressions to which it is applied.

Substitution considers what happens when we simultaneously replace all variables in one expression, that is typeable under an environment, with expressions that are well formed under another environment. The resulting substituted expression should be typeable under the new environment, and the denotation of the new expression should be composed from the denotation of the old relation and the denotations of the expressions that replace the variables.

As we go on, I shall define and state the denotations of specific substitutions and weakenings, upon both the effect environment and the typing environment.

In this dissertation, substitutions and weakenings come in two flavours: weakening and substitution of the effect-variable environment and weakening and substitution of the typing environment. For each of these there is a family of theorems defining the effects of the applying a substitution and weakening to the various language structures and their denotations, such as well-formedness and typing relations.

The first family of theorems is that of weakening and substitution of the effect environment. Weakenings are a relation between effect environments $\omega : \Phi' \triangleright \Phi$, that are defined as so:

$$\text{(Id)} \frac{\Phi \text{ Ok}}{\iota : \Phi \triangleright \Phi} \quad \text{(Project)} \frac{\omega : \Phi' \triangleright \Phi}{\omega\pi : (\Phi', \alpha) \triangleright \Phi} \text{ (if } \alpha \notin \Phi') \quad \text{(Extend)} \frac{\omega : \Phi' \triangleright \Phi}{\omega \times : (\Phi', \alpha) \triangleright (\Phi, \alpha)} \text{ (if } \alpha \notin \Phi')$$

We can define the denotation of an effect-environment weakening as a morphism in the base category: $\llbracket \omega : \Phi' \triangleright \Phi \rrbracket : I' \rightarrow I$. The denotations are defined as follows:

$$\llbracket \iota : \Phi \triangleright \Phi \rrbracket = \text{Id}_I : I \rightarrow I \quad \llbracket \omega\pi : \Phi', \alpha \triangleright \Phi \rrbracket = \llbracket \omega : \Phi' \triangleright \Phi \rrbracket \circ \pi_1 : I' \times U \rightarrow I$$

$$\llbracket \omega \times : \Phi', \alpha \triangleright \Phi, \alpha \rrbracket = (\llbracket \omega : \Phi' \triangleright \Phi \rrbracket \times \text{Id}_U) : I' \times U \rightarrow I \times U$$

Substitutions are also an inductively defined relation between effect environments, with inductively defined denotations. Substitutions may be represented as a snoc-list of variable-effect pairs.

$$\sigma :: = \diamond \mid \sigma, \alpha : = \epsilon$$

The substitution relation matches substitutions to the environments it operates between. The relation instance $\Phi' \vdash \sigma : \Phi$ means that σ is a substitution from Φ' to Φ . It is defined inductively using the following rules.

$$\text{(Nil)} \frac{\Phi' \text{ Ok}}{\Phi' \vdash \diamond : \Phi} \quad \text{(Extend)} \frac{\Phi' \vdash \sigma : \Phi \quad \Phi' \vdash \epsilon}{\Phi' \vdash \sigma, \alpha : = \epsilon : (\Phi, \alpha)} \text{ (if } \alpha \notin \Phi)$$

The denotation of an effect-environment substitutions, $\llbracket \Phi' \vdash \sigma : \Phi \rrbracket$, is a morphism $I' \rightarrow I$ in the base category. These denotations are defined inductively as so:

$$\llbracket \Phi' \vdash \diamond : \diamond \rrbracket = \langle \rangle_I : \mathbb{C}(I', 1) \quad \llbracket \Phi' \vdash (\sigma, \alpha : \epsilon) : \Phi, \alpha \rrbracket = \langle \llbracket \Phi' \vdash \sigma : \Phi \rrbracket, \llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket \rangle : \mathbb{C}(I', I \times U)$$

We can also define the action of substitutions on effects.

$$\begin{aligned} \sigma(e) &= e \\ \sigma(\epsilon_1 \cdot \epsilon_2) &= (\sigma(\epsilon_1)) \cdot (\sigma(\epsilon_2)) \\ \diamond(\alpha) &= \alpha \\ (\sigma, \beta : \epsilon)(\alpha) &= \sigma(\alpha) \\ (\sigma, \alpha : \epsilon)(\alpha) &= \epsilon \end{aligned}$$

The general construction for effect-variable substitution allows us to define, in particular, the identity substitution, $\Phi \vdash \text{Id}_\Phi : \Phi$ and the single substitution, $\Phi \vdash [\epsilon/\alpha] : \Phi, \alpha$.

$$\text{Id}_\diamond = \diamond \tag{3.7}$$

$$\text{Id}_{\Phi, \alpha} = (\text{Id}_\Phi, \alpha : \alpha) \tag{3.8}$$

$$[\epsilon/\alpha] = (\text{Id}_\Phi, \alpha : \epsilon) \tag{3.9}$$

By inspection, we can find the denotations of these special-case substitutions to be:

$$\llbracket \Phi \vdash \text{Id}_\Phi : \Phi \rrbracket = \text{Id}_I \tag{3.10}$$

$$\llbracket \Phi \vdash [\epsilon/\alpha] : (\Phi, \alpha) \rrbracket = \langle \text{Id}_I, \llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket \rangle \tag{3.11}$$

These substitutions will form a useful shorthand later. We also need to define the property $\alpha \# \sigma$. This indicates that α does not occur in the domain or any of the substitution expression of σ . Finally, it will be useful to examine how the substitutions generated by extending the effect environment, such as in the case of polymorphic types, relate to the original substitution.

Theorem 3.5.1 (Effect Substitution on Effects). *The substitution theorem on effects is the proposition that if $\Phi \vdash \epsilon$ and $\Phi' \vdash \sigma : \Phi$ then $\Phi' \vdash \sigma(\epsilon)$ and, writing σ for $\llbracket \Phi' \vdash \sigma : \Phi \rrbracket$, $\llbracket \Phi' \vdash \sigma(\epsilon) \rrbracket = \llbracket \Phi \vdash \epsilon \rrbracket \circ \sigma$.*

Proof: The proof of this depends on the naturality of Mul_I and inversion to narrow down case splitting on the structure of the effect environments.

Case Ground: This case holds due to the terminal morphism being unique. Hence, pre-composing it with another morphism yields the terminal morphism.

$$\begin{aligned} \llbracket \Phi \vdash e : \text{Effect} \rrbracket \circ \sigma &= \llbracket e \rrbracket \circ \langle \rangle_I \circ \sigma \\ &= \llbracket e \rrbracket \circ \langle \rangle_{I'} \\ &= \llbracket \Phi' \vdash e : \text{Type} \rrbracket \end{aligned}$$

Case Var: Since the structure of the substitution σ depends on the structure of Φ , we can perform inversion to infer the denotation of σ .

$$\begin{aligned} \llbracket \Phi, \alpha \vdash \alpha: \text{Effect} \rrbracket \circ \sigma &= \pi_2 \circ \langle \sigma', \llbracket \Phi' \vdash \epsilon: \text{Effect} \rrbracket \rangle \quad \text{By inversion } \sigma = (\sigma', \alpha := \epsilon) \\ &= \llbracket \Phi' \vdash \epsilon: \text{Effect} \rrbracket \\ &= \llbracket \Phi' \vdash \sigma'(\alpha): \text{Effect} \rrbracket \end{aligned}$$

Case Multiply: We make use of the naturality of Mul_I and induction upon the subterms.

$$\begin{aligned} \llbracket \Phi \vdash \epsilon_1 \cdot \epsilon_2: \text{Effect} \rrbracket \circ \sigma &= \text{Mul}_I(\llbracket \Phi \vdash \epsilon_1: \text{Effect} \rrbracket, \llbracket \Phi \vdash \epsilon_2: \text{Effect} \rrbracket) \circ \sigma \\ &= \text{Mul}_I(\llbracket \Phi \vdash \epsilon_1: \text{Effect} \rrbracket \circ \sigma, \llbracket \Phi \vdash \epsilon_2: \text{Effect} \rrbracket \circ \sigma) \quad \text{By Naturality} \\ &= \text{Mul}_{I'}(\llbracket \Phi' \vdash \sigma(\epsilon_1): \text{Effect} \rrbracket, \llbracket \Phi \vdash \sigma(\epsilon_2): \text{Effect} \rrbracket) \\ &= \llbracket \Phi' \vdash \sigma(\epsilon_1) \cdot \sigma(\epsilon_2): \text{Effect} \rrbracket \\ &= \llbracket \Phi' \vdash \sigma(\epsilon_1 \cdot \epsilon_2): \text{Effect} \rrbracket \end{aligned}$$

□

Theorem 3.5.2 (Effect Weakening on Effects). *The weakening theorem proceeds similarly. If $\Phi \vdash \epsilon$ and $\omega: \Phi' \triangleright \Phi$ then $\Phi' \vdash \omega$ and, writing ω for $\llbracket \omega: \Phi' \triangleright \Phi \rrbracket$, $\llbracket \Phi' \vdash \epsilon \rrbracket = \llbracket \Phi \vdash \epsilon \rrbracket \circ \omega$.*

Proof: This proof also depends on the naturality of Mul_I and case splitting on the structure of ω . It is harder to use inversion on the structure of ω , since the structure of ω does not depend as strongly on the structure of Φ . I present here the cases for variables.

Case Var: We do a case split on ω .

Case: $\omega = \iota$ Then $\Phi' = \Phi$ and $\omega = \text{Id}_I$. So the theorem holds trivially.

Case: $\omega = \omega' \times$ Then by the definition of its denotation:

$$\begin{aligned} \llbracket \Phi, \alpha \vdash \alpha: \text{Effect} \rrbracket \circ \omega &= \pi_2 \circ (\omega' \times \text{Id}_U) \\ &= \pi_2 \\ &= \llbracket \Phi', \alpha \vdash \alpha: \text{Effect} \rrbracket \end{aligned}$$

Case: $\omega = \omega' \pi$ Then

$$\llbracket \Phi, \alpha \vdash \alpha: \text{Effect} \rrbracket = \pi_2 \circ \omega' \circ \pi_1 \tag{3.12}$$

Where $\Phi' = \Phi, \beta$ and $\omega': \Phi'' \triangleright \Phi$.

So

$$\begin{aligned} \pi_2 \circ \omega' &= \llbracket \Phi'' \vdash \alpha: \text{Effect} \rrbracket \\ \pi_2 \circ \omega' \circ \pi_1 &= \llbracket \Phi'', \beta \vdash \alpha: \text{Effect} \rrbracket \\ &= \llbracket \Phi' \vdash \alpha: \text{Effect} \rrbracket \end{aligned}$$

Case Weaken:

$$\llbracket \Phi, \beta \vdash \alpha : \text{Effect} \rrbracket \circ \omega = \llbracket \Phi \vdash \alpha : \text{Effect} \rrbracket \circ \pi_1 \circ \omega \quad (3.13)$$

Similarly, we perform a case split on structure of ω :

Case: $\omega = \iota$ Then $\Phi' = \Phi, \beta$ so $\omega = \text{Id}_I$ So $\llbracket \Phi, \beta \vdash \alpha : \text{Effect} \rrbracket \circ \omega = \llbracket \Phi' \vdash \alpha : \text{Effect} \rrbracket$

Case: $\omega = \omega' \pi_1$ Then $\Phi' = (\Phi'', \gamma)$ and $\omega = \omega' \circ \pi_1$ Where $\omega' : \Phi'' \triangleright \Phi, \beta$. So

$$\begin{aligned} \llbracket \Phi, \beta \vdash \alpha : \text{Effect} \rrbracket \circ \omega &= \llbracket \Phi, \beta \vdash \alpha : \text{Effect} \rrbracket \circ \omega' \circ \pi_1 \\ &= \llbracket \Phi'' \vdash \alpha : \text{Effect} \rrbracket \circ \pi_1 \\ &= \llbracket \Phi'', \gamma \vdash \alpha : \text{Effect} \rrbracket \\ &= \llbracket \Phi' \vdash \alpha : \text{Effect} \rrbracket \end{aligned}$$

Case: $\omega = \omega' \times$ Then $\Phi' = \Phi'', \beta$ and $\omega' : \Phi'' \triangleright \Phi$

So

$$\begin{aligned} \llbracket \Phi, \beta \vdash \alpha : \text{Effect} \rrbracket \circ \omega &= \llbracket \Phi \vdash \alpha : \text{Effect} \rrbracket \circ \pi_1 \circ (\omega' \times \text{Id}_U) \\ &= \llbracket \Phi \vdash \alpha : \text{Effect} \rrbracket \circ \omega' \circ \pi_1 \\ &= \llbracket \Phi'' \vdash \alpha : \text{Effect} \rrbracket \circ \pi_1 \\ &= \llbracket \Phi' \vdash \alpha : \text{Effect} \rrbracket \end{aligned}$$

□

Lemma 3.5.3 (Extension Lemma on Effect Substitutions). *If $\Phi' \vdash \sigma : \Phi$, and $\alpha \# \sigma$, then $(\Phi', \alpha) \vdash (\sigma, \alpha = \alpha) : (\Phi, \alpha)$ with denotation $\llbracket (\Phi', \alpha) \vdash (\sigma, \alpha = \alpha) : (\Phi, \alpha) \rrbracket = (\llbracket \Phi' \vdash \sigma : \Phi \rrbracket \times \text{Id}_U)$*

Proof: This holds by the weakening-theorem on effects, theorem 3.5.2

□

We can then move on to state and prove the weakening and substitution theorems on types, subtyping, and type environments. The general structure of these theorems, as well as the term-based theorems later, is that when we want to quantify the effect of a morphism $\theta : I' \rightarrow I$ between objects in the base category on structure in the fibres $\mathbb{C}(I)$, we should simply apply the associated re-indexing functor $\theta^* : \mathbb{C}(I) \rightarrow \mathbb{C}(I')$ to the structure. The proof of the soundness of this operation is driven by the fact that the re-indexing functor is S-preserving.

Specifically, effect substitutions have the following actions on types, and type-environments:

$$\begin{aligned} \gamma[\sigma] &= \gamma \\ (A \rightarrow B)[\sigma] &= (A[\sigma]) \rightarrow (B[\sigma]) \\ (\mathbb{M}_\epsilon A)[\sigma] &= \mathbb{M}_{\sigma(\epsilon)}(A[\sigma]) \\ (\forall \alpha. A)[\sigma] &= \forall \alpha. (A[\sigma]) \quad \text{If } \alpha \# \sigma \end{aligned}$$

$$\begin{aligned}\diamond[\sigma] &= \diamond \\ (\Gamma, x:A)[\sigma] &= (\Gamma[\sigma], x:(A[\sigma]))\end{aligned}$$

Theorem 3.5.4 (Effect Substitution on Types). *The specific effect substitution theorem on types is if $\Phi \vdash A$ and $\Phi' \vdash \sigma: \Phi$, then $\Phi' \vdash A[\sigma]$ and $\llbracket \Phi' \vdash A[\sigma] \rrbracket = \sigma^* \llbracket \Phi \vdash A \rrbracket$.*

Proof: By the fact that σ^* is S-preserving and the Beck-Chevalley Condition.

