A Denotational Semantics for Polymorphic Effect Systems

Alexander Taylor

May 3, 2019

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. The usage of monads was made more precise by Katsumata ([2]) using a graded monad to model languages with a range of independent and dependent effects at an operational level. Separately, 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). However, 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 with parameterisation over effects to yield a more general lambda calculus with polymorphism over effects. 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, though in a terser format, can be found online on my github repository[1], since due to the number of theorems and cases, the total size is well over 100 pages of definitions, theorems, and proofs.

[1] `https://github.com/A1153/Part3Project/blob/master/diss/OnlinePECSemantics.pdf`

Contents

1 Introduction **5**

1.1 What is an Effect System? . . . 5

1.2 What is Effect Polymorphism? . . . 5

1.3 An Introduction to Categorical Semantics . . . 6

2 Background **8**

2.1 Required Category Theory . . . 8

2.1.1 Cartesian Closed Category . . . 8

2.1.2 Co-Product . . . 9

2.1.3 Functors . . . 9

2.1.4 Natural Transformations . . . 10

2.1.5 Monad . . . 10

2.1.6 Monoid . . . 10

2.1.7 Graded Monad . . . 11

2.1.8 Tensor Strength . . . 11

2.1.9 Adjunction . . . 11

2.1.10 Strictly Indexed Category . . . 13

2.2 Language Features and Their Requirements . . . 13

2.3 The Effect Calculus . . . 16

2.4 Polymorphic Effect Calculus . . . 19

2.4.1 Type System . . . 19

3 The Semantics of PEC in an Strictly Indexed Category **25**

3.1 Semantics for EC in an S-Category . . . 25

3.2 Required Category Structure . . . 26

3.3 Road Map . . . 28

3.4 Denotations . . . 31

3.5 Effect Substitution and Weakening Theorems . . . 34

3.5.1 Substitution and Weakening on Effects . . . 35

3.5.2 Substitution and Weakening on Typing . . . 38

3.5.3 Effect Substitution and Weakening on Terms . . . 41
3.5.4 Term Substitution and Weakening . . . 44
3.5.5 Summary . . . 50
3.6 Uniqueness of Denotations . . . 50
3.7 Soundness . . . 55
3.8 Summary . . . 59

4 Constructing a Model of PEC **60**

5 Conclusion **69**

Bibliography **70**

A Additional Proofs **71**
A.1 Effect Weakening on Effects . . . 71
A.2 Effect Weakening on Types . . . 72
A.3 Effect Weakening on Terms . . . 73
A.4 Term Weakening Theorem . . . 74

B S-Category Functors **77**
B.1 Re-Indexing Functors . . . 77
B.1.1 Preserves Ground Types . . . 77
B.1.2 f^* Preserves Products . . . 77
B.1.3 f^* Preserves Terminal Object . . . 78
B.1.4 f^* Preserves Exponentials . . . 78
B.1.5 f^* Preserves Co-product on Terminal . . . 78
B.1.6 f^* Preserves Graded Monad . . . 78
B.1.7 f^* and Effects . . . 78
B.1.8 f^* Preserves Tensor Strength . . . 78
B.1.9 f^* Preserves Ground Constants . . . 79
B.1.10 f^* Preserves Ground Subeffecting . . . 79
B.1.11 f^* Preserves Ground Subtyping . . . 79
B.2 The $\forall_I$ functor . . . 79
B.2.1 Beck Chevalley Condition . . . 79
B.3 Naturality Corollaries . . . 80
B.3.1 Naturality . . . 80
B.3.2 $\overline{(-)}$ and Re-indexing Functors . . . 80
B.3.3 $\hat{(-)}$ and Re-Indexing Functors . . . 80
B.3.4 Pushing Morphisms into f^* . . . 81

C Functorial S-Categories **82**

C.1 Base Category is Monoidal 82
C.2 S-Categories 82
C.2.1 Each S-Category is a CCC 84
C.2.2 The Terminal Co-Product 84
C.2.3 Ground Types and Terms 84
C.2.4 Graded Monad 85
C.2.5 Subeffecting 86
C.3 Re-Indexing Functors 86

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 returning 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 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 perform effect analysis using 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](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, where types can range over themselves [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 $[\![X]\!]$ 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 mapping of types and type environments to objects in a specific categorical structure. That is, there should exist objects $[\![A]\!], [\![\Gamma]\!]$ in the category $\mathbb{C}$ such that:

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

Furthermore, 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 $\mathbb{C}$.

Aside 1.3.1 (Syntax highlighting)**.** *In this dissertation, I have made use of some simple syntax highlighting in order to make some relations easier to parse by eye. There is not a specific meaning attached to each colour, but as a guide: green indicates an assumption, blue a conclusion, and type annotations within terms are purple. For example:* $\texttt{Assumption} \vdash \lambda x : A.\, \texttt{term} : \texttt{Type}$.

$$\Gamma \vdash v : A \mapsto \mathbb{C}([\![\Gamma]\!], [\![A]\!])$$

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 $[\![\Gamma \vdash v_1 : A]\!]$ and $[\![\Gamma \vdash v_2 : A]\!]$ are equal morphisms in $\mathbb{C}$ from $[\![\Gamma]\!]$ to $[\![A]\!]$.

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

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

In this dissertation, I 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 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

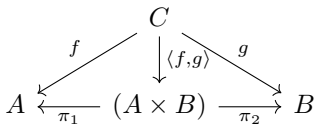
A category is *cartesian closed* if it has a terminal object, products for all pairs of objects, and exponentials. These concepts will be explained below.