Case Monad: This case makes use of the fact that σ^* is S-preserving. Specifically the property upon the graded monad functor T : $\sigma^*(T_f A) = T_{f \circ \sigma}(\sigma^* A)$.

$$\begin{aligned}\sigma^* \llbracket \Phi \vdash M_\epsilon A : \text{Type} \rrbracket &= \sigma^*(T_{\llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket} \llbracket \Phi \vdash A : \text{Type} \rrbracket) \\ &= T_{\llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket \circ \sigma} \sigma^*(\llbracket \Phi \vdash A : \text{Type} \rrbracket) \\ &= \llbracket \Phi' \vdash (M_\epsilon A)[\sigma] : \text{Type} \rrbracket\end{aligned}$$

Case Quantification: This case makes use of the Beck-Chevalley condition and the fact that $\llbracket (\Phi', \alpha) \vdash (\sigma, \alpha: \alpha): (\Phi, \alpha) \rrbracket = \sigma \times \text{Id}_U$, which we can induct upon.

$$\begin{aligned}\sigma^* \llbracket \Phi \vdash \forall \alpha. A : \text{Type} \rrbracket &= \sigma^*(\forall_I(\llbracket \Phi, \alpha \vdash A : \text{Type} \rrbracket)) \\ &= \forall_I((\sigma \times \text{Id}_U)^* \llbracket \Phi, \alpha \vdash A : \text{Type} \rrbracket) \quad \text{By Beck-Chevalley} \\ &= \forall_I(\llbracket \Phi', \alpha \vdash A[\sigma, \alpha: \alpha] : \text{Type} \rrbracket) \quad \text{By induction} \\ &= \llbracket \Phi' \vdash \forall \alpha. (A[\sigma, \alpha: \alpha]) : \text{Type} \rrbracket \\ &= \llbracket \Phi' \vdash (\forall \alpha. A)[\sigma] : \text{Type} \rrbracket\end{aligned}$$

□

Theorem 3.5.5 (Effect Substitution on Type Environments). *Similarly, the effect-substitution theorem on type environments is that if $\Phi \vdash \Gamma \text{ Ok}$, then $\Phi' \vdash \Gamma[\sigma] \text{ Ok}$ and $\llbracket \Phi' \vdash \Gamma[\sigma] \text{ Ok} \rrbracket = \sigma^* \llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket$.*

Proof: By induction on the derivation on $\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket$ whilst making use of the S-preserving property of the re-indexing functor.

Case Nil: We make use of the fact that an S-preserving functor preserves the terminal object.

$$\begin{aligned}\sigma^* \llbracket \Phi \vdash \diamond \text{ Ok} \rrbracket &= \sigma^* 1 \\ &= 1 \quad \text{By S-preserving property} \\ &= \llbracket \Phi' \vdash \diamond \text{ Ok} \rrbracket \\ &= \llbracket \Phi' \vdash \diamond[\sigma] \text{ Ok} \rrbracket\end{aligned}$$

Case Var: The S-preserving property means that $\sigma^*(A \times B) = (\sigma^*A) \times (\sigma^*B)$.

$$\begin{aligned}
\sigma^*[\llbracket \Phi \vdash \Gamma, x : A \text{ Ok} \rrbracket] &= \sigma^*([\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket \times \llbracket \Phi \vdash A : \text{Type} \rrbracket]) \\
&= (\sigma^*[\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket] \times \sigma^*[\llbracket \Phi \vdash A : \text{Type} \rrbracket]) \\
&= ([\llbracket \Phi' \vdash \Gamma[\sigma] \text{ Ok} \rrbracket \times \llbracket \Phi' \vdash A[\sigma] : \text{Type} \rrbracket]) \\
&= [\llbracket \Phi' \vdash \Gamma[\sigma], x : A[\sigma] \text{ Ok} \rrbracket] \\
&= [\llbracket \Phi' \vdash (\Gamma, x : A) [\sigma] \text{ Ok} \rrbracket]
\end{aligned}$$

□

The effect-weakening theorem on types and type environments is also very similar.

Theorem 3.5.6 (Effect Weakening on Types). *If $\omega: \Phi' \triangleright \Phi$ and $\Phi \vdash A$ then $\Phi' \vdash A$ $\llbracket \Phi' \vdash A \rrbracket = \omega^*[\llbracket \Phi \vdash A \rrbracket]$. Further more, $\omega: \Phi' \triangleright \Phi$ and $\Phi \vdash \Gamma \text{ Ok}$ imply $\llbracket \Phi' \vdash \Gamma \text{ Ok} \rrbracket = \omega^*[\llbracket \Phi \vdash \Gamma \text{ Ok} \rrbracket]$*

Proof: In a similar fashion, we make use of the Beck-Chevalley condition and the S-preserving property of σ^* .

Case Quantification: This case uses the Beck-Chevalley condition and the fact that $\llbracket \omega \times: (\Phi', \alpha) \triangleright (\Phi, \alpha) \rrbracket = \omega \times \text{Id}_U$. This property is used in conjunction with induction to change the environment from (Φ, α) to (Φ', α) .

$$\begin{aligned}
\omega^*[\llbracket \Phi \vdash \forall \alpha. A : \text{Type} \rrbracket] &= \omega^*(\forall_I([\llbracket \Phi, \alpha \vdash A : \text{Type} \rrbracket])) \\
&= \forall_I((\omega \times \text{Id}_U)^*[\llbracket \Phi, \alpha \vdash A : \text{Type} \rrbracket]) \quad \text{By Beck-Chevalley} \\
&= \forall_I([\llbracket \Phi', \alpha \vdash A : \text{Type} \rrbracket]) \quad \text{By induction} \\
&= \forall_I([\llbracket \Phi', \alpha \vdash A : \text{Type} \rrbracket]) \\
&= [\llbracket \Phi' \vdash \forall \alpha. A : \text{Type} \rrbracket]
\end{aligned}$$

Case Function: This case makes use of the S-preserving property that ω^* preserves exponentials. Specifically $\omega^*(B^A) = (\omega^*B)^{(\omega^*A)}$.

$$\begin{aligned}
\omega^*[\llbracket \Phi \vdash A \rightarrow B : \text{Type} \rrbracket] &= \omega^*([\llbracket \Phi \vdash B : \text{Type} \rrbracket^{\llbracket \Phi \vdash A : \text{Type} \rrbracket}]) \\
&= \omega^*([\llbracket \Phi \vdash B : \text{Type} \rrbracket]^{\omega^*([\llbracket \Phi \vdash A : \text{Type} \rrbracket])}) \\
&= [\llbracket \Phi' \vdash B : \text{Type} \rrbracket]^{\llbracket \Phi' \vdash A : \text{Type} \rrbracket} \\
&= [\llbracket \Phi' \vdash A \rightarrow B : \text{Type} \rrbracket]
\end{aligned}$$

The proof for type environments follows the same steps as effect-substitution proof.

□

Next we consider the action of weakening and substitution on subtyping relations.

Theorem 3.5.7 (Effect Substitution on Subtyping). *If A is a subtype of B under environment Φ ($A \leq_\Phi B$), and σ is a substitution from Φ' to Φ ($\Phi' \vdash \sigma : \Phi$), then $A[\sigma]$ is also a subtype of $B[\sigma]$ under Φ' ($A[\sigma] \leq_{\Phi'} B[\sigma]$) and $\llbracket A[\sigma] \leq_{\Phi'} B[\sigma] \rrbracket = \sigma^*[\llbracket A \leq_\Phi B \rrbracket]$.*

Proof: By rule induction over the definition of the subtype relation, making use of S-preserving property and the effect-substitution theorem on types.

Case Ground: This case holds by the property of the S-preserving property of σ^* that σ^* preserves the ground-type subtyping denotation.

$$\sigma^*(\gamma_1 \leq_{:\gamma} \gamma_2) = (\gamma_1 \leq_{:\gamma} \gamma_2)$$

Case Fn: This case holds by several S-preserving properties of σ^* . σ^* preserves the exponential morphisms: $\sigma^*(\text{cur}(f)) = \text{cur}(\sigma^*(f))$ and $\sigma^*(\text{app}) = \text{app}$. σ^* should also preserve the product morphisms: $\sigma^*(\pi_1) = \pi_1$, $\sigma^*(\pi_2) = \pi_2$ and $\sigma^*(\langle f, g \rangle) = \langle \sigma^*f, \sigma^*g \rangle$. Hence $\sigma^*(f \times g) = (\sigma^*f \times \sigma^*g)$.

$$\begin{aligned} \sigma^*[\llbracket (A \rightarrow B) \leq_{:\Phi} A' \rightarrow B' \rrbracket] &= \sigma^*([\llbracket B \leq_{:\Phi} B' \rrbracket^{A'} \circ B^{[A' \leq_{:\Phi} A]}]) \\ &= \sigma^*(\text{cur}([\llbracket B \leq_{:\Phi} B' \rrbracket] \circ \text{app})) \circ \sigma^*(\text{cur}(\text{app} \circ (\text{Id}_B \times [\llbracket A' \leq_{:\Phi} A \rrbracket]))) \\ &= \text{cur}(\sigma^*([\llbracket B \leq_{:\Phi} B' \rrbracket]) \circ \text{app}) \circ \text{cur}(\text{app} \circ (\text{Id}_B \times \sigma^*([\llbracket A' \leq_{:\Phi} A \rrbracket]))) \\ &= \text{cur}([\llbracket B[\sigma] \leq_{:\Phi'} B'[\sigma] \rrbracket] \circ \text{app}) \circ \text{cur}(\text{app} \circ (\text{Id}_{B[\sigma]} \times [\llbracket A'[\sigma] \leq_{:\Phi'} A[\sigma] \rrbracket]))) \\ &= [\llbracket (A[\sigma] \rightarrow B[\sigma]) \leq_{:\Phi'} (A'[\sigma] \rightarrow B'[\sigma]) \rrbracket] \\ &= [\llbracket (A \rightarrow B)[\sigma] \leq_{:\Phi'} (A' \rightarrow B')[\sigma] \rrbracket] \end{aligned}$$

□

Similarly we can form the symmetrical weakening theorem.

Theorem 3.5.8 (Effect Weakening on Subtyping). *If $A \leq_{:\Phi} B$ and $\omega: \Phi' \triangleright \Phi$, then $A \leq_{:\Phi'} B$ and $\llbracket A \leq_{:\Phi'} B \rrbracket = \omega^*[\llbracket A \leq_{:\Phi} B \rrbracket]$.*

Proof: The cases hold the same as in the corresponding substitution theorem.

□

We are now at a point to define and prove the effect weakening and substitution theorems on terms. Following the intuition above that changes of index object should be modelled by applying the re-indexing functor to the morphisms denoting the terms, we can construct the theorems. Firstly, we must define the operation of effect substitutions on terms.

$$\begin{aligned} x[\sigma] &= x \\ \mathbf{c}^A[\sigma] &= \mathbf{c}^{(A[\sigma])} \\ (\lambda x: A. v)[\sigma] &= \lambda x: (A[\sigma]). (v[\sigma]) \\ (\text{if}_A v \text{ then } v_1 \text{ else } v_2)[\sigma] &= \text{if}_{(A[\sigma])} v[\sigma] \text{ then } v_1[\sigma] \text{ else } v_2[\sigma] \\ (v_1 v_2)[\sigma] &= (v_1[\sigma]) v_2[\sigma] \\ (\text{do } x \leftarrow v_1 \text{ in } v_2)[\sigma] &= \text{do } x \leftarrow (v_1[\sigma]) \text{ in } (v_2[\sigma]) \\ (\Lambda \alpha. v)[\sigma] &= \Lambda \alpha. (v[\sigma]) \quad \text{If } \alpha \# \sigma \\ (v \epsilon)[\sigma] &= (v[\sigma]) \sigma(\epsilon) \end{aligned}$$

The substitution theorem is defined as so:

Theorem 3.5.9 (Effect Substitution on Terms). *If $\Phi' \vdash \sigma : \Phi$ and typing derivation tree Δ derives $\Phi \mid \Gamma \vdash v : A$ then there exists a typing derivation tree Δ' deriving $\Phi' \mid \Gamma[\sigma] \vdash v[\sigma] : A[\sigma]$ and $\Delta' = \sigma^*(\Delta)$.*

Proof: This proof makes use of the previous effect-substitution theorems, and the adjunction of quantification and the re-indexing functor of projection, π_1^* .

Case Effect-Lambda: This case makes use of the naturality condition 3.1 and simple reductions. If Δ derives $\Phi \mid \Gamma \vdash \Lambda\alpha.v : \forall\alpha.A$ then by inversion, there exists Δ_1 deriving $\Phi, \alpha \mid \Gamma \vdash v : A$ such that $\Delta = \overline{\Delta_1}$. By the extension lemma, we can pick $\alpha \notin \Phi' \wedge \alpha \notin \Phi$ so that we have: $(\Phi', \alpha) \vdash (\sigma, \alpha = \alpha) : (\Phi, \alpha)$. By induction, there exists Δ'_1 deriving $(\Phi', \alpha) \mid \Gamma[\sigma, \alpha = \alpha] \vdash v[\sigma, \alpha = \alpha] : A[\sigma, \alpha = \alpha]$, which we can simplify to $\Phi', \alpha \mid \Gamma[\sigma] \vdash v[\sigma] : A[\sigma]$ since α does not occur in Φ or Φ' . Hence $\Phi' \mid \Gamma[\sigma] \vdash v[\sigma] : (\forall\alpha.A)[\sigma]$ by the tree given in equation 3.14 (Δ').

$$\Delta' = (\text{Effect-Lambda}) \frac{\overline{\Delta'_1} \quad \overline{\Phi', \alpha \mid \Gamma[\sigma] \vdash v[\sigma] : A[\sigma]}}{\Phi' \mid \Gamma[\sigma] \vdash v[\sigma] : (\forall\alpha.A)[\sigma]} \quad (3.14)$$

It is also the case that:

$$\sigma \times \text{Id} = \llbracket (\Phi', \alpha) \vdash (\sigma, \alpha = \epsilon) : (\Phi, \alpha) \rrbracket \quad (3.15)$$

So

$$\begin{aligned} \sigma^* \Delta &= \sigma^*(\overline{\Delta_1}) \\ &= \overline{(\sigma \times \text{Id}_U)^* \Delta_1} \quad \text{By naturality} \\ &= \overline{\Delta'_1} \quad \text{By induction} \\ &= \Delta' \end{aligned}$$

Case Effect-Application: This is a more complex case, as it makes use of several naturality properties and the adjunction $\pi_1^* \dashv \forall_I$. By inversion, if Δ derives $\Phi \mid \Gamma \vdash v : A[\epsilon/\alpha]$ then there exists Δ_1 deriving $\Phi \mid \Gamma \vdash v : \forall\alpha.A$ and $h = \llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket$ such that Δ is derived from Δ_1 and h as in equations 3.16 and 3.17.

$$\Delta = (\text{Effect-Application}) \frac{\overline{\Delta_1} \quad \overline{\Phi \vdash \epsilon : \text{Effect}}}{\overline{\Phi \mid \Gamma \vdash v : \forall\alpha.A} \quad \overline{\Phi \mid \Gamma \vdash v : A[\epsilon/\alpha]}} \quad (3.16)$$

$$\Delta = \langle \text{Id}_\Gamma, h \rangle^* (\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}) \circ \Delta_1 \quad (3.17)$$

So by induction and effect-substitution preserving well-formedness of effects, we have Δ'_1 deriving $\Phi' \mid \Gamma[\sigma] \vdash v[\sigma] : (\forall\alpha.A)[\sigma]$ and $\Phi' \vdash \sigma(\epsilon)$. Using the effect-application type rule, we can construct Δ' deriving $\Phi' \mid \Gamma[\sigma] \vdash (v[\sigma]) (\sigma(\epsilon)) : A[\sigma][\sigma(\epsilon)/\alpha]$ from Δ'_1 , as in equation 3.18. Since $\alpha \# \sigma$, we can commute the applications of substitution. As a result, Δ' also derives $\Phi' \mid \Gamma[\sigma] \vdash (v \epsilon)[\sigma] : A[\epsilon/\alpha][\sigma]$.

$$\Delta' = (\text{Effect-Application}) \frac{\overline{\Delta'_1} \quad \overline{\Phi' \vdash \sigma(\epsilon)}}{\overline{\Phi' \mid \Gamma[\sigma] \vdash v[\sigma] : (\forall\alpha.A)[\sigma]} \quad \overline{\Phi' \mid \Gamma[\sigma] \vdash (v[\sigma]) (\sigma(\epsilon)) : A[\sigma][\sigma(\epsilon)/\alpha]}} \quad (3.18)$$

So, due to the substitution theorem on effects,

$$h \circ \sigma = \llbracket \Phi \vdash \epsilon : \text{Effect} \rrbracket \circ \sigma = \llbracket \Phi' \vdash \sigma(\epsilon) : \text{Effect} \rrbracket = h' \quad (3.19)$$

Hence, by applying the re-indexing functor to Δ , we have:

$$\begin{aligned} \sigma^* \Delta &= \sigma^* (\langle \text{Id}_\Gamma, h \rangle^* (\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}) \circ \Delta_1) \\ &= (\langle \text{Id}_\Gamma, h \rangle \circ \sigma)^* (\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}) \circ \sigma^* (\Delta_1) \\ &= ((\sigma \times \text{Id}_U) \circ \langle \text{Id}_\Gamma, h \circ \sigma \rangle)^* (\epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}) \circ \Delta'_1 \\ &= (\langle \text{Id}_\Gamma, h' \rangle)^* ((\sigma \times \text{Id}_U)^* \epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket}) \circ \Delta'_1 \end{aligned}$$

Looking at the inner part of the functor application: Let

$$A = \llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket$$

$$\begin{aligned} (\sigma \times \text{Id}_U)^* \epsilon_{\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \text{Type} \rrbracket} &= (\sigma \times \text{Id}_U)^* \epsilon_A \\ &= (\sigma \times \text{Id}_U)^* (\widehat{\text{Id}_{\forall_I(A)}}) \\ &= \overline{(\sigma \times \text{Id}_U)^* (\widehat{\text{Id}_{\forall_I(A)}})} \quad \text{By bijection} \\ &= \overline{\sigma^* (\widehat{\text{Id}_{\forall_I(A)}})} \quad \text{By naturality} \\ &= \overline{\sigma^* (\text{Id}_{\forall_I(A)})} \quad \text{By bijection} \\ &= \overline{\text{Id}_{\forall_{I'}(A \circ (\sigma \times \text{Id}_U))}} \quad \text{By S-preserving property, naturality} \\ &= \overline{\text{Id}_{\forall_{I'}(A[\sigma, \alpha := \alpha])}} \quad \text{By Substitution theorem} \\ &= \epsilon_{A[\sigma]} \end{aligned}$$

Going back to the original expression:

$$\begin{aligned} \sigma^* \Delta &= (\langle \text{Id}_\Gamma, h' \rangle)^* (\epsilon_{A[\sigma]}) \circ \Delta'_1 \\ &= \Delta' \end{aligned}$$

□

Theorem 3.5.10 (Effect Weakening on Terms). *Similarly, we can derive the weakening theorem on terms.*

If $\omega : \Phi' \triangleright \Phi$ and Δ derives $\Phi \mid \Gamma \vdash v : A$ then there exists Δ' deriving $\Phi' \mid \Gamma \vdash v : A$ and $\Delta' = \omega^ \Delta$.*

Proof: This theorem is proved in a similar fashion to the substitution theorem, and many of its cases are the same.

Case Subtype: If Δ derives $\Phi \mid \Gamma \vdash v : B$ then, by inversion, there exists Δ_1 deriving $\Phi \mid \Gamma \vdash v : A$, such that $\Delta = \llbracket A \leq_\Phi B \rrbracket \circ \Delta_1$, as shown in equation 3.20.

$$\Delta = (\text{Subtype}) \frac{\frac{\Delta_1}{\Phi \mid \Gamma \vdash v : A} \quad A \leq_\Phi B}{\Phi \mid \Gamma \vdash v : B} \quad (3.20)$$

By induction there exists Δ'_1 deriving $\Phi' \mid \Gamma \vdash v : A$, so we can construct Δ' using the subtyping rule in equation 3.21. So, using the weakening of the subtyping morphism, we can derive the denotation of Δ' .

$$\Delta' = (\text{Subtype}) \frac{\frac{\Delta'_1}{\Phi' \mid \Gamma \vdash v : A} \quad A \leq_{\Phi} B}{\Phi' \mid \Gamma \vdash v : B} \quad (3.21)$$

$$\begin{aligned} \omega^*(\Delta) &= \omega^*[\![A \leq_{\Phi} B]\!] \circ \omega^*\Delta_1 \\ &= [\![A \leq_{\Phi'} B]\!] \circ \Delta'_1 \quad \text{By induction} \\ &= \Delta' \end{aligned}$$

Case Lambda: This case holds by induction and the S-preserving property of ω^* .

If Δ derives $\Phi \mid \Gamma \vdash \lambda x : A.v : A \rightarrow B$ then by inversion there exists Δ_1 deriving $\Phi \mid \Gamma, x : A \vdash v : B$ such that Δ and its denotation can be defined as in equations 3.22 and 3.23

$$\Delta = (\text{Lambda}) \frac{\frac{\Delta_1}{\Phi \mid \Gamma, x : A \vdash v : B}}{\Phi \mid \Gamma \vdash \lambda x : A.v : A \rightarrow B} \quad (3.22)$$

$$\Delta = \text{cur}(\Delta_1) \quad (3.23)$$

By induction, there exists Δ'_1 deriving $\Phi' \mid \Gamma, x : A \vdash v : B$ with denotation $\Delta'_1 = \omega^*(\Delta_1)$. Using S-preserving property, we can derive Δ' and its denotation, seen in equations 3.24 and 3.25.

$$\Delta' = (\text{Lambda}) \frac{\frac{\Delta'_1}{\Phi' \mid \Gamma, x : A \vdash v : B}}{\Phi' \mid \Gamma \vdash \lambda x : A.v : A \rightarrow B} \quad (3.24)$$

$$\Delta' = \text{cur}(\Delta'_1) \quad (3.25)$$

Using S-closure, we can show that $\Delta' = \omega^*(\Delta)$.

$$\begin{aligned} \omega^*(\Delta) &= \omega^*(\text{cur}(\Delta_1)) \\ &= \text{cur}(\omega^*(\Delta_1)) \quad \text{By S-preserving property} \\ &= \text{cur}(\Delta'_1) \quad \text{By induction} \\ &= \Delta' \end{aligned}$$

□

Now we are at a point to start considering weakenings and substitution of the typing environment. Type environment weakenings are inductively defined with respect to an effect environment.

$$\begin{array}{ll} (\text{Id}) \frac{\Phi \vdash \Gamma \text{ Ok}}{\Phi \vdash \iota : \Gamma \triangleright \Gamma} & (\text{Project}) \frac{\Phi \vdash \omega : \Gamma' \triangleright \Gamma \quad \Phi \vdash A : \text{Type}}{\Phi \vdash \omega\pi : \Gamma, x : A \triangleright \Gamma} \text{ (if } x \notin \text{dom}(\Gamma') \text{)} \end{array}$$

$$(\text{Extend}) \frac{\Phi \vdash \omega : \Gamma' \triangleright \Gamma \quad A \leq B}{\Phi \vdash \omega \times : \Gamma', x : A \triangleright \Gamma, x : B} (\text{if } x \notin \text{dom}(\Gamma'))$$

The denotation of such a weakening is a morphism within an appropriate fibre category: $\llbracket \Phi \vdash \omega : \Gamma' \triangleright \Gamma \rrbracket : \Gamma' \rightarrow \Gamma \in \mathbb{C}(I)$. These denotations are defined as so:

$$\llbracket \Phi \vdash \iota : \Gamma \triangleright \Gamma \rrbracket = \text{Id}_\Gamma : \Gamma \rightarrow \Gamma \in \mathbb{C}(I) \quad \llbracket \Phi \vdash \omega \pi : \Gamma', ax \triangleright \Gamma \rrbracket = \llbracket \Phi \vdash \omega : \Gamma' \triangleright \Gamma \rrbracket \circ \pi_1 : \Gamma' \times A \rightarrow \Gamma$$

$$\llbracket \Phi \vdash \omega \times : \Gamma', x : A \triangleright \Gamma, x : B \rrbracket = \llbracket \Phi \vdash \omega : \Gamma' \triangleright \Gamma \rrbracket \times \llbracket A \leq_\Phi B \rrbracket : \Gamma' \times A \rightarrow \Gamma \times B$$

Type-environment substitutions are also derived inductively with respect to a effect environment.

$$(\text{Nil}) \frac{\Phi \vdash \Gamma' \text{ Ok}}{\Phi \mid \Gamma' \vdash \diamond : \diamond} \quad (\text{Extend}) \frac{\Phi \mid \Gamma' \vdash \sigma : \Gamma \quad \Phi \mid \Gamma' \vdash v : A}{\Phi \mid \Gamma' \vdash (\sigma, x := v) : (\Gamma, x : A)} (\text{if } x \notin \text{dom}(\Gamma'))$$

Denotations of type-environment substitutions are morphisms in the appropriate fibre category: $\llbracket \Phi \mid \Gamma' \vdash \sigma : \Gamma \rrbracket : \Gamma' \rightarrow \Gamma \in \mathbb{C}(I)$ and are defined as so:

$$\llbracket \Phi \mid \Gamma' \vdash \diamond : \diamond \rrbracket = \langle \rangle_{\Gamma'} : \Gamma' \rightarrow 1 \quad \llbracket \Phi \mid \Gamma' \vdash (\sigma, x := v) : \Gamma, x : A \rrbracket = \langle \llbracket \Phi \mid \Gamma' \vdash \sigma : \Gamma \rrbracket, \llbracket \Phi \mid \Gamma' \vdash v : A \rrbracket \rangle : \Gamma' \rightarrow \Gamma \times 1$$

We also need to explain the action of these term substitutions on terms. We define the action of applying a substitution σ on term v as $v[\sigma]$. The property $x \# \sigma$ indicates that x does not occur in the domain of σ or as a free variable in any of its substituted terms.

$$\begin{aligned} x[\sigma] &= x \\ x[\sigma, x := v] &= v \\ x[\sigma, x' := v'] &= x[\sigma] \quad \text{If } x \neq x' \\ \mathbf{C}^A[\sigma] &= \mathbf{C}^A \\ (\lambda x : A. v)[\sigma] &= \lambda x : A. (v[\sigma]) \quad \text{If } x \# \sigma \\ (\text{if}_A v \text{ then } v_1 \text{ else } v_2)[\sigma] &= \text{if}_A v[\sigma] \text{ then } v_1[\sigma] \text{ else } v_2[\sigma] \\ (v_1 v_2)[\sigma] &= (v_1[\sigma]) v_2[\sigma] \\ (\text{do } x \leftarrow v_1 \text{ in } v_2) &= \text{do } x \leftarrow (v_1[\sigma]) \text{ in } (v_2[\sigma]) \quad \text{If } x \# \sigma \\ (\Lambda \alpha. v)[\sigma] &= \Lambda \alpha. (v[\sigma]) \\ (v \epsilon)[\sigma] &= (v[\sigma]) \epsilon \end{aligned}$$

In order to prove the quantification case of typing-environment weakening and substitution theorems on terms, as can be seen in the road map in figure 3.4, it will be necessary to be able to weaken the effect environment on term-environment weakenings and substitutions

Let us first consider the action of effect weakenings on these morphisms. The weakening theorem on term-environment weakenings is as so:

Theorem 3.5.11 (Effect Weakening on Term Weakening). *If $\omega_1 : \Phi' \triangleright \Phi$ and $\Phi \vdash \omega : \Gamma' \triangleright \Gamma$ then $\Phi' \vdash \omega : \Gamma' \triangleright \Gamma$ and $\llbracket \Phi' \vdash \omega : \Gamma' \triangleright \Gamma \rrbracket = \omega_1^* \llbracket \Phi \vdash \omega : \Gamma' \triangleright \Gamma \rrbracket$.*

Proof: By induction on the derivation of ω . making use of weakening on types, type environments, and subtyping.