Terminal Object

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

Products

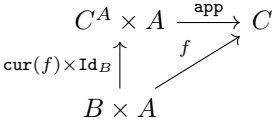
A *binary product* of a pair of objects $A, B \in \mathtt{obj}\ \mathbb{C}$ consists of an object, written $A \times B$, with morphisms $\pi_1 : A \times B \to A$, $\pi_2 : A \times B \to B$ such that for any other object C and morphisms, $f : C \to A$, $g : C \to B$ there exists a unique morphism $\langle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



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

Exponentials

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



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

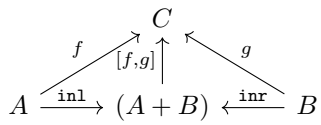
Diagonal and Twist Morphisms

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

$$\tau_{A,B} : (A \times B) \to (B \times A) = \langle \pi_2, \pi_1 \rangle$$
$$\delta_A : A \to (A \times A) = \langle \texttt{Id}_A, \texttt{Id}_A \rangle$$

2.1.2 Co-Product

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



2.1.3 Functors

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

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

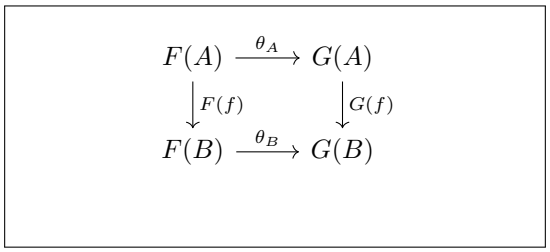


Figure 2.1: Naturality of a natural transformation

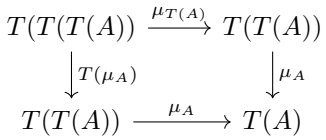


Figure 2.2: Monad Associativity Laws

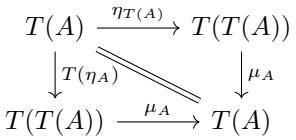


Figure 2.3: Monad Left- and Right-Unit laws

2.1.4 Natural Transformations

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

2.1.5 Monad

A *monad* consists of a functor, $T : \mathbb{C} \to \mathbb{C}$, 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 the *unit* natural transformation $\eta_A : A \to T(A)$, which is used to treat pure values as an effectful expression. The second is the *join* natural transformation, $\mu_A : T(T(A)) \to T(A)$, 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 $(M, \cdot, \mathtt{1})$ where $\cdot : M \times M \to M$, is associative $(a \cdot (b \cdot c) = (a \cdot b) \cdot c)$, and has an identity $a \cdot \mathtt{1} = a = \mathtt{1} \cdot a$. In this dissertation, I 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.

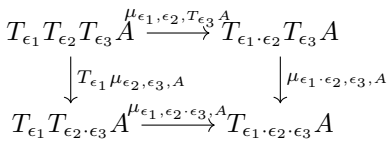


Figure 2.4: Associativity of a graded monad

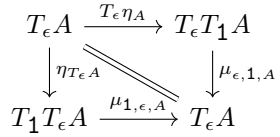


Figure 2.5: Left- and Right- Units of a graded monad

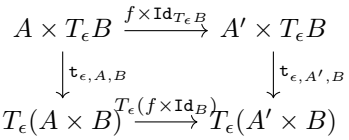


Figure 2.6: Left Naturality of Graded Tensor Strength

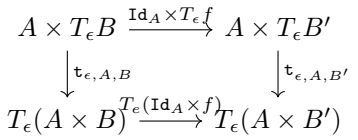


Figure 2.7: Right Naturality of Graded Tensor Strength

2.1.7 Graded Monad

A *graded monad* is a generalisation of a monad to be indexed by a monoidal algebra $(E, \cdot, \mathtt{1})$. It consists of an endofunctor indexed by elements in the monoid, $T : (E, \cdot, \mathtt{1}) \to [\mathbb{C}, \mathbb{C}]$, and a pair of indexed natural transformations. Firstly, there is the unit natural transformation to the monad functor indexed by the identity $\eta : \mathtt{Id} \to T_{\mathtt{1}}$. The second natural transformation is a generalisation of join which composes nested instances of the indexed functor $\mu_{\epsilon_1,\epsilon_2,} : T_{\epsilon_1} T_{\epsilon_2} \to T_{\epsilon_1 \cdot \epsilon_2}$. 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: $\mathtt{t}_{A,B} : A \times TB \to T(A \times B)$, which is required to have well-defined interactions with the graded monad morphisms and the product-reordering natural transformation $\alpha_{A,B,C} = \langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle \rangle : ((A \times B) \times C) \to (A \times (B \times C))$, 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, such as in Figure 2.13 in Section 2.2. 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. An adjunction consists of functors $F : C \to D$, and $G : D \to C$ and a pair of natural transformations, known respectively as the unit and co-unit: $\eta_A : A \to G(FA)$ in $\mathbb{C}$ and $\epsilon_B : F(GB) \to B$ in $\mathbb{D}$, such that $\epsilon_{FA} \circ F(\eta_A) = \mathtt{Id}_{FA}$ and $G(\epsilon_B) \circ \eta_{FB} = \mathtt{Id}_{GB}$. 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.

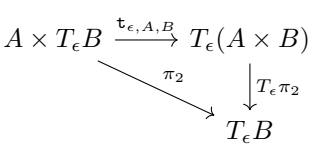


Figure 2.8: Tensor Strength Unitor Law

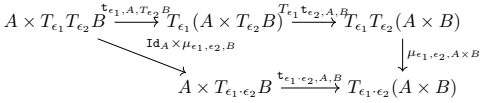


Figure 2.9: How the tensor-strength natural transformation commutes with the join natural transformation.