Case Id: Then $\omega = \iota$, so its denotation is $\omega = \text{Id}_{\Gamma_I}$

So

$$\omega_1^*(\text{Id}_{\Gamma_I}) = \text{Id}_{\Gamma_{I'}} = \llbracket \Phi' \vdash \iota : \Gamma \triangleright \Gamma \rrbracket \quad (3.26)$$

Case Project: Then $\omega = \omega' \pi$

$$(\text{Project}) \frac{\Phi \vdash \omega' : \Gamma' \triangleright \Gamma}{\Phi \vdash \omega \pi : \Gamma', x : A \triangleright \Gamma} \quad (3.27)$$

So $\omega = \omega' \circ \pi_1$

Hence

$$\begin{aligned} \omega_1^*(\omega) &= \omega_1^*(\omega') \circ \omega_1^*(\pi_1) \\ &= \llbracket \Phi' \vdash \omega' : \Gamma' \triangleright \Gamma \rrbracket \circ \pi_1 \\ &= \llbracket \Phi' \vdash \omega' \pi : (\Gamma', x : A) \triangleright \Gamma \rrbracket \\ &= \llbracket \Phi' \vdash \omega : (\Gamma', x : A) \triangleright \Gamma \rrbracket \end{aligned}$$

Case Extend: Then $\omega = \omega' \times$

$$(\text{Extend}) \frac{\Phi \vdash \omega' : \Gamma' \triangleright \Gamma \quad A \leq_{\Phi} B}{\Phi \vdash \omega \times : (\Gamma', x : A) \triangleright (\Gamma, x : B)} \quad (3.28)$$

So $\omega = \omega' \times \llbracket A \leq_{\Phi} B \rrbracket$

Hence

$$\begin{aligned} \omega_1^*(\omega) &= (\omega_1^*(\omega') \times \omega_1^*(\llbracket A \leq_{\Phi} B \rrbracket)) \\ &= (\llbracket \Phi' \vdash \omega' : \Gamma' \triangleright \Gamma \rrbracket \times \llbracket A \leq_{\Phi} B \rrbracket) \\ &= \llbracket \Phi' \vdash \omega : (\Gamma', x : A) \triangleright (\Gamma, x : B) \rrbracket \end{aligned}$$

□

Secondly, we can form the weakening theorem on term-environment substitutions.

Theorem 3.5.12 (Effect Weakening on Term Substitution). *If $\Phi \mid \Gamma' \vdash \sigma : \Gamma$ and $\omega : \Phi' \triangleright \Phi$ then $\Phi' \mid \Gamma' \vdash \sigma : \Gamma$ and $\llbracket \Phi' \mid \Gamma' \vdash \sigma : \Gamma \rrbracket = \omega^* \llbracket \Phi \mid \Gamma' \vdash \sigma : \Gamma \rrbracket$*

Proof: By induction on the definition of σ , making use of the weakening on terms, types, and type environments.

Case Nil: Then $\sigma = \langle \rangle_{\Gamma'}$, so $\omega^*(\sigma) = \langle \rangle_{\Gamma'_{I'}} = \llbracket \Phi' \mid \Gamma' \vdash \sigma : \Gamma \rrbracket$

Case Var: Then $\sigma = (\sigma', x: v)$

$$\begin{aligned}
\omega^* \sigma &= \omega * \langle \sigma', \llbracket \Phi \mid \Gamma \vdash v: A \rrbracket \rangle \\
&= \langle \omega^* \sigma', \omega^* \llbracket \Phi \mid \Gamma \vdash v: A \rrbracket \rangle \\
&= \langle \llbracket \Phi' \mid \Gamma' \vdash \sigma': \Gamma \rrbracket, \llbracket \Gamma' \mid \Phi' \vdash v: A \rrbracket \rangle \\
&= \llbracket \Phi' \mid \Gamma' \vdash \sigma: \Gamma, x: A \rrbracket
\end{aligned}$$

□

As with effect substitutions, it is useful to define a couple of special substitutions and lemmas on term substitutions. Firstly, we define the identity and single substitutions in an equivalent way.

$$\begin{aligned}
\Phi \mid \Gamma &\vdash \text{Id}_\Gamma: \Gamma \\
\text{Id}_\diamond &= \diamond \\
\text{Id}_{\Gamma, x: A} &= (\text{Id}_\Gamma, x: = x) \\
[v/x] &= (\text{Id}_\Gamma, x: = v)
\end{aligned}$$

By inspection, these also have simple denotations, similar to those seen in the case of effect substitutions (Equation 3.10).

$$\begin{aligned}
\llbracket \Phi \mid \Gamma \vdash \text{Id}_\Gamma: \Gamma \rrbracket &: \Gamma \rightarrow \Gamma \\
\llbracket \Phi \mid \Gamma \vdash \text{Id}_\Gamma: \Gamma \rrbracket &= \text{Id}_\Gamma \\
\llbracket \Phi \mid \Gamma \vdash [v/x]: \Gamma, x: A \rrbracket &: \Gamma \rightarrow \Gamma \times A \\
\llbracket \Phi \mid \Gamma \vdash [v/x]: \Gamma, x: A \rrbracket &= \langle \text{Id}_\Gamma, \llbracket \Phi \mid \Gamma \vdash v: A \rrbracket \rangle
\end{aligned}$$

Furthermore, it will be useful to have an analogue of the extension lemma for term substitutions.

Lemma 3.5.13 (Extension Lemma on Term Substitutions). *If $\Phi \mid \Gamma' \vdash \sigma: \Gamma$ and $x \# \sigma$ then $\Phi \mid (\Gamma', x: A) \vdash (\sigma, x: = x): (\Gamma, x: A)$ with denotation*

$$\llbracket \Phi \mid (\Gamma', x: A) \vdash (\sigma, x: = x): (\Gamma, x: A) \rrbracket = \llbracket \Phi \mid \Gamma' \vdash \sigma: \Gamma \rrbracket \times \text{Id}_A$$

Proof: Makes use of the weakening on terms (theorem 3.5.14). If $\Phi \mid \Gamma' \vdash \sigma: \Gamma$ then $\Phi \mid (\Gamma', x: A) \vdash \sigma: \Gamma$ with denotation $\sigma \circ \pi_1$. □

Now we can move onto the term substitution and weakening theorems. These theorems are the final step before we prove that derivations of the same typing-relation instance have the same denotation and then move onto soundness. They demonstrate that the can model the action of applying well-formed type-environment changes by pre-composing the morphisms to be acted on with a morphism modelling the change in environment.

The term-weakening theorem is as so:

Theorem 3.5.14 (Term Weakening). *If $\Phi \vdash \omega: \Gamma' \triangleright \Gamma$ and Δ is a derivation of $\Phi \mid \Gamma \vdash v: A$ then we can derive Δ' , a derivation of $\Phi \mid \Gamma' \vdash v: A$ with denotation $\Delta' = \Delta \circ \omega$.*

Proof: By induction on the derivation of Δ . Making use of the weakening of effect environments on term weakenings.

Case Effect-Lambda: This case makes use of the effect weakening of term weakenings.

If Δ derives $\Phi \mid \Gamma \vdash \Lambda\alpha.v:\forall\epsilon.A$, then by inversion, we have Δ_1 such that:

$$\Delta = (\text{Effect-Fn}) \frac{\frac{\Delta_1}{\Phi, \alpha \mid \Gamma \vdash v:A}}{\Phi \mid \Gamma \vdash \Lambda\alpha.v:\forall\epsilon.A} \quad (3.29)$$

By induction, we derive Δ'_1 such that it completes the tree in equation 3.30. By induction, its denotation can be related to that of Δ_1 , as in equation 3.31. Hence we can show that the denotation of Δ is preserved in equation 3.32.

$$\Delta' = (\text{Effect-Fn}) \frac{\frac{\Delta'_1}{\Phi, \alpha \mid \Gamma' \vdash v:A}}{\Phi \mid \Gamma' \vdash (\Lambda\alpha.v):\forall\epsilon.A} \quad (3.30)$$

$$\begin{aligned} \Delta'_1 &= \Delta_1 \circ \llbracket \Phi, \alpha \vdash \omega: \Gamma' \triangleright \Gamma \rrbracket \\ &= \Delta_1 \circ \llbracket \iota\pi: \Phi, a \triangleright \Phi \rrbracket^*(\omega) \\ &= \Delta_1 \circ \pi_1^*(\omega) \end{aligned} \quad (3.31)$$

$$\begin{aligned} \Delta \circ \omega &= \overline{\Delta_1} \circ \omega \\ &= \overline{\Delta_1 \circ \pi_1^*(\omega)} \\ &= \overline{\Delta'_1} \\ &= \Delta' \end{aligned} \quad (3.32)$$

Case Bind: This case makes use of the properties of an S-category, specifically the tensor strength on the graded monad. By inversion, we have derivations Δ_1, Δ_2 such that:

$$\Delta = (\text{Bind}) \frac{\frac{\Delta_1}{\Phi \mid \Gamma \vdash v_1:\mathbb{M}_{\epsilon_1}A} \quad \frac{\Delta_2}{\Phi \mid \Gamma, x:A \vdash v_2:\mathbb{M}_{\epsilon_2}B}}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2:\mathbb{M}_{\epsilon_1 \cdot \epsilon_2}B} \quad (3.33)$$

If $\Phi \vdash \omega:\Gamma' \triangleright \Gamma$ then $\Phi \vdash \omega \times: \Gamma', x:A \triangleright \Gamma, x:A$, so by induction, we can derive Δ'_1, Δ'_2 such that:

$$\Delta' = (\text{Bind}) \frac{\frac{\Delta'_1}{\Phi \mid \Gamma' \vdash v_1:\mathbb{M}_{\epsilon_1}A} \quad \frac{\Delta'_2}{\Phi \mid \Gamma', x:A \vdash v_2:\mathbb{M}_{\epsilon_2}B}}{\Phi \mid \Gamma' \vdash \text{do } x \leftarrow v_1 \text{ in } v_2:\mathbb{M}_{\epsilon_1 \cdot \epsilon_2}B} \quad (3.34)$$

This preserves denotations:

$$\begin{aligned}
\Delta' &= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta'_2 \circ \mathfrak{t}_{\epsilon_1, \Gamma', A} \circ \langle \text{Id}_{G'}, \Delta'_1 \rangle \quad \text{By definition} \\
&= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} (\Delta_2 \circ (\omega \times \text{Id}_A)) \circ \mathfrak{t}_{\epsilon_1, \Gamma', A} \circ \langle \text{Id}_{G'}, \Delta_1 \circ \omega \rangle \quad \text{By induction on } \Delta'_1, \Delta'_2 \\
&= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta_2 \circ \mathfrak{t}_{\epsilon_1, \Gamma, A} \circ \langle \omega, \Delta_1 \circ \omega \rangle \quad \text{By tensor strength} \\
&= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta_2 \circ \mathfrak{t}_{\epsilon_1, \Gamma, A} \circ \langle \text{Id}_\Gamma, \Delta_1 \rangle \circ \omega \quad \text{By product property} \\
&= \Delta \quad \text{By definition}
\end{aligned}$$

Case Return: We have the sub-derivation Δ_1 such that

$$\Delta = (\text{Return}) \frac{\frac{\Delta_1}{\Phi \mid \Gamma \vdash v : A}}{\Phi \mid \Gamma \vdash \text{return } v : \mathbf{M}_1 A} \quad (3.35)$$

Hence, by induction, with $\Phi \vdash \omega : \Gamma' \triangleright \Gamma$, we find the derivation Δ'_1 such that:

$$\Delta' = (\text{Return}) \frac{\frac{\Delta'_1}{\Phi \mid \Gamma' \vdash v : A}}{\Phi \mid \Gamma' \vdash \text{return } v : \mathbf{M}_1 A} \quad (3.36)$$

This preserves denotations:

$$\begin{aligned}
\Delta' &= \eta_A \circ \Delta'_1 \quad \text{By definition} \\
&= \eta_A \circ \Delta_1 \circ \omega \quad \text{By induction of } \Delta_1, \Delta'_1 \\
&= \Delta \circ \omega \quad \text{By Definition}
\end{aligned}$$

□

The term-substitution theorem is formed similarly:

Theorem 3.5.15 (Term Substitution). *If $\Phi \mid \Gamma' \vdash \sigma : \Gamma$ and Δ is a derivation of $\Phi \mid \Gamma \vdash v : A$, then we can construct Δ' deriving $\Phi \mid \Gamma' \vdash v[\sigma] : A$ with denotation $\Delta' = \Delta \circ \sigma$.*

Proof: This proof proceeds by induction on the derivation Δ similarly to the term weakening theorem and making use of weakening of term substitutions.

Case Weaken:

$$(\text{Weaken}) \frac{\frac{\Delta_1}{\Phi \mid \Gamma'' \vdash x : A}}{\Phi \mid \Gamma'', y : B \vdash x : A} \quad (3.37)$$

By inversion, $\Gamma = \Gamma', y : B$ and we have Δ_1 deriving the tree in equation 3.37. In addition, by inversion of the well-formedness of $\Phi \mid \Gamma' \vdash \sigma : \Gamma$, we can decompose σ into $(\sigma', y := v)$. Where σ' satisfies $\Phi \mid \Gamma' \vdash \sigma' : \Gamma''$, and the denotation of σ depends on that of σ' : $\llbracket \Phi \mid \Gamma' \vdash \sigma : \Gamma \rrbracket = \langle \llbracket \Phi \mid \Gamma' \vdash \sigma' : \Gamma'' \rrbracket, \llbracket \Phi \mid \Gamma' \vdash v : B \rrbracket \rangle$. Hence by induction on Δ_1 we have Δ'_1 deriving $(\Phi \mid \Gamma' \vdash x[\sigma] : A)$. So we can take Δ' to be Δ'_1 .

Hence

$$\begin{aligned}
\Delta' &= \Delta'_1 \quad \text{By definition} \\
&= \Delta_1 \circ \sigma' \quad \text{By induction} \\
&= \Delta_1 \circ \pi_1 \circ \langle \sigma', \llbracket \Phi \mid \Gamma' \vdash v : B \rrbracket \rangle \quad \text{By product property} \\
&= \Delta_1 \circ \pi_1 \circ \sigma \quad \text{By definition of the denotation of } \sigma \\
&= \Delta \circ \sigma \quad \text{By definition.}
\end{aligned}$$

Case Lambda: By inversion, we have Δ_1 such that the derivation tree in equation 3.38 holds. By induction on $(\Gamma, x : A)$ and Δ_1 we have Δ'_1 deriving $\Phi \mid \Gamma', x : A \vdash (v[\sigma]) : B$. Also by induction, and making use of the extension lemma, we can derive the denotation of Δ'_1 from Δ_1 : $\Delta'_1 = \Delta_1 \circ (\sigma \times \text{Id}_A)$. Furthermore, Δ'_1 allows us to construct the derivation Δ' , in equation 3.39.

$$\Delta = (\text{Fn}) \frac{\overline{D_1}}{\Phi \mid \Gamma, x : A \vdash v : B} \quad (3.38)$$

$$\Delta' = (\text{Fn}) \frac{\overline{\Delta'_1}}{\Phi \mid \Gamma' \vdash (\lambda x : A. v) [\sigma] : A \rightarrow B} \quad (3.39)$$

Hence:

$$\begin{aligned}
\Delta' &= \text{cur}(\Delta'_1) \quad \text{By definition} \\
&= \text{cur}(\Delta_1 \circ (\sigma \times \text{Id}_A)) \quad \text{By induction and extension lemma.} \\
&= \text{cur}(\Delta_1) \circ \sigma \quad \text{By the exponential property (Uniqueness)} \\
&= \Delta \circ \sigma \quad \text{By Definition}
\end{aligned}$$

□

3.6 Uniqueness of Denotations

Up until this point we have had to be careful about the implicit typing derivation for every term denotation $\llbracket \Phi \mid \Gamma \vdash v : A \rrbracket$. We are now equipped with the tools to prove that all derivations of the same typing relation instance $\Phi \mid \Gamma \vdash v : A$ induce the same denotation. This allows us to no longer worry about the equality of denotations, which will be helpful in the soundness proof.

To prove that all typing denotations have the same denotation, I first introduce the concept of a *reduced* typing derivation that is unique to each term and type in each typing environment. Next, I present a function, *reduce*, which recursively maps typing derivations to their reduced equivalent. I also prove that this function preserves the denotation of the derivations. That is $\llbracket \Phi \mid \Gamma \vdash v : A \rrbracket = \llbracket \text{reduce}(\Phi \mid \Gamma \vdash v : A) \rrbracket$. Hence, we can conclude, since all derivations for a typing relation instance reduce to the same unique typing derivation, and that the reduction function preserves the denotations, that all derivations of a typing derivation have the same denotation.

The need for reduced typing derivations comes about because of subtyping. The subtyping rule can be inserted into different places in a derivation to derive the same typing relation. Hence, the reduction function focuses on only placing subtyping rule instances in specific places.

In particular a reduced typing derivation is one where we only allow the subtyping rule to occur in three places. Firstly, the subtyping rule must occur exactly once at the root of the tree in order to coerce the the typing relation to the correct type. This root rule may be the identity subtyping rule, making use of the reflexive relation $A \leq_{\Phi} A$. Secondly, the subtyping rule can occur at the root of any of subtrees deriving the preconditions of the type-annotated (if) rule. This is to coerce the condition to a boolean type, and each of the branches to the type A in the if-expression itself. Finally, we allow the subtyping rule to occur at the root of the argument subtree of an (apply) rule. This allows us to coerce the argument to the correct type to match the parameter type in a lambda expression. These rules have the effect of only introducing subtyping rule uses when it is necessary maintain syntactic correctness of the derivation. As an aside, if subtyping were a syntactic feature, such as being induced by explicit casts, then the rules for the typing relation would be entirely syntax directed, and hence we would not have the ambiguity that these theorems are required in order to solve.

Theorem 3.6.1 (Uniqueness of reduced Derivations). *These reduced derivations are unique.*

Proof: This proof proceeds by induction on the term structure, making use of the unique derivations of the subterms to show that a reduced derivation of the whole term must also be unique. There are no cases for subtyping, as it is not a syntactic feature.

Case Bind: This case makes use of the weakening theorem on typing environments. Let the trees in equations 3.40, 3.41 be the respective unique reduced type derivations of the subterms. By weakening, $\Phi \vdash \iota \times : (\Gamma, x : A) \triangleright (\Gamma, x : A')$ so if there's a derivation of $\Phi \mid (\Gamma, x : A') \vdash v_2 : B$, there's also one of $\Phi \mid \Gamma, x : A \vdash v_2 : B$ (equation 3.42).

$$\text{(Subtype)} \frac{\frac{\Delta}{\Phi \mid \Gamma \vdash v_1 : \mathbf{M}_{\epsilon_1} A} \quad (\text{Computation}) \frac{A \leq_{\Phi} A' \quad \epsilon_1 \leq_{\Phi} \epsilon'_1}{\mathbf{M}_{\epsilon_1} A \leq_{\Phi} \mathbf{M}_{\epsilon'_1} A'}}{\Phi \mid \Gamma \vdash v_1 : \mathbf{M}_{\epsilon'_1} A'} \quad (3.40)$$

$$\text{(Subtype)} \frac{\frac{\Delta'}{\Phi \mid \Gamma, x : A' \vdash v_2 : \mathbf{M}_{\epsilon_2} B} \quad (\text{Computation}) \frac{B \leq_{\Phi} B' \quad \epsilon_2 \leq_{\Phi} \epsilon'_2}{\mathbf{M}_{\epsilon_2} B \leq_{\Phi} \mathbf{M}_{\epsilon'_2} B'}}{\Phi \mid \Gamma, x : A' \vdash v_2 : \mathbf{M}_{\epsilon'_2} B'} \quad (3.41)$$

$$\text{(Subtype)} \frac{\frac{\Delta''}{\Phi \mid (\Gamma, x : A) \vdash v_2 : \mathbf{M}_{\epsilon_2} B} \quad (\text{Computation}) \frac{B \leq_{\Phi} B' \quad \epsilon_2 \leq_{\Phi} \epsilon'_2}{\mathbf{M}_{\epsilon_2} B \leq_{\Phi} \mathbf{M}_{\epsilon'_2} B'}}{\Phi \mid (\Gamma, x : A) \vdash v_2 : \mathbf{M}_{\epsilon'_2} B'} \quad (3.42)$$

Since the effects monoid operation is monotone, if $\epsilon_1 \leq_{\Phi} \epsilon'_1$ and $\epsilon_2 \leq_{\Phi} \epsilon'_2$ then $\epsilon_1 \cdot \epsilon_2 \leq_{\Phi} \epsilon'_1 \cdot \epsilon'_2$. Hence the reduced type derivation of $\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : \mathbf{M}_{\epsilon'_1 \cdot \epsilon'_2} B'$ can be seen in equation 3.43.

$$\text{(Bind)} \frac{\frac{\Delta}{\Phi \mid \Gamma \vdash v_1 : \mathbf{M}_{\epsilon_1} A} \quad \frac{\Delta''}{\Phi \mid \Gamma, x : A \vdash v_2 : \mathbf{M}_{\epsilon_2} B} \quad (\text{Effect}) \frac{\epsilon_1 \cdot \epsilon_2 \leq_{\Phi} \epsilon'_1 \cdot \epsilon'_2 \quad B \leq_{\Phi} B'}{\mathbf{M}_{\epsilon_1 \cdot \epsilon_2} B \leq_{\Phi} \mathbf{M}_{\epsilon'_1 \cdot \epsilon'_2} B'}}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : \mathbf{M}_{\epsilon'_1 \cdot \epsilon'_2} B'} \quad (3.43)$$

Case Effect-Fn: If equation 3.45 is the unique reduced derivation of $\Phi, \alpha \mid \Gamma \vdash v : B$, then the unique reduced derivation of $\Phi \mid \Gamma \vdash \Lambda \alpha. A : \forall \alpha. B$ is shown in equation 3.44.

$$\text{(Subtype)} \frac{\text{(Effect-Fn)} \frac{\Delta}{\Phi, \alpha \mid \Gamma \vdash v : A} \quad \forall \alpha. A \leq_{\Phi} \forall \alpha. B}{\Phi \mid \Gamma \vdash \Lambda \alpha. v : \forall \alpha. A} \quad (3.44)$$

$$\text{(Subtype)} \frac{\Delta \quad A \leq_{\Phi, \alpha} B}{\Phi, \alpha \mid \Gamma \vdash v : B} \quad (3.45)$$

□

The *reduce* function maps each derivation to its reduced equivalent. It does this by pushing subtyping rules from the leaves of the derivation tree down towards the root of the tree. The function case splits on the root of the tree and works up recursively. Some cases of the function are given in figure 3.9.

Theorem 3.6.2 (Reduction preserves denotations). *If the derivation Δ' is the result of applying *reduce* to Δ then the denotations of the derivations are equal. That is $\Delta' = \text{reduce}(\Delta) \implies \Delta' = \Delta$.*

Proof: We proceed by induction over the structure of Δ , making use of the substitution and weakening theorems. We make use of the definition of the *reduce* function as defined in figure 3.9. We also use the definitions of $\Delta_1, \Delta_2 \dots$ from the figure. **TODO: Should I define $\Delta_1 \dots$ locally?**

Case Apply: This case makes use of the fact that composing subtyping morphisms gives the transitive subtyping morphism. Let us define some short-hands.

$$\begin{aligned} f &= \llbracket A \leq_{\Phi} A' \rrbracket : A \rightarrow A' \\ f' &= \llbracket A'' \leq_{\Phi} A \rrbracket : A'' \rightarrow A \\ g &= \llbracket B' \leq_{\Phi} B \rrbracket : B' \rightarrow B \end{aligned}$$

Hence

$$\begin{aligned} \llbracket A' \rightarrow B' \leq_{\Phi} A \rightarrow B \rrbracket &= (g)^A \circ (B')^f \\ &= \text{cur}(\text{app} \circ \text{app}) \circ \text{cur}(\text{app} \circ (\text{Id} \times f)) \\ &= \text{cur}(g \circ \text{app} \circ (\text{Id} \times f)) \end{aligned}$$

Then

$$\begin{aligned} \Delta &= \text{app} \circ \langle \Delta_1, \Delta_2 \rangle \quad \text{By definition} \\ &= \text{app} \circ \langle \text{cur}(g \circ \text{app} \circ (\text{Id} \times f)) \circ \Delta'_1, f' \circ \Delta'_2 \rangle \quad \text{By reductions of } \Delta_1, \Delta_2 \\ &= \text{app} \circ (\text{cur}(g \circ \text{app} \circ (\text{Id} \times f)) \times \text{Id}_A) \circ \langle \Delta'_1, f' \circ \Delta'_2 \rangle \quad \text{Factoring out} \\ &= g \circ \text{app} \circ (\text{Id} \times f) \circ \langle \Delta'_1, f' \circ \Delta'_2 \rangle \quad \text{By the exponential property} \\ &= g \circ \text{app} \circ \langle \Delta'_1, f \circ f' \circ \Delta'_2 \rangle \\ &= \Delta' \quad \text{By definition} \end{aligned}$$

Case Apply: To find the reduction of a tree ending in (*Apply*) as seen in equation 3.46, we pattern match on the reductions of the subtrees to find trees Δ'_1 , Δ'_2 , as seen in equations 3.47 and 3.48. These trees allow us to build a reduced derivation tree as in equation 3.49.

$$reduce((Apply) \frac{\frac{\Delta_1}{\Phi \mid \Gamma \vdash v_1 : A \rightarrow B} \quad \frac{\Delta_2}{\Phi \mid \Gamma \vdash v_2 : A}}{\Phi \mid \Gamma \vdash v_1 v_2 : B}) \quad (3.46)$$

$$(Subtype) \frac{\frac{\Delta'_1}{\Phi \mid \Gamma \vdash v_1 : A' \rightarrow B'} \quad A' \rightarrow B' \leq_{\Phi} A \rightarrow B}{\Phi \mid \Gamma \vdash v_1 : A \rightarrow B} = reduce(\Delta_1) \quad (3.47)$$

$$(Subtype) \frac{\frac{\Delta'_2}{\Phi \mid \Gamma \vdash v : A'} \quad A' \leq_{\Phi} A}{\Phi \mid \Gamma \vdash v_1 : A} = reduce(\Delta_2) \quad (3.48)$$

$$(Subtype) \frac{(Apply) \frac{\frac{\Delta'_1}{\Phi \mid \Gamma \vdash v_1 : A' \rightarrow B'} \quad (Subtype) \frac{\frac{\Delta'_2}{\Phi \mid \Gamma \vdash v_2 : A''} \quad A'' \leq_{\Phi} A \leq_{\Phi} A'}{\Phi \mid \Gamma \vdash v_2 : A'}}{\Phi \mid \Gamma \vdash v_1 v_2 : B'} \quad B' \leq_{\Phi} B}{\Phi \mid \Gamma \vdash v_1 v_2 : B} \quad (3.49)$$

Case Bind: To find the reduction of a tree ending in a use of the (*Bind*) rule as seen in equation 3.50, we first reduce the left-most subtree, Δ_1 to find Δ'_1 , as seen in equation 3.51.

$$reduce((Bind) \frac{\frac{\Delta_1}{\Phi \mid \Gamma \vdash v_1 : M_{\epsilon_1} A} \quad \frac{\Delta_2}{\Phi \mid \Gamma, x : A \vdash v_2 : M_{\epsilon_2} B}}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : M_{\epsilon_1 \cdot \epsilon_2} B}) \quad (3.50)$$

$$(Subtype) \frac{\frac{\Delta'_1}{\Phi \mid \Gamma \vdash v_1 : M_{\epsilon'_1} A'} \quad (Effect) \frac{\epsilon'_1 \leq_{\Phi} \epsilon_1 \quad A' \leq_{\Phi} A}{M_{\epsilon'_1} A' \leq_{\Phi} M_{\epsilon_1} A}}{\Phi \mid \Gamma \vdash v_1 : M_{\epsilon_1} A} = reduce(\Delta_1) \quad (3.51)$$

Since $\Phi \vdash (i, \times) : (\Gamma, x : A') \triangleright (\Gamma, x : A)$ if $A' \leq_{\Phi} A$, and because Δ_2 derives $\Phi \mid (\Gamma, x : A) \vdash v_2 : M_{\epsilon_2} B$, there also exists a derivation Δ_3 of $\Phi \mid (\Gamma, x : A') \vdash v_2 : M_{\epsilon_2} B$. Δ_3 is derived from Δ_2 simply by inserting a (*Subtype*) rule below all instances of the (*Var*) rule. Then we can reduce Δ_3 to find Δ'_3 in 3.52.

$$(Subtype) \frac{\frac{\Delta'_3}{\Phi \mid \Gamma, x : A' \vdash v_2 : M_{\epsilon'_2} B'} \quad (Effect) \frac{\epsilon'_1 \leq_{\Phi} \epsilon_2 \quad B' \leq_{\Phi} B}{M_{\epsilon'_1} B' \leq_{\Phi} M_{\epsilon_2} B}}{\Phi \mid \Gamma, x : A' \vdash v_2 : M_{\epsilon_2} B} = reduce(\Delta_3) \quad (3.52)$$

Since the effects monoid operation is monotone, if $\epsilon_1 \leq_{\Phi} \epsilon'_1$ and $\epsilon_2 \leq_{\Phi} \epsilon'_2$ then $\epsilon_1 \cdot \epsilon_2 \leq_{\Phi} \epsilon'_1 \cdot \epsilon'_2$. Then the result of reduction of the whole bind expression is:

$$(Subtype) \frac{subeffecttree \epsilon'_1 \cdot \epsilon'_2 B' \epsilon_1 \cdot \epsilon_2 B \quad (Bind) \frac{\frac{\Delta'_1}{\Phi \mid \Gamma \vdash v_1 : M_{\epsilon'_1} A'} \quad \frac{\Delta'_3}{\Phi \mid \Gamma, x : A' \vdash v_2 : M_{\epsilon'_2} B'}}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : M_{\epsilon'_1 \cdot \epsilon'_2} B}}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } v_2 : M_{\epsilon_1 \cdot \epsilon_2} B} \quad (3.53)$$

Figure 3.9: Some cases of the reduce function.