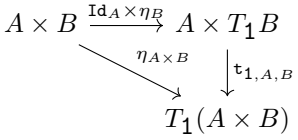


Figure 2.10: How the tensor-strength natural transformation commutes with the unit natural transformation

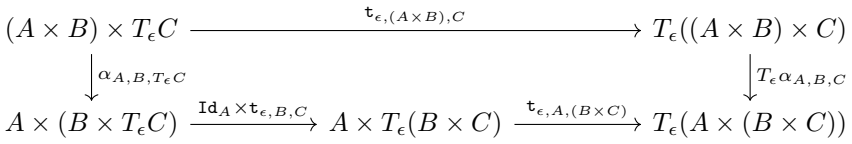


Figure 2.11: Tensor strength commutes with the reordering natural transformation.

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

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, $f : B \to A$, are contravariantly mapped to functors, written $f^* : \mathbb{C}(A) \to \mathbb{C}(B)$ 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 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 $\mathtt{Id}^*_A(B) = B \in \mathtt{obj}\ \mathbb{C}(A)$, and $\mathtt{Id}^*$ is the identity functor. For example, we may use the category of cartesian closed categories and functors that preserve the cartesian closed structure, ($CCCat$) indexed by a partial order, $\mathbb{P}$:

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

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 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 type environments, 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 type environments.

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

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 $(\Gamma \times \mathtt{M}A)$ to $\mathtt{M}(\Gamma \times A)$.

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 $[\![\Gamma \vdash \texttt{condition}\colon \texttt{Bool}]\!] : \Gamma \to [\![\texttt{Bool}]\!]$, $[\![\Gamma \vdash \texttt{if\_true}\colon A]\!] : \Gamma \to A$, and $[\![\Gamma \vdash \texttt{if\_false}\colon A]\!] : \Gamma \to A$ to form a morphism $\Gamma \to A$. If we have a co-product $\Gamma + \Gamma$, and a morphism mapping `Bool`s to $\Gamma + \Gamma$, we could use the fold morphism $[[\![\texttt{if\_true}]\!], [\![\texttt{if\_false}]\!]]$ to achieve the required composite morphism. It turns out that using exponentials, as seen in the denotation of the (If) type rule in Figure 3.9, we can factor out the Γ to instead use a co-product $\mathtt{1} + \mathtt{1}$ instead of $\Gamma + \Gamma$. It now a natural choice to use $\mathtt{1} + \mathtt{1}$ as $[\![\texttt{Bool}]\!]$. 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 $\mathtt{M}(-)$ 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.

$$\text{(Return)}\frac{\Gamma \vdash v\colon A}{\Gamma \vdash \texttt{return}\ v : \mathtt{M}A} \quad \text{(Bind)}\frac{\Gamma \vdash v_1\colon \mathtt{M}A \qquad \Gamma, x\colon A \vdash v_2\colon \mathtt{M}B}{\Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2 : \mathtt{M}B} \tag{2.1}$$

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 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 systematically possible 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 $(E, \cdot, \leq, \mathtt{1})$. The set E gives the various effects that can be produced, $\mathtt{1}$ 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 `M` with the effect ϵ that is produced when the corresponding expression is evaluated. Furthermore, 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 transforms the type rules in Equation 2.1 to 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]).

$$\text{(Return)}\frac{\Gamma \vdash v : A}{\Gamma \vdash \texttt{return}\ v : \texttt{M}_{\texttt{1}} A} \quad \text{(Bind)}\frac{\Gamma \vdash v_1 : \texttt{M}_{\epsilon_1} A \qquad \Gamma, x : A \vdash v_2 : \texttt{M}_{\epsilon_2} B}{\Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2 : \texttt{M}_{\epsilon_1 \cdot \epsilon_2} B} \tag{2.2}$$

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, is not applicable. A solution for this is to introduce a notion of subtyping, using a subtyping relation $\leq:$. 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.

$$\text{(If)}\frac{\Gamma \vdash \texttt{condition} : \texttt{Bool} \qquad \Gamma \vdash \texttt{if\_true} : A \qquad \Gamma \vdash \texttt{if\_false} : A}{\Gamma \vdash \texttt{if condition then if\_true else if\_false} : A} \tag{2.3}$$

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

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 G, such as types, effects, type constructors, or some other language specific feature, we can add parametric polymorphism over G by allowing G variables to occur in expressions and introducing expression terms to indicate when these G variables are introduced and when they are eliminated. 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 type environments Γ. A term or type is well formed (written $\Phi \vdash f : F$) 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.

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

System F
Types are extended with type variables and parameterisation.

$$A ::= \ldots \mid \alpha \mid \forall \alpha.A$$

Values are extended with generalisation and specialisation terms respectively.

$$v ::= \ldots \mid \Lambda \alpha.t \mid t\ A$$

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

If we can model the language without the polymorphic terms in Equation 2.4, at each specific F environment, then we can instantiate a collection of categories, each of which models the language at a given F environment. This collection of categories (which we call *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:

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

Here, $\mathtt{k}^A$ is one of collection of constants, and A ranges over the types:

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

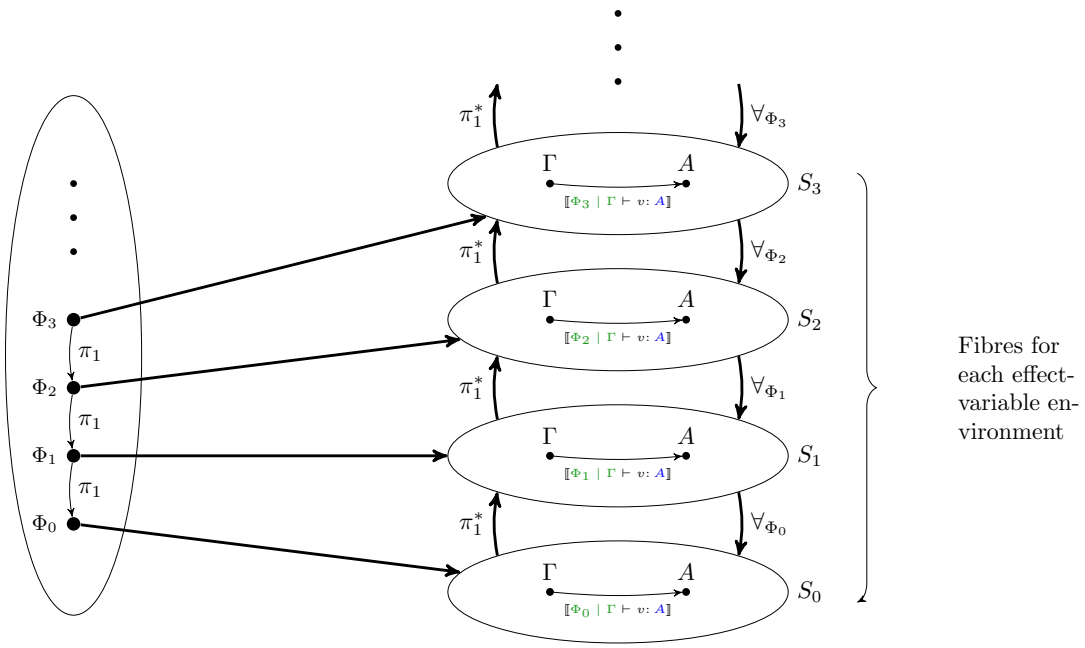


Figure 2.15: Diagram of the structure of an indexed category for modelling a polymorphic language. Thick arrows between categories represent functors and thinner arrows withing categories represent internal morphisms. The left-hand category is the base category.

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

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

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

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 IO action, as can be seen in the extended example in Figure 2.20.

$$\texttt{if}_A\ \texttt{true}\ \texttt{then}\ v_1\ \texttt{else}\ v_2 \rightsquigarrow v_1$$
$$\texttt{if}_A\ \texttt{true}\ \texttt{then}\ v_2\ \texttt{else}\ v_2 \rightsquigarrow v_2$$

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

$$\texttt{do}\ x \leftarrow \texttt{return}\ v_1\ \ \texttt{in}\ v_2 \rightsquigarrow v_2[v_1/x]$$
$$\texttt{do}\ x \leftarrow v\ \texttt{in}\ \texttt{return}\ x \rightsquigarrow v$$
$$\texttt{do}\ x \leftarrow v_1\ \texttt{in}\ (\texttt{do}\ y \leftarrow v_2\ \texttt{in}\ v_3\ ) \leftrightsquigarrow \texttt{do}\ y \leftarrow (\texttt{do}\ y \leftarrow v_1\ \texttt{in}\ v_2\ )\ \texttt{in}\ v_3$$

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

2.4 Polymorphic Effect Calculus

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

$$v ::= .. \mid \Lambda\alpha.v \mid v\ \epsilon \qquad A, B, C ::= ... \mid \forall\alpha.A \mid \mathtt{M}_\epsilon A \qquad \epsilon ::= e \mid \alpha \mid \epsilon \cdot \epsilon$$

Figure 2.21: The extension to the grammar of EC which yields PEC

2.4.1 Type System

Environments

As mentioned before, expressions 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 $(E, \cdot, \mathtt{1}, \leq)$ over ground elements e as in EC (Section 2.3). However, we now want to be able to reason about effects that contain polymorphic variables. As a result, we need to be able reason about effect expressions in a symbolic fashion. As a result, we construct a new partially monoid over effects with variables in a effect-variable environment Φ, written $(E_\Phi, \cdot_\Phi, \leq_\Phi, \mathtt{1})$. The set E_Φ consists of expressions generated from the grammar given in Figure 2.21 where α ranges over variables in Φ.

In order to define the $\cdot_\Phi$ operator, we need to consider a simple, extensional equational equivalence: $\Phi \vdash \epsilon_1 \approx \epsilon_2 : \mathtt{Effect}$, defined in Figure 2.22. Specifically, we define polymorphic effects $\epsilon \in E_\Phi$ to be equivalence classes up to this equivalence.

$$\Phi \vdash \epsilon_1 \approx \epsilon_2 : \mathtt{Effect} \Leftrightarrow \forall \sigma \in \mathtt{Ground}.\ \epsilon_1[\sigma] = \epsilon_2[\sigma] \tag{2.5}$$

Figure 2.22: Equational equivalence of effects with respect to an effect-variable environment. `Ground` is defined to be the set of ground-effect substitutions.

Now we can define the monoid operator.

Definition 2.4.1 (Polymorphic Effect Monoid)**.**

$$\epsilon_1 \cdot_\Phi \epsilon_2 = \epsilon_3 \Leftrightarrow \forall \sigma \in \mathtt{Ground}.\ (\epsilon_1[\sigma] \cdot \epsilon_2[\sigma] = \epsilon_3[\sigma])$$

This choice of operator preserves any identity or annihilator elements in ground monoid. To define the $\leq_\Phi$ relation, we must again take inspiration from the substitution-based equivalence.

Definition 2.4.2 (Polymorphic Subeffecting)**.**

$$\epsilon_1 \leq_\Phi \epsilon_2 \Leftrightarrow \forall \sigma \in \mathit{Ground}.\ \epsilon_1[\sigma] \leq \epsilon_2$$

This subeffecting relation preserves ground effect structures such as a *top* effect, which has all effects as subeffects. Where it is obvious from the context, I use $\cdot$ instead of $\cdot_\Phi$.

Types

As stated, types are now generated by the grammar below. Note that γ ranges over the ground types of a particular instantiation of PEC.

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

Type Environments

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

Definition 2.4.3 (Domain Function on Type Environments)**.**

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

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. We write $\alpha \in \Phi$ if α appears in the list represented by Φ.