Case Bind: This is a long case that makes use of the typing-environment weakening theorem on terms.

Let us define the following helper morphisms:

$$\begin{aligned} f &= \llbracket A' \leq_{\Phi} A \rrbracket : A' \rightarrow A \\ g &= \llbracket B' \leq_{\Phi} B \rrbracket : B' \rightarrow B \\ h_1 &= \llbracket \epsilon'_1 \leq_{\Phi} \epsilon_1 \rrbracket : T_{\epsilon'_1} \rightarrow T_{\epsilon_1} \\ h_2 &= \llbracket \epsilon'_2 \leq_{\Phi} \epsilon_2 \rrbracket : T_{\epsilon'_2} \rightarrow T_{\epsilon_2} \\ h &= \llbracket \epsilon'_1 \cdot \epsilon'_2 \leq_{\Phi} \epsilon_1 \cdot \epsilon_2 \rrbracket : T_{\epsilon'_1 \cdot \epsilon'_2} \rightarrow T_{\epsilon_1 \cdot \epsilon_2} \end{aligned}$$

Due to the denotation of the weakening used to derive Δ_3 from Δ_2 , we have

$$\Delta_3 = \Delta_2 \circ (\text{Id}_{\Gamma} \times f) \quad (3.54)$$

Due to the reduction of Δ_3 , we have

$$\Delta_3 = h_{2,B} \circ T_{\epsilon'_2} g \circ \Delta'_3 \quad (3.55)$$

So we can look at the denotation of the whole term.

$$\begin{aligned} \Delta &= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta_2 \circ \mathbf{t}_{\epsilon_1, \Gamma, A} \circ \langle \text{Id}_{\Gamma}, \Delta_1 \rangle \quad \text{By definition.} \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta_2 \circ \mathbf{t}_{\epsilon_1, \Gamma, A} \circ \langle \text{Id}_{\Gamma}, h_{1,A} \circ T_{\epsilon'_1} f \circ \Delta'_1 \rangle \quad \text{By reduction of } \Delta_1. \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta_2 \circ \mathbf{t}_{\epsilon_1, \Gamma, A} \circ (\text{Id}_{\Gamma} \times h_{1,A}) \circ \langle \text{Id}_{\Gamma}, T_{\epsilon'_1} f \circ \Delta'_1 \rangle \quad \text{Factor out } h_1 \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ T_{\epsilon_1} \Delta_2 \circ h_{1,(\Gamma \times A)} \circ \mathbf{t}_{\epsilon'_1, \Gamma, A} \circ \langle \text{Id}_{\Gamma}, T_{\epsilon'_1} f \circ \Delta'_1 \rangle \quad \text{Tensor strength and subeffecting } h_1 \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ h_{1,B} \circ T_{\epsilon'_1} \Delta_2 \circ \mathbf{t}_{\epsilon'_1, \Gamma, A} \circ \langle \text{Id}_{\Gamma}, T_{\epsilon'_1} f \circ \Delta'_1 \rangle \quad \text{Naturality of } h_1 \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ h_{1,B} \circ T_{\epsilon'_1} \Delta_2 \circ \mathbf{t}_{\epsilon'_1, \Gamma, A} \circ (\text{Id}_{\Gamma} \times T_{\epsilon'_1} f) \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{Factor out pairing again} \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ h_{1,B} \circ T_{\epsilon'_1} (\Delta_2 \circ (\text{Id}_{\Gamma} \times f)) \circ \mathbf{t}_{\epsilon'_1, \Gamma, A'} \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{Tensor strength} \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ h_{1,B} \circ T_{\epsilon'_1} (\Delta_3) \circ \mathbf{t}_{\epsilon'_1, \Gamma, A'} \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{By the definition of } \Delta_3 \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ h_{1,B} \circ T_{\epsilon'_1} (h_{2,B} \circ T_{\epsilon'_2} g \circ \Delta'_3) \circ \mathbf{t}_{\epsilon'_1, \Gamma, A'} \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{By the reduction of } \Delta_3 \\ &= \mu_{\epsilon_1, \epsilon_2, B} \circ h_{1,B} \circ T_{\epsilon'_1} h_{2,B} \circ T_{\epsilon'_1} T_{\epsilon'_2} g \circ T_{\epsilon'_1} \Delta'_3 \circ \mathbf{t}_{\epsilon'_1, \Gamma, A'} \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{Factor out the functor} \\ &= h_B \circ \mu_{\epsilon'_1, \epsilon'_2, B} \circ T_{\epsilon'_1} T_{\epsilon'_2} g \circ T_{\epsilon'_1} \Delta'_3 \circ \mathbf{t}_{\epsilon'_1, \Gamma, A'} \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{By the } \mu \text{ and Subtype rule} \\ &= h_B \circ T_{\epsilon'_1 \cdot \epsilon'_2} g \circ \mu_{\epsilon'_1, \epsilon'_2, B'} \circ T_{\epsilon'_1} \Delta'_3 \circ \mathbf{t}_{\epsilon'_1, \Gamma, A'} \circ \langle \text{Id}_{\Gamma}, \Delta'_1 \rangle \quad \text{By naturality of } \mu \\ &= \Delta' \quad \text{By definition} \end{aligned}$$

□

$$\text{(Subtype)} \frac{\text{(Lambda)} \frac{\Phi \mid \Gamma, x : A \vdash v : B \quad B \leq_{\Phi} B'}{\Phi \mid \Gamma, x : A \vdash v : B'}}{\Phi \mid \Gamma \vdash \lambda x : A. v : A \rightarrow B'} \quad (3.56)$$

$$\text{(Subtype)} \frac{\text{(Lambda)} \frac{\Phi \mid \Gamma, x : A \vdash v : B}{\Phi \mid \Gamma \vdash \lambda x : A. v : A \rightarrow B} \quad A \rightarrow B \leq : A \rightarrow B'}{\Phi \mid \Gamma \vdash \lambda x : A. v : A \rightarrow B'} \quad (3.57)$$

Figure 3.10: Two derivations of the same type relation. The second derivation is reduced.

3.7 Soundness

We are now at a stage where we can state and prove the most important theorem for a denotational semantics: soundness with respect to an equational equivalence. Soundness follows from the common sense requirement that terms that are equivalent in a given language should also have equivalent denotations. In our case, I introduce a $\beta\eta$ -based equational equivalence relation and then prove that equivalent terms have equal denotations.

The equational equivalence relation is a rule based relation with three main flavours of rules. Firstly, as seen in figure 3.11, there are the reductions which formalise how we expect the program to execute given an appropriate implementation. We give a $\beta\eta$ -reduction for each term transition, such as the application of lambda terms or the execution of an **if** expression, as well as the monad equivalence laws. Secondly, there are congruences, seen in figure 3.12, which formalise how the reduction of subexpressions affects the rest of the expression in a compositional way. Finally, we extend this relation into an equivalence relation by closing it under transitivity, reflexivity and symmetry as seen in figure 3.13.

$$\begin{array}{c}
\text{(Lambda-Beta)} \frac{\Phi \mid \Gamma, x: A \vdash v_2: B \quad \Phi \mid \Gamma \vdash v_1: A}{\Phi \mid \Gamma \vdash (\lambda x: A. v_1) v_2 \approx v_1 [v_2/x]: B} \quad \text{(Lambda-Eta)} \frac{\Phi \mid \Gamma \vdash v: A \rightarrow B}{\Phi \mid \Gamma \vdash \lambda x: A. (v x) \approx v: A \rightarrow B} \\
\\
\text{(Left Unit)} \frac{\Phi \mid \Gamma \vdash v_1: A \quad \Phi \mid \Gamma, x: A \vdash v_2: \mathbf{M}_\epsilon B}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow \text{return } v_1 \text{ in } v_2 \approx v_2 [v_1/x]: \mathbf{M}_\epsilon B} \quad \text{(Right Unit)} \frac{\Phi \mid \Gamma \vdash v: \mathbf{M}_\epsilon A}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v \text{ in return } x \approx v: \mathbf{M}_\epsilon A} \\
\\
\text{(Associativity)} \frac{\Phi \mid \Gamma \vdash v_1: \mathbf{M}_{\epsilon_1} A \quad \Phi \mid \Gamma, x: A \vdash v_2: \mathbf{M}_{\epsilon_2} B \quad \Phi \mid \Gamma, y: B \vdash v_3: \mathbf{M}_{\epsilon_3} C}{\Phi \mid \Gamma \vdash \text{do } x \leftarrow v_1 \text{ in } (\text{do } y \leftarrow v_2 \text{ in } v_3) \approx \text{do } y \leftarrow (\text{do } x \leftarrow v_1 \text{ in } v_2) \text{ in } v_3: \mathbf{M}_{\epsilon_1 \cdot \epsilon_2 \cdot \epsilon_3} C} \\
\\
\text{(Unit)} \frac{\Phi \mid \Gamma \vdash v: \mathbf{Unit}}{\Phi \mid \Gamma \vdash v \approx (): \mathbf{Unit}} \\
\\
\text{(If-True)} \frac{\Phi \mid \Gamma \vdash v_1: A \quad \Phi \mid \Gamma \vdash v_2: A}{\Phi \mid \Gamma \vdash \text{if}_A \text{ true then } v_1 \text{ else } v_2 \approx v_1: A} \quad \text{(If-False)} \frac{\Phi \mid \Gamma \vdash v_2: A \quad \Phi \mid \Gamma \vdash v_1: A}{\Phi \mid \Gamma \vdash \text{if}_A \text{ false then } v_1 \text{ else } v_2 \approx v_2: A} \\
\\
\text{(If-Eta)} \frac{\Phi \mid \Gamma, x: \mathbf{Bool} \vdash v_2: A \quad \Phi \mid \Gamma \vdash v_1: \mathbf{Bool}}{\Phi \mid \Gamma \vdash \text{if}_A v_1 \text{ then } v_2 [\text{true}/x] \text{ else } v_2 [\text{false}/x] \approx v_2 [v_1/x]: A} \\
\\
\text{(Effect-beta)} \frac{\Phi \vdash \epsilon \quad \Phi, \alpha \mid \Gamma \vdash v: A}{\Phi \mid \Gamma \vdash (\Lambda \alpha. v \epsilon) \approx v [\epsilon/\alpha]: A [\epsilon/\alpha]} \text{ntreeruleIEffect-eta} \Phi \mid \Gamma \vdash v: \forall \alpha. A \Phi \mid \Gamma \vdash \Lambda \alpha. (v \alpha) \approx v: \forall \alpha. A
\end{array}$$

Figure 3.11: The reduction rules for PEC

Now we can state the soundness theorem.

Theorem 3.7.1 (Soundness). *If $\Phi \mid \Gamma \vdash v_1 \approx v_2: A$, then $\Phi \mid \Gamma \vdash v_1: A$, $\Phi \mid \Gamma \vdash v_2: A$, and $\llbracket \Phi \mid \Gamma \vdash v_1: A \rrbracket = \llbracket \Phi \mid \Gamma \vdash v_2: A \rrbracket$.*

Proof: The proof proceeds by induction on the definition of the equational equivalence relation.

This proof has a lot of cases and each of the reduction cases makes use of many of the requirements we have placed on the indexed S-category.

I have omitted the congruence cases here as they hold through simple application of the inductive hypothesis on subterms. This occurs because this denotational semantics is compositional, meaning that denotations of terms are defined entirely in terms of the denotations of their subexpression. Similarly,

$$\begin{array}{c}
\text{(Effect-Abs)} \frac{\Phi, \alpha \mid \Gamma \vdash v_1 \approx v_2 : A}{\Phi \mid \Gamma \vdash \Lambda \alpha. v_1 \approx \Lambda \alpha. v_2 : \forall \alpha. A} \quad \text{(Effect-Apply)} \frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : \forall \alpha. A \quad \Phi \vdash \epsilon}{\Phi \mid \Gamma \vdash v_1 \epsilon \approx v_2 \epsilon : A[\epsilon/\alpha]} \\
\\
\text{(Lambda)} \frac{\Phi \mid \Gamma, x : A \vdash v_1 \approx v_2 : B}{\Phi \mid \Gamma \vdash \lambda x : A. v_1 \approx \lambda x : A. v_2 : A \rightarrow B} \quad \text{(Return)} \frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : A}{\Phi \mid \Gamma \vdash \mathbf{return} v_1 \approx \mathbf{return} v_2 : \mathbf{M}_1 A} \\
\\
\text{(Apply)} \frac{\Phi \mid \Gamma \vdash v_1 \approx v'_1 : A \rightarrow B \quad \Phi \mid \Gamma \vdash v_2 \approx v'_2 : A}{\Phi \mid \Gamma \vdash v_1 v_2 \approx v'_1 v'_2 : B} \quad \text{(Bind)} \frac{\Phi \mid \Gamma \vdash v_1 \approx v'_1 : \mathbf{M}_{\epsilon_1} A \quad \Phi \mid \Gamma, x : A \vdash v_2 \approx v'_2 : \mathbf{M}_{\epsilon_2} B}{\Phi \mid \Gamma \vdash \mathbf{do} x \leftarrow v_1 \mathbf{in} v_2 \approx \mathbf{do} x \leftarrow v'_1 \mathbf{in} v'_2 : \mathbf{M}_{\epsilon_1 \cdot \epsilon_2} B} \\
\\
\text{(If)} \frac{\Phi \mid \Gamma \vdash v \approx v' : \mathbf{Bool} \quad \Phi \mid \Gamma \vdash v_1 \approx v'_1 : A \quad \Phi \mid \Gamma \vdash v_2 \approx v'_2 : A}{\Phi \mid \Gamma \vdash \mathbf{if}_A v \mathbf{then} v_1 \mathbf{else} v_2 \approx \mathbf{if}_A v' \mathbf{then} v'_1 \mathbf{else} v'_2 : A} \quad \text{(Subtype)} \frac{\Phi \mid \Gamma \vdash v \approx v' : A \quad A \leq_\Phi B}{\Phi \mid \Gamma \vdash v \approx v' : B}
\end{array}$$

Figure 3.12: The congruence rules for PEC

$$\begin{array}{c}
\text{(Reflexive)} \frac{\Phi \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash v \approx v : A} \quad \text{(Symmetric)} \frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : A}{\Phi \mid \Gamma \vdash v_2 \approx v_1 : A} \\
\\
\text{(Transitive)} \frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : A \quad \Phi \mid \Gamma \vdash v_2 \approx v_3 : A}{\Phi \mid \Gamma \vdash v_1 \approx v_3 : A}
\end{array}$$

Figure 3.13: Rules expanding the reduction and congruence relation to an equivalence relation

the equivalence relation cases hold simply because equality on morphisms is an equivalence relation by definition. I do however, give a selection of the reduction cases to demonstrate the necessity of the S-category requirements.