The `Ok` predicate (Figure 2.23) on effect-variable environments asserts that the effect-variable environment does not contain any duplicated effect variables. Using this, we can define the well-formedness relation on effects, $\Phi \vdash \epsilon : \mathtt{Effect}$ (Figure 2.24). In short, this relation ensures that effects only reference variables that are in the effect-variable environment. This is equivalent to saying that $\Phi \vdash \epsilon : \mathtt{Effect}$ means that $\epsilon \in E_\Phi$.

$$\text{(E-Env-Nil)}\ \frac{}{\diamond\ \mathtt{Ok}} \qquad \text{(E-Env-Extend)}\ \frac{\Phi\ \mathtt{Ok}}{\Phi, \alpha\ \mathtt{Ok}}\ (\text{if } \alpha \notin \Phi)$$

Figure 2.23: The `Ok` predicate on effect-variable environments

$$\text{(E-Ground)}\frac{\Phi\ \texttt{Ok}}{\Phi \vdash e\colon \texttt{Effect}} \qquad \text{(E-Var)}\frac{\Phi, \alpha\ \texttt{Ok}}{\Phi, \alpha \vdash \alpha\colon \texttt{Effect}}$$

$$\text{(E-Weaken)}\frac{\Phi \vdash \alpha\colon \texttt{Effect}}{\Phi, \beta \vdash \alpha\colon \texttt{Effect}}(\text{if } \alpha \neq \beta, \beta \notin \Phi) \qquad \text{(E-Compose)}\frac{\Phi \vdash \epsilon_1\colon \texttt{Effect} \qquad \Phi \vdash \epsilon_2\colon \texttt{Effect}}{\Phi \vdash \epsilon_1 \cdot \epsilon_2\colon \texttt{Effect}}$$

Figure 2.24: The well-formedness relation on effects

The well-formedness of effects can be used to a similar well-typed-relation on types, $\Phi \vdash A\colon \texttt{Type}$, which asserts that all effects in the type are well formed (Figure 2.25). Finally, we can derive the a well-formedness of type environments, $\Phi \vdash \Gamma\ \texttt{Ok}$, which ensures that all types in the environment are well formed (Figure 2.26).

$$\text{(T-Ground)}\frac{\Phi\ \texttt{Ok}}{\Phi \vdash \gamma\colon \texttt{Type}} \qquad \text{(T-Fn)}\frac{\Phi \vdash A\colon \texttt{Type} \qquad \Phi \vdash B\colon \texttt{Type}}{\Phi \vdash A \to B\colon \texttt{Type}}$$

$$\text{(T-Effect)}\frac{\Phi \vdash A\colon \texttt{Type} \qquad \Phi \vdash \epsilon\colon \texttt{Effect}}{\Phi \vdash \texttt{M}_\epsilon A\colon \texttt{Type}} \qquad \text{(T-Quantification)}\frac{\Phi, \alpha \vdash A\colon \texttt{Type}}{\Phi \vdash \forall \alpha.A\colon \texttt{Type}}$$

Figure 2.25: The well-formedness relation on types

$$\text{(Env-Nil)}\frac{}{\Phi \vdash \diamond\ \texttt{Ok}} \qquad \text{(Env-Extend)}\frac{\Phi \vdash \Gamma\ \texttt{Ok} \quad \Phi \vdash A\colon \texttt{Type}}{\Phi \vdash \Gamma, x\colon A\ \texttt{Ok}}(\text{if } x \notin \texttt{dom}(\Gamma))$$

Figure 2.26: The well-formedness relation on type environments

Subtyping

We assume that the set of ground types (γ) has a subtyping partial order relation $\leq:_\gamma$. 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 antisymmetric, transitive and reflexive (Figure 2.27).

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

Figure 2.27: Ground subtyping rules.

Using this ground type relation, we can then construct a subtyping relation between pairs of types A, B with respect to an effect-variable environment Φ (Figure 2.28).

$$\text{(S-Ground)}\frac{A \leq:_\gamma B}{A \leq:_\Phi B} \quad \text{(S-Fn)}\frac{A \leq:_\Phi A' \qquad B' \leq:_\Phi B}{A' \to B' \leq:_\Phi A \to B}$$

$$\text{(S-Quantification)}\frac{A \leq:_\Phi A'}{\forall\alpha.A \leq:_{\Phi,\alpha} \forall\alpha.A'}(\text{if } \alpha \notin \Phi) \quad \text{(S-Effect)}\frac{A \leq:_\Phi B \qquad \epsilon_1 \leq_\Phi \epsilon_2}{\mathtt{M}_{\epsilon_1} A \leq:_\Phi \mathtt{M}_{\epsilon_2} B}$$

Figure 2.28: The extension of ground subtyping to the full subtyping relation.

Since the ground subtyping and subeffecting relations are partial orders, 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 $\Phi \vdash A : \mathtt{Type}$ then $\Phi \vdash B : \mathtt{Type}$. This last property comes about because the subeffecting relation $\leq_\Phi$ only holds between effects in E_Φ, which are well formed by definition.

Type Rules

We define a fairly standard set of type rules on the language (Figure 2.29).

$$\text{(Const)}\frac{\Phi \vdash \Gamma\ \mathtt{Ok} \qquad \Phi \vdash A : \mathtt{Type}}{\Phi \mid \Gamma \vdash \mathtt{k}^A : A} \quad \text{(Unit)}\frac{\Phi \vdash \Gamma\ \mathtt{Ok}}{\Phi \mid \Gamma \vdash () : \mathtt{Unit}} \quad \text{(True)}\frac{\Phi \vdash \Gamma\ \mathtt{Ok}}{\Phi \mid \Gamma \vdash \mathtt{true} : \mathtt{Bool}} \quad \text{(False)}\frac{\Phi \vdash \Gamma\ \mathtt{Ok}}{\Phi \mid \Gamma \vdash \mathtt{false} : \mathtt{Bool}}$$

$$\text{(Var)}\frac{\Phi \vdash \Gamma, x : A\ \mathtt{Ok}}{\Phi \mid \Gamma, x : A \vdash x : A} \quad \text{(Weaken)}\frac{\Phi \mid \Gamma \vdash x : A \qquad \Phi \vdash B : \mathtt{Type}}{\Phi \mid \Gamma, y : B \vdash x : A}(\text{if } x \neq y, y \notin \mathtt{dom}(\Gamma)) \quad \text{(Fn)}\frac{\Phi \mid \Gamma, x : A \vdash v : B}{\Phi \mid \Gamma \vdash \lambda x : A.v : A \to B}$$

$$\text{(Subtype)}\frac{\Phi \mid \Gamma \vdash v : A \qquad A \leq:_\Phi B}{\Phi \mid \Gamma \vdash v : B} \quad \text{(Effect-Gen)}\frac{\Phi, \alpha \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash \Lambda\alpha.v : \forall\alpha.A} \quad \text{(Effect-Spec)}\frac{\Phi \mid \Gamma \vdash v : \forall\alpha.A \qquad \Phi \vdash \epsilon : \mathtt{Effect}}{\Phi \mid \Gamma \vdash v\ \epsilon : A[\epsilon/\alpha]}$$

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

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

Figure 2.29: The type rules for PEC

Ok Lemmas

Lemma 2.4.1 (Ok Lemma on Type Environments)**.** *If* $\Phi \mid \Gamma \vdash v : A$ *then* $\Phi \vdash \Gamma$ `Ok`*.*

Proof: If $\Phi \vdash \Gamma, x : A$ `Ok` then by inverting (See Figure 2.30) the `Ok` rule, we have $\Phi \vdash \Gamma$ `Ok`

Only the type rule (*Env-Extend*) 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 $\Phi \vdash \Gamma$ `Ok`. All non-axiom derivations preserve the `Ok` property.

□

Lemma 2.4.2 (Ok Lemma on Effect-Variable Environments)**.** *A second group of lemmas is that each of the judgments* $\Phi \vdash \epsilon :$ `Effect`*,* $\Phi \vdash A :$ `Type`*, and* $\Phi \vdash \Gamma$ `Ok` *implies* Φ `Ok`*.*

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

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

Secondly, in the case of the well formedness relation on types, the only leaf rule is (*T-Ground*). If $\Phi \vdash \gamma :$ `Type`, then by inversion, Φ `Ok`. Similarly to the (*E-Compose*) rule for effects, the (*T-Fn*) rule's preconditions have the same effect-environment, so Φ `Ok` holds by inversion and induction. By inversion on the (*T-Effect*) rule, $\Phi \vdash \epsilon :$ `Effect` so Φ `Ok`. Finally the (*T-Quantification*) rule preserves `Ok` in a similar way to the (*E-Weaken*) rule above.

Finally, if $\Phi \vdash \Gamma, x : A$ `Ok` then, by inversion, $\Phi \vdash A :$ `Type` and $\Phi \vdash \Gamma$ `Ok`. Since Γ is finite, $\Phi \vdash \diamond$ `Ok`. By inversion of the (*Env-Nil*) rule, Φ `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 $\Phi \mid \Gamma \vdash v : B$, then we can case split over the type rules. If $\Phi \mid \Gamma \vdash v : B$ was generated by the (*Apply*) rule (2.6), then by inspecting the rule, we can infer that there exist two subterms $\Phi \mid \Gamma \vdash v_1 : A \to B$, $\Phi \mid \Gamma \vdash v_2 : A$ such that $v = v_1\ v_2$.

$$\text{(Apply)}\frac{\Phi \mid \Gamma \vdash v_1 : A \to B \qquad \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash v_1\ v_2 : B} \tag{2.6}$$

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

Figure 2.30: An explanation of the concept of inversion.

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 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 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 $\mathtt{1} + \mathtt{1}$, which models the type of booleans. With the addition of some extra category structure to handle subtyping, which I 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 explain in more detail later 3.4. Semantics for similar languages to EC have also been given using an S-category-style construct, such as by Katsumata [2].

In order to 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 the category, $\mathbb{C}$, to be able to model subtyping.

If a cartesian closed category has a morphism to represent each instance of ground subtyping relation $A \leq :_\gamma B$ and a natural transformation to represent each instance of the subeffect relation $\epsilon_1 \leq \epsilon_2$, then using the cartesian closed structure of the category, all instances of the general subtyping relation can be modelled. This is explained in Section 3.4.

From this point onwards, I 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 $\mathtt{1} + \mathtt{1}$ as an *S-Category* (Semantic Category).

Definition 3.1.1 (S-Category for an EC Instance)**.** *An S-category for a given instance of the effect*

calculus is a category that satisfies the following requirements.

i. The category is cartesian closed.

ii. The category has a co-product for the terminal object ($\mathbf{1}+\mathbf{1}$).

iii. The category has a graded monad indexed by the effect monoid of the EC instance.

iv. The category has an object $[\![\gamma]\!]$ for each ground type γ in EC.

v. The category has a point morphism $[\![k^A]\!] : \mathbf{1} \to [\![A]\!]$ for each constant in EC. The object $[\![A]\!]$ is defined in section 3.4.

vi. For each instance of the ground subtyping relation, $A \leq :_\gamma B$, there exists a morphism between the objects representing the ground types A and B.

vii. For each instance of $\epsilon_1 \leq \epsilon_2$, there should exist a natural transformation $[\![\epsilon_1 \leq \epsilon_2]\!] : T_{\epsilon_1} \to T_{\epsilon_2}$ such that it has interactions with the graded monad as specified in Figures 3.1, 3.2.

A full derivation and proof of soundness of the semantics of the Effect Calculus can be found online on my github repository[1] as it is too long to include here and many of the concepts are explained 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.

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

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

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

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

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-variable 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 $\mathbb{C}([\![\Phi]\!])$ are called *fibres*. To extend our mapping $\mathbb{C}(-)$ to a contravariant functor, as required in Section 2.1.10, it needs to map morphisms in the base category $\mathbb{C}$ to functors between fibres. In particular, all functors derived from $\mathbb{C}$ must be *S-preserving* (Definition 3.2.1). Thus, each morphism $\theta : [\![\Phi']\!] \to [\![\Phi]\!]$ in $\mathbb{C}$ should induce an S-preserving, re-indexing functor $\theta^* : \mathbb{C}([\![\Phi]\!]) \to \mathbb{C}([\![\Phi']\!])$ between the fibres.

[1] `https://github.com/Al153/Part3Project/blob/master/diss/OnlineECSemantics.pdf`

Definition 3.2.1 (S-Preserving Functor)**.** *A functor F preserves the properties of S-categories if it preserves each of the features within an S-category (Definition 3.1.1). 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 B.1.*

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

In order to form denotations of effects, ϵ, which are well formed with respect to an effect-variable environment, 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, $\mathtt{1}$ and binary products. We can form finite products as so: $U^0 = \mathtt{1}$ 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 $\mathtt{1}$, 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 $[\![e]\!] : \mathbb{C}(\mathtt{1}, U)$ for each ground effect e. Furthermore, each fibre should contain and object $[\![\gamma]\!]$ for each ground type γ. Finally, for each constant, $\mathtt{k}^A$, there should exist a morphism in each fibre: $[\![\mathtt{k}^A]\!] : \mathtt{1} \to A$. These last two requirements are satisfied by the fibres all being S-categories.

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

Our penultimate requirement is that the re-indexing functor induced by $\pi_1 : I \times U \to I$ (that is $\pi_1^* : \mathbb{C}(I) \to \mathbb{C}(I \times U)$) has a right adjoint, denoted by $\forall_I : \mathbb{C}(I \times U) \to \mathbb{C}(I)$. 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, this adjunction should satisfy the Beck-Chevalley condition, as seen in Figure 3.3. That is $\theta^* \circ \forall_I = \forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*$, and the natural transformation $\overline{(\theta \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon})}$ between these functors is equal to the identity natural transformation. This allows us to commute the re-indexing functors with the quantification functor.

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

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

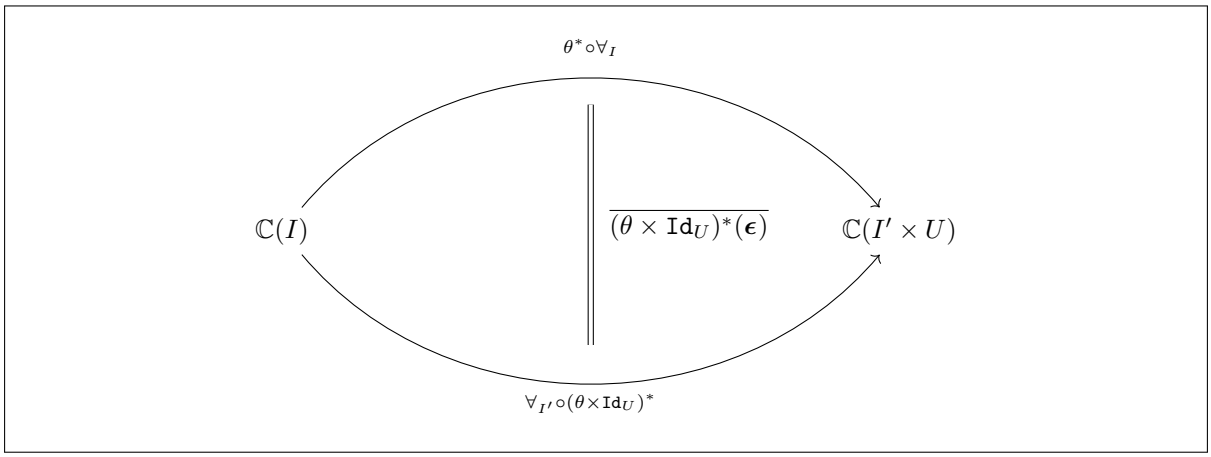


Figure 3.3: A functor diagram of the Beck-Chevalley condition

adjunction[2].

$$
\begin{aligned}
\theta^*\boldsymbol{\eta}_A : \quad & \theta^* A \to \theta^* \circ \forall_I \circ \pi_1^* A \\
\theta^*\boldsymbol{\eta} =& \overline{(\theta \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon}_{\pi_1^*})} \circ \theta^*\boldsymbol{\eta} \\
=& (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)(\boldsymbol{\epsilon}_{\pi_1^*}) \circ \boldsymbol{\eta}_{(\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*) \circ \pi_1^*} \circ \theta^*\boldsymbol{\eta} \\
=& (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)(\boldsymbol{\epsilon}_{\pi_1^*}) \circ \boldsymbol{\eta}_{\theta^* \circ \forall_I \circ \pi_1^*} \circ \theta^*\boldsymbol{\eta} \\
=& (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)(\boldsymbol{\epsilon}_{\pi_1^*}) \circ (\theta^* \circ \forall_I \circ \pi_1^*)\boldsymbol{\eta} \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*} \\
=& (\theta^* \circ \forall_I)(\boldsymbol{\epsilon}_{\pi_1^*} \circ \pi_1^*\boldsymbol{\eta}) \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*} \\
=& (\theta^* \circ \forall_I)(\mathtt{Id}) \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*} \\
=& \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*}
\end{aligned}
$$

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

$$
\begin{aligned}
\theta^*(\overline{f}) &= \theta^*(\forall_I(f) \circ \eta_A) && (3.1)\\
&= \theta^*(\forall_I(f)) \circ \theta^*(\eta_A) && (3.2)\\
&= (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)f \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^* A} && (3.3)\\
&= \overline{(\theta \times \mathtt{Id}_U)^* f} && (3.4)\\
& && (3.5)
\end{aligned}
$$

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 (3.5.1, 3.5.2) 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

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

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 in Theorems 3.5.4, 3.5.5, and 3.5.6. From this, we can also look at the action of weakening and substituting effect-variable environments on the subtyping relation between types in Theorems 3.5.7, 3.5.8.

The next step is to use these substitution theorems to formalise the action of substitution and weakening of the effect-variable environments on terms in Theorems 3.5.9, 3.5.10. 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 in Theorems 3.5.14, 3.5.15.

Separately, we prove that all derivable denotations for a typing relation instance, $\Phi \mid \Gamma \vdash v : A$ have the same denotation (Section 3.6). 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.

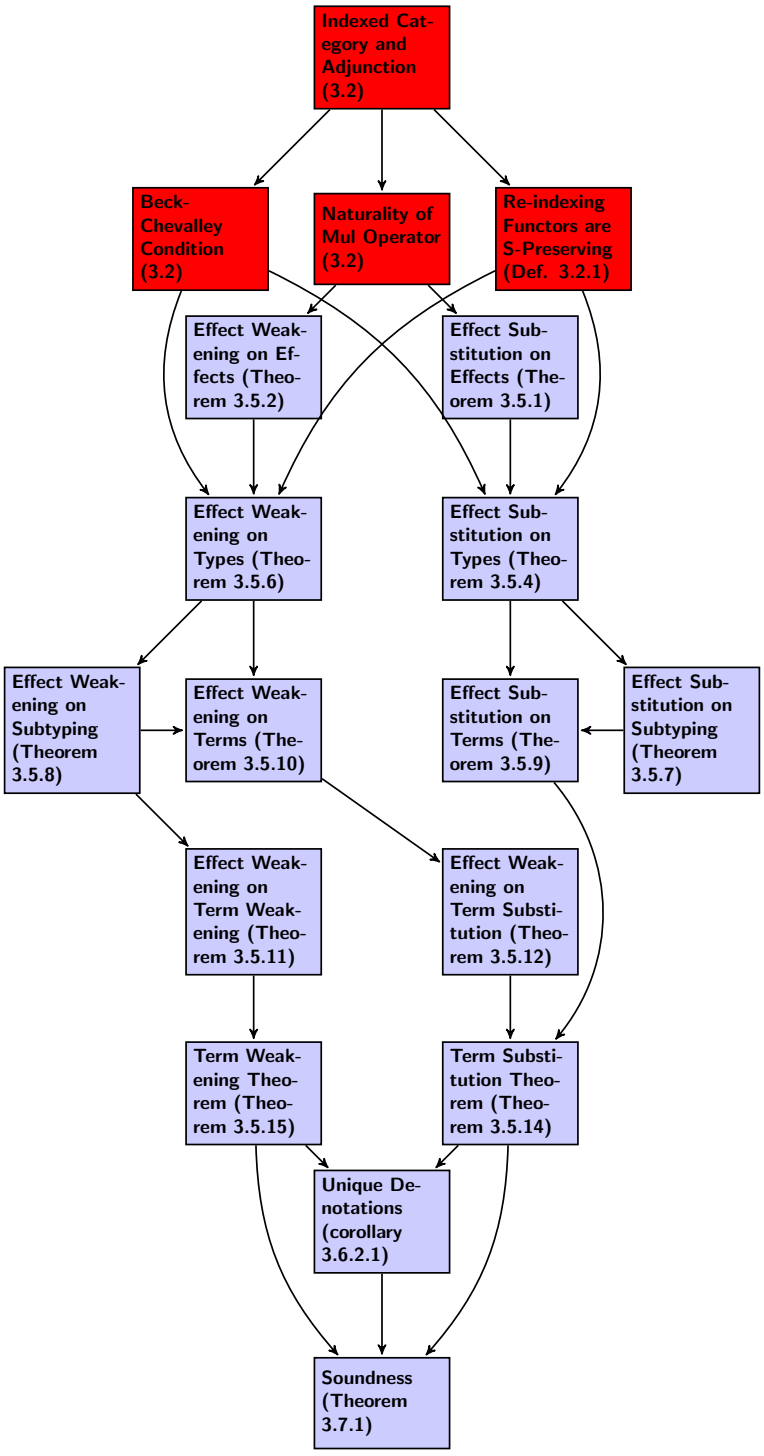


Figure 3.4: A road map of the proof dependencies. Assumptions are in red, theorems in blue.

3.4 Denotations

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

$$[\![\diamond]\!] = \mathtt{1} \qquad [\![\Phi, \alpha]\!] = [\![\Phi]\!] \times U$$

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 Crole ([7]), I write I as shorthand for $[\![\Phi]\!]$. As stated earlier, the denotation of a well-formed effect is a morphism $[\![\Phi \vdash \epsilon : \mathtt{Effect}]\!] : I \to U$ in $\mathbb{C}$. The denotations for effects are shown in Figure 3.5.

$$[\![\Phi \vdash e : \mathtt{Effect}]\!] = [\![\epsilon]\!] \circ \langle\rangle_I : I \to U \qquad [\![\Phi, \alpha \vdash \alpha : \mathtt{Effect}]\!] = \pi_2 : I \times U \to U$$

$$[\![\Phi, \beta \vdash \alpha : \mathtt{Effect}]\!] = [\![\Phi \vdash \alpha : \mathtt{Effect}]\!] \circ \pi_1 : I \times U \to U$$

$$[\![\Phi \vdash \epsilon_1 \cdot \epsilon_2 : \mathtt{Effect}]\!] = \mathtt