Case Right Unit: This case makes use of the right-unit monad law. Let $f = \llbracket \Phi \mid \Gamma \vdash v : \mathbf{M}_\epsilon A \rrbracket$.

$$\begin{aligned}
\llbracket \Phi \mid \Gamma \vdash \mathbf{do} x \leftarrow v \mathbf{in} \mathbf{return} x : \mathbf{M}_\epsilon A \rrbracket &= \mu_{\epsilon, 1, A} \circ T_\epsilon(\eta_A \circ \pi_2) \circ \mathbf{t}_{\epsilon, \Gamma, A} \circ \langle \mathbf{Id}_\Gamma, f \rangle \\
&= T_\epsilon \pi_2 \circ \mathbf{t}_{\epsilon, \Gamma, A} \circ \langle \mathbf{Id}_\Gamma, f \rangle \\
&= \pi_2 \circ \langle \mathbf{Id}_\Gamma, f \rangle \\
&= f
\end{aligned} \tag{3.58}$$

Case If-True: This case makes use of the co-product diagram on $1 + 1$. Let $f = \llbracket \Phi \mid \Gamma \vdash v_1 : A \rrbracket$ and $g = \llbracket \Phi \mid \Gamma \vdash v_2 : A \rrbracket$. Then we can simplify the denotation of the whole expression.

$$\begin{aligned}
\llbracket \Phi \mid \Gamma \vdash \mathbf{if}_A v \mathbf{then} v_1 \mathbf{else} v_2 : A \rrbracket &= \mathbf{app} \circ ((\llbracket \mathbf{cur}(f \circ \pi_2), \mathbf{cur}(g \circ \pi_2) \rrbracket \circ \mathbf{inl} \circ \langle \rangle_\Gamma) \times \mathbf{Id}_\Gamma) \circ \delta_\Gamma \\
&= \mathbf{app} \circ ((\llbracket \mathbf{cur}(f \circ \pi_2) \rrbracket \circ \langle \rangle_\Gamma) \times \mathbf{Id}_\Gamma) \circ \delta_\Gamma \\
&= \mathbf{app} \circ (\llbracket \mathbf{cur}(f \circ \pi_2) \rrbracket \times \mathbf{Id}_\Gamma) \circ (\langle \rangle_\Gamma \times \mathbf{Id}_\Gamma) \circ \delta_\Gamma \\
&= f \circ \pi_2 \circ \langle \rangle_\Gamma \circ \mathbf{Id}_\Gamma \\
&= f \\
&= \llbracket \Phi \mid \Gamma \vdash v_1 : A \rrbracket
\end{aligned} \tag{3.59}$$

Case Effect-Beta: This case makes use of the adjunction properties of \forall_I, π_1^* . Let $h = \llbracket \Phi \vdash \epsilon : \mathbf{Effect} \rrbracket$, $f = \llbracket \Phi, \alpha \mid \Gamma \vdash v : A \rrbracket$, and $A = \llbracket \Phi, \alpha \vdash A[\alpha/\alpha] : \mathbf{Type} \rrbracket$. Then we can find the sub-term denotation $\llbracket \Phi \mid \Gamma \vdash \Lambda\alpha.v : \forall\alpha.A \rrbracket = \bar{f}$, which allows us to express the denotation of the whole expression.

$$\begin{aligned}
\llbracket \Phi \mid \Gamma \vdash (\Lambda\alpha.v) \epsilon : \forall\alpha.A \rrbracket &= \langle \text{Id}_I, h \rangle^*(\epsilon_A) \circ \bar{f} \\
&= \langle \text{Id}_I, h \rangle^*(\epsilon_A) \circ \langle \text{Id}_I, h \rangle^*(\pi_1^*(\bar{f})) \quad \text{Identity functor} \\
&= \langle \text{Id}_I, h \rangle^*(\epsilon_A \circ \pi_1^*(\bar{f})) \\
&= \langle \text{Id}_I, h \rangle^*(f) \quad \text{By adjunction} \\
&= \llbracket \Phi \mid \Gamma \vdash v[\epsilon/\alpha] : A[\epsilon/\alpha] \rrbracket \quad \text{By substitution theorem}
\end{aligned}$$

Case Effect-Eta: Let $f = \llbracket \Phi \mid \Gamma \vdash v : \forall\alpha.A \rrbracket$, and $A = \llbracket \Phi, \alpha \vdash A : \mathbf{Type} \rrbracket$. We can now express the denotation of the entire expression.

$$\begin{aligned}
\llbracket \Phi \mid \Gamma \vdash \Lambda\alpha.(v \alpha) : \forall\alpha.A \rrbracket &= \overline{\llbracket \Phi, \alpha \mid \Gamma \vdash v \alpha : A \rrbracket} \\
&= \overline{\langle \text{Id}_{I \times U}, \pi_2 \rangle^*(\epsilon_{\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket}) \circ \pi_1^*(f)}
\end{aligned}$$

Let us look at $\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket$. We have the weakening $\iota\pi \times : \Phi, \alpha, \beta \triangleright \Phi, \beta$, so by the weakening theorem on type denotations, we can re-arrange the quantification and projection functors.

$$\begin{aligned}
\forall_{I \times U}(\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket) &= \forall_I((\pi_1 \times \text{Id}_U)^*\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket) \\
&= \pi_1^*\forall_I(\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket)
\end{aligned}$$

Using this and the adjunction and naturality properties (3.1), we can unwind the definition of the co-unit on $\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket$.

$$\begin{aligned}
\epsilon_{\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket} &= \overline{\text{Id}_{\forall_{I \times U}(\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket)}} \\
&= \overline{\text{Id}_{\pi_1^*\forall_I(\llbracket \Phi, \beta \vdash A[\beta/\alpha] : \mathbf{Type} \rrbracket)}} \\
&= \overline{\text{Id}_{\pi_1^*\forall_I A}} \\
&= \overline{\pi_1^*(\text{Id}_{\forall_I A})} \\
&= \overline{\pi_1^*(\overline{\epsilon_A})} \\
&= \overline{(\pi_1 \times \text{Id}_U)^*(\epsilon_A)} \\
&= (\pi_1 \times \text{Id}_U)^*(\epsilon_A)
\end{aligned}$$

This rearrangement can now be used in conjunction with the contravariant composition of re-indexing

functors to show the denotation $\llbracket \Phi \mid \Gamma \vdash \Lambda\alpha.(v\ \alpha):\forall\alpha.A \rrbracket$ is equal to the morphism f .

$$\begin{aligned}
\llbracket \Phi \mid \Gamma \vdash \Lambda\alpha.(v\ \alpha):\forall\alpha.A \rrbracket &= \overline{\langle \text{Id}_{I \times U}, \pi_2 \rangle^* (\epsilon_{\llbracket \Phi, \alpha, \beta \vdash A[\beta/\alpha]:\text{Type} \rrbracket}) \circ \pi_1^*(f)} \\
&= \overline{\langle \text{Id}_{I \times U}, \pi_2 \rangle^* ((\pi_1 \times \text{Id}_U)^* (\epsilon_A)) \circ \pi_1^*(f)} \\
&= \overline{\langle \pi_1, \pi_2 \rangle^* (\epsilon_A) \circ \pi_1^*(f)} \\
&= \overline{\text{Id}_{I \times U}^* (\epsilon_A) \circ \pi_1^*(f)} \\
&= \overline{\epsilon_A \circ \pi_1^*(f)} \quad \text{The identity re-indexing functor is the identity.} \\
&= f \quad \text{By adjunction}
\end{aligned}$$

□

This completes the proof of soundness for this semantics.

Chapter 4

Instantiating a Model of PEC

Now we have proved that an appropriate categorical structure can model PEC, it remains to show that it is feasible to construct such an indexed category. There exist **Set**-based models for the semantics of many effectful languages with a graded monad, such as a certain instantiations of the Effect Calculus [2]. More specifically, it is possible to treat **Set** as an S-category. Hence, I use a **Set** based S-category as a starting point. In this section, I demonstrate how to construct an strictly indexed S-category, which can model the PEC, from such an S-category.

Suppose we have an S-category structure derived from **Set** which models an instance of EC. That is, there exists a graded monad $\mathbf{T}^0, \mu^0, \eta^0, \mathbf{t}^0$ on **Set**, and there exist subtyping functions $\llbracket A \leq_\gamma B \rrbracket : A \rightarrow B$ for each instance of the ground subtyping relation, and natural transformations $\llbracket \epsilon_1 \leq_0 \epsilon_2 \rrbracket : \mathbf{T}_{\epsilon_1}^0 \rightarrow \mathbf{T}_{\epsilon_2}^0$ for each instance of the subeffecting relation. Further more, **Set** is cartesian closed and has co-products for all pairs of objects. Since this structure is a model for the EC, it is graded by a partially-ordered monoid on ground effects: $(E, \leq_0, 1, \cdot)$. I have marked each of these instances of S-category structure with 0 to indicate that they occur in the fibre induced by the empty effect-variable environment.

Using α -equivalence, we can see that all effect environments of the same length are equivalent. For example, the terms in equation 4.1 are α -equivalent. Hence, we can reduce an effect environment to the natural number n indicating its length.

$$((\diamond, \alpha, \beta) \mid \Gamma \vdash \lambda f : \text{Int} \rightarrow \mathbf{M}_{\alpha, \beta} \mathbf{Unit}. (f \ 0) : \mathbf{M}_{\alpha, \beta} \mathbf{Unit}) =_\alpha ((\diamond, \gamma, \delta) \mid \Gamma \vdash \lambda f : \text{Int} \rightarrow \mathbf{M}_{\gamma, \delta} \mathbf{Unit}. (f \ 0) : \mathbf{M}_{\gamma, \delta} \mathbf{Unit}) \quad (4.1)$$

We also need to define the base category. This shall be **Eff**, a sub category of **Set**, populated by the set of ground effects E as the effect object U , its finite products, and monotone functions between them.

$$\begin{aligned} E^0 &= 1 = \{*\} \\ E^{n+1} &= E^n \times E \end{aligned}$$

Morphisms $E^n \rightarrow E$ in **Eff** are monotone functions taking n ground-effect parameters and returning a ground effect. The ground effects ranged over by e are represented by point functions $e : 1 \rightarrow E$, so the denotation of a ground effect e is the constant valued function $\llbracket \Phi \vdash e : \mathbf{Effect} \rrbracket = \vec{e} \mapsto e$. We also need to define a monoidal operation $\mathbf{Mul}_n : \mathbf{Eff}(E^n, E) \times \mathbf{Eff}(E^n, E) \rightarrow \mathbf{Eff}(E^n, E)$ that is natural as described in section 3.2. A patent choice for \mathbf{Mul}_n and proof of the naturality property is shown in figure 4.1.

Next we must pick our fibre categories for non-zero values of n . A simple instantiation is to pick each fibre $\mathbf{C}(E^n)$ to be the functor category $[E^n, \mathbf{Set}]$, where E^n is a discrete category (a category that only contains the identity morphisms). That is, the category of functions returning an object in **Set** given

The Monoidal Mul Operator

$$\mathbf{Mul}_n(f, g)(\vec{e}) = (f\vec{e}) \cdot (g\vec{e})$$

This satisfies naturality of \mathbf{Mul} .

$$(\mathbf{Mul}_m(f, g) \circ \theta)\vec{e} = (f(\theta\vec{e})) \cdot (g(\theta\vec{e})) = (\mathbf{Mul}_n(f \circ \theta), (g \circ \theta))\vec{e}$$

\mathbf{Mul}_n is monoidal, as required, due to the monoid structure of the operator \cdot on ground effects.

Figure 4.1: Definition of the \mathbf{Mul}_n operator

Cartesian Closed Category

Since E is small, E^n is also small, and hence $[E^n, \mathbf{Set}]$ is a CCC. This can be demonstrated pointwise.

$$(A \times B)\vec{e} = (A\vec{e}) \times (B\vec{e})$$

$$1\vec{e} = 1$$

$$(B^A)\vec{e} = (B\vec{e})^{(A\vec{e})}$$

$$\pi_1\vec{e} = \pi_1$$

$$\pi_2\vec{e} = \pi_2$$

$$\mathbf{app}\vec{e} = \mathbf{app}$$

$$\mathbf{cur}(f)\vec{e} = \mathbf{cur}(f\vec{e})$$

$$\langle f, g \rangle \vec{e} = \langle f\vec{e}, g\vec{e} \rangle$$

Figure 4.2: The construction of CCC features in the functor category $[E^n, \mathbf{Set}]$

n ground effects. Morphisms between objects are functions which, given a collection of ground effects, yield a morphism (function) in C . If $m \in [E^n, \mathbf{Set}](A, B)$ then $m\vec{e} \in \mathbf{Set}(A\vec{e}, B\vec{e})$. The fibres, $[E^n, C]$ are S-categories. This can be proved by constructing the S-category structures pointwise with respect to their parameter $\vec{e} \in E^n$ as seen in figures 4.2 - 4.7. Full proofs of the naturality and monad laws can be found **Reference Needed: (link/appendix)**.

Now we need to define the required morphisms between fibres. Firstly, for any function $\theta : E^m \rightarrow E^n$ in \mathbf{Eff} , there should exist an S-preserving re-indexing functor $\theta^* : [E^n, \mathbf{Set}] \rightarrow [E^m, \mathbf{Set}]$. A simple instantiation is the pre-composition functor, as seen in figure 4.8. This also obeys the composition law of re-indexing functors figure 4.9 and is S-preserving.

Next, the re-indexing functor π_1^* should have a right adjoint, \forall_{E^n} . Under the current construction, the only option for such a right adjoint is for \forall_{E^n} to be defined as a product over the set of effects (figure 4.11). This is possible because types and effects are not impredicative (that is, they do not quantify over themselves), meaning that the set of effects is countable or finite.

We can now prove that $\pi_1^* \dashv \forall_{E^n}$. To do this, we need a natural bijection between morphisms in the

The Terminal Co-Product
 We can define the co-product pointwise.

$$\begin{aligned}
 (1 + 1)\vec{\epsilon} &= (1\vec{\epsilon} + 1\vec{\epsilon}) \\
 &= (1 + 1) \\
 \text{inl}\vec{\epsilon} &= \text{inl} \\
 \text{inr}\vec{\epsilon} &= \text{inr} \\
 [f, g]\vec{\epsilon} &= [f\vec{\epsilon}, g\vec{\epsilon}]
 \end{aligned}$$

Figure 4.3: The construction of co-product features in the functor category $[E^n, \mathbf{Set}]$

Ground Types and Terms

Each ground type in the non-polymorphic calculus has a fixed denotation $\llbracket \gamma \rrbracket \in \mathbf{obj} \ \mathbf{Set}$. The ground type in the polymorphic calculus hence has a denotation represented by the constant function.

$$\begin{aligned}
 \llbracket \gamma \rrbracket &: E^n \rightarrow \mathbf{obj} \ \mathbf{Set} \\
 \vec{\epsilon} &\mapsto \llbracket \gamma \rrbracket
 \end{aligned}$$

Each constant term \mathbf{C}^A in the non-polymorphic calculus has a fixed denotation $\llbracket \mathbf{C}^A \rrbracket \in \mathbf{Set}(1, A)$. So the morphism $\llbracket \mathbf{C}^A \rrbracket$ in $[E^n, \mathbf{Set}]$ is the corresponding constant dependently typed morphism.

$$\begin{aligned}
 \llbracket \mathbf{C}^A \rrbracket &: [E^n, \mathbf{Set}](1, A) \\
 \vec{\epsilon} &\mapsto \llbracket \mathbf{C}^A \rrbracket
 \end{aligned}$$

Figure 4.4: The definition of ground types and terms in the category $[E^n, \mathbf{Set}]$

Strong Graded Monad

Given the strong graded monad $(T^0, \eta^0, \mu^0, \tau^0)$ on **Set** we can construct an appropriate graded monad, $(T^n, \eta^n, \mu^n, \tau^n)$, on $[E^n, \mathbf{Set}]$.

$$\begin{aligned} T^n : (E^n, \cdot, \leq_n, 1_n) &\rightarrow [[E^n, \mathbf{Set}], [E^n, \mathbf{Set}]] \\ (T_f^n A)\vec{e} &= T_{(f\vec{e})}^0 A\vec{e} \\ (\eta_A^n)\vec{e} &= \eta_{A\vec{e}}^0 \\ (\mu_{f,g,A}^n)\vec{e} &= \mu_{(f\vec{e}), (g\vec{e}), (A\vec{e})}^0 \\ (\tau_{f,A,B}^n)\vec{e} &= \tau_{(f\vec{e}), (A\vec{e}), (B\vec{e})}^0 \end{aligned}$$

Through some mechanical proof and the naturality of the T^0 strong graded monad, these morphisms are natural in their type parameters and form a strong graded monad in $[E^n, \mathbf{Set}]$

Figure 4.5: The definition of the graded monad in the category $[E^n, \mathbf{Set}]$

Subeffecting

Given a collection of subeffecting natural transformations in **Set**,

$$[\epsilon_1 \leq_0 \epsilon_2] : T_{\epsilon_1}^0 \rightarrow T_{\epsilon_2}^0$$

We can form subeffect natural transformations in $[E^n, \mathbf{Set}]$:

$$\begin{aligned} [f \leq_n g] : T_f^n &\rightarrow T_g^n \\ ([f \leq_n g]A)\vec{e} : T_{f\vec{e}}^n(A\vec{e}) &\rightarrow T_{g\vec{e}}^n(B\vec{e}) \\ &= [f\vec{e} \leq_0 g\vec{e}](A\vec{e}) \end{aligned}$$

Figure 4.6: Subeffect natural transformations in the category $[E^n, \mathbf{Set}]$

Subtyping

Subtyping in $[E^n, \mathbf{Set}]$ holds via subtyping in **Set**

$$\begin{aligned} [A \leq_n B] : A &\rightarrow B \\ [A \leq_n B]\vec{e} &= [A\vec{e} \leq_0 B\vec{e}] \end{aligned}$$

So the subtyping relation $A \leq B$ forms a morphism in $[E^n, \mathbf{Set}]$

Figure 4.7: Definition of subtyping morphisms in the functor category $[E^n, \mathbf{Set}]$

The Pre-composition Functor

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

Figure 4.8: A description of the pre-composition functor.

Composition of Pre-Composition Functor

$$\begin{aligned} \theta^*(\phi^* A)\vec{\epsilon} &= \phi^*(A)(\theta\vec{\epsilon}) \\ &= A(\phi(\theta\vec{\epsilon})) \\ &= A((\phi \circ \theta)\vec{\epsilon}) \\ &= (\phi \circ \theta)^*(A)\vec{\epsilon} \end{aligned}$$

Figure 4.9: The pre-composition functor composes in a contravariant fashion.

Re-Indexing Functors are S-Preserving

Theorem 4.0.1. *The re-indexing functors are also S-preserving. That is all the S-category features in $[E^n, \mathbf{Set}]$ are preserved by θ^* in $[E^m, \mathbf{Set}]$. For a full list of features please see appendix **Reference Needed:** ().*

Proof: All of the S-category features are proved pointwise.

Case Graded Monad Functor:

$$\begin{aligned} (\theta^* \mathbf{T}_f^n A) \vec{\epsilon} &= \mathbf{T}_f^n A(\theta \vec{\epsilon}) \\ &= \mathbf{T}_{(f(\theta \vec{\epsilon}))}^0 (A(\theta \vec{\epsilon})) \\ &= (\mathbf{T}_{(f \circ \theta)}^m \theta^* A) \vec{\epsilon} \end{aligned}$$

Case Terminal Object:

$$\begin{aligned} (\theta^* 1) \vec{\epsilon} &= 1(\theta \vec{\epsilon}) \\ &= 1 \\ &= 1 \vec{\epsilon} \end{aligned}$$

$$\begin{aligned} (\theta^* \langle \rangle_A) \vec{\epsilon} &= \langle \rangle_A (\theta \vec{\epsilon}) \\ &= \langle \rangle_{A(\theta \vec{\epsilon})} \\ &= \langle \rangle_{\theta^* A} \vec{\epsilon} \end{aligned}$$

□

Figure 4.10: The pre-composition re-indexing functor f^* is S-Preserving

The Quantification Functor

$$\begin{aligned} \forall_{E^n} : [E^{n+1}, \mathbf{Set}] &\rightarrow [E^n, \mathbf{Set}] \\ \forall_{E^n}(A) \vec{\epsilon}_n &= \prod_{\epsilon \in E} A(\vec{\epsilon}_n, \epsilon) \\ \forall_{E^n}(f) \vec{\epsilon}_n &= \prod_{\epsilon \in E} f(\vec{\epsilon}_n, \epsilon) \end{aligned}$$

Figure 4.11: Definition of the quantification functor

The Adjunction Operations

$$\begin{aligned} f &: \pi_1^* A \rightarrow B \\ \bar{f} &: A \rightarrow \forall_{E^n} B \\ \bar{f}(\vec{\epsilon}_n) &= \langle f(\vec{\epsilon}_n) \rangle_{e \in E} \end{aligned}$$

$$\begin{aligned} g &: A \rightarrow \forall_{E^n} B \\ \hat{g} &: \pi_1^* A \rightarrow B \\ \hat{g}(\vec{\epsilon}_n, \epsilon_{n+1}) &= \pi_{\epsilon_{n+1}} \circ g(\vec{\epsilon}_n) \end{aligned}$$

Figure 4.12: The definition of the adjunction's bijection operations

Unit	Co-Unit
$\eta_A : A \rightarrow \forall_{E^n} \pi_1^* A$ $\eta_A(\vec{\epsilon}_n) = \langle \text{Id}_{A(\vec{\epsilon}_n, e)} \rangle_{e \in E}$	$\epsilon_B : \pi_1^* \forall_{E^n} B \rightarrow B$ $\epsilon_B(\vec{\epsilon}_n, \epsilon') = \pi_{\epsilon'} : \prod_{e \in E} B(\vec{\epsilon}_n, e) \rightarrow B(\vec{\epsilon}_n, \epsilon')$

Figure 4.13: The unit and co-unit of the adjunction

functor categories $[E^n, \mathbf{Set}]$ and $[E^{n+1}, \mathbf{Set}]$.

$$\overline{(-)} : [E^{n+1}, \mathbf{Set}](\pi_1^* A, B) \rightleftharpoons [E^n, \mathbf{Set}](A, \forall_{E^n} B) : \widehat{(-)} \quad (4.2)$$

The leftwards and rightwards components of this bijection can be derived as follows. The leftwards component maps each morphism to a finite pairing of the morphism over each ground effect. The inverse is simply to project out the appropriate value of ϵ from the product. These mappings can be seen in figure 4.12. These transformations give rise to the unit and co-unit of the adjunction, which are defined in figure 4.13. The unit and co-unit allow us to prove that this construction is an adjunction (figure 4.14).

Finally, we need to prove that the Beck-Chevalley conditions given in 3.2 holds. The proof of the appropriate theorems are given in figure 4.15.

Hence we have proof that our construction is indeed a valid indexed S-category. Importantly, this shows that reasonable models of the PEC are possible to instantiate and that our requirements do not overconstrain potential models to the point that they are not viable for doing actual analysis.

Verifying We Have an Adjunction

For any $g : \pi_1^* A \rightarrow B$,

$$\begin{aligned} (\epsilon_B \circ \pi_1^*(\bar{g}))(\vec{\epsilon}_n, \epsilon_{n+1}) &= \pi_{\epsilon_{n+1}} \circ \langle g(\vec{\epsilon}_n, \epsilon') \rangle_{\epsilon' \in E} \\ &= g(\vec{\epsilon}_n, \epsilon_{n+1}) \end{aligned}$$

So $\epsilon_B \circ \pi_1^*(\bar{g}) = g$.

Similarly, for any $f : A \rightarrow \forall_{E^n} B$,

$$\begin{aligned} ((\forall_{E^n}(\hat{f})) \circ \eta_A)(\vec{\epsilon}_n) &= \Pi_{e \in E}(\hat{f}(\vec{\epsilon}_n, \epsilon)) \circ \langle \text{Id}_{A(\vec{\epsilon}_n, \epsilon)} \rangle_{\epsilon \in E} \\ &= \Pi_{e \in E}(\pi_\epsilon \circ f(\vec{\epsilon}_n)) \circ \langle \text{Id}_{A(\vec{\epsilon}_n, \epsilon)} \rangle_{\epsilon \in E} \\ &= \langle \pi_\epsilon \circ f(\vec{\epsilon}_n) \rangle_{\epsilon \in E} \\ &= \langle \pi_\epsilon \rangle_{\epsilon \in E} \circ f(\vec{\epsilon}_n) \\ &= f(\vec{\epsilon}_n) \end{aligned}$$

□

Figure 4.14: Proof that we have formed an adjunction

Theorem 4.0.2 (Beck-Chevalley condition, part I). *Firstly, the functors $(\theta^* \circ \forall_{E^n})$ and $(\forall_{E^m} \circ (\theta \times \text{Id}_E)^*)$ are equal. Secondly, the natural transformation $(\theta \times \text{Id}_U)^* \epsilon$ is equal to the identity natural transformation.*

Proof: Firstly,

$$\begin{aligned}
((\theta^* \circ \forall_{E^n})A)\epsilon_n^{\vec{}} &= \theta^*(\forall_{E^n}A)\epsilon_n^{\vec{}} \\
&= (\forall_{E^n}A)(\theta(\epsilon_n^{\vec{}})) \\
&= \Pi_{\epsilon \in E}(A(\theta(\epsilon_n^{\vec{}}), \epsilon)) \\
&= \Pi_{\epsilon \in E}((\theta \times \text{Id}_U)^*A)(\epsilon_n^{\vec{}}, \epsilon) \\
&= \forall_{E^m}((\theta \times \text{Id}_E)^*A)\epsilon_n^{\vec{}} \\
&= ((\forall_{E^m} \circ (\theta \times \text{Id}_E)^*)A)\epsilon_n^{\vec{}}
\end{aligned}$$

Secondly,

$$\begin{aligned}
\overline{(\theta \times \text{Id}_U)^* \epsilon_A \vec{\epsilon}} &= \langle (\theta \times \text{Id}_U)^* \epsilon_A(\vec{\epsilon}, \epsilon) \rangle_{\epsilon \in E} \\
&= \langle \epsilon_A(\theta \vec{\epsilon}, \epsilon) \rangle_{\epsilon \in E} \\
&= \langle \pi_\epsilon \rangle_{\epsilon \in E} : \Pi_{\epsilon \in E} A(\theta \vec{\epsilon}, \epsilon) \rightarrow \Pi_{\epsilon \in E} A(\theta \vec{\epsilon}, \epsilon) \\
&= \text{Id}_{\Pi_{\epsilon \in E} A(\theta \vec{\epsilon}, \epsilon)} \\
&= \text{Id}_{\forall_{I'} \circ (\theta \times \text{Id}_U)^* A} \vec{\epsilon} \\
&= \text{Id}_{\theta^* \circ \forall_I}
\end{aligned}$$

□

Figure 4.15: Proof that the adjunction satisfies the Beck-Chevalley condition.

Chapter 5

Conclusion

This dissertation has introduced a denotational semantics for polymorphic effect systems in category theory and has proved that non-trivial models capable of interpreting effect-polymorphic languages can indeed be instantiated. The final piece of the puzzle would be to find an adequate model of a given effect-polymorphic language. This means instantiating a sound model such that if the denotations of two terms are equal in the model, then the terms themselves are equal up to the same equational equivalence. This is typically a hard problem which is often not addressed in presentations of denotational semantics.

This work forms a potential foundation for more precise analysis tools which could be used to improve compilers and interpreters in the future. A more precise analysis of programs allows more specific optimisations to be made, such as code re-ordering or removal of dead code, where they would not be considered sound under a non-polymorphic system.

Bibliography

- [1] E. Moggi, “Notions of computation and monads,” *Information and Computation*, vol. 93, no. 1, pp. 55–92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science.
- [2] S.-y. Katsumata, “Parametric effect monads and semantics of effect systems,” *SIGPLAN Not.*, vol. 49, pp. 633–645, Jan. 2014.
- [3] Q. Ma and J. C. Reynolds, “Types, abstraction, and parametric polymorphism, part 2,” in *Mathematical Foundations of Programming Semantics* (S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, eds.), (Berlin, Heidelberg), pp. 1–40, Springer Berlin Heidelberg, 1992.
- [4] J. C. Reynolds, “Polymorphism is not set-theoretic,” in *Semantics of Data Types* (G. Kahn, D. B. MacQueen, and G. Plotkin, eds.), (Berlin, Heidelberg), pp. 145–156, Springer Berlin Heidelberg, 1984.
- [5] S. MacLane, *Categories for the Working Mathematician*. New York: Springer-Verlag, 1971. Graduate Texts in Mathematics, Vol. 5.
- [6] J. C. Reynolds, “Towards a theory of type structure,” in *Programming Symposium, Proceedings Colloque Sur La Programmation*, (Berlin, Heidelberg), pp. 408–423, Springer-Verlag, 1974.
- [7] R. L. Crole, *Polymorphic Functional Type Theory*, p. 201–274. Cambridge University Press, 1994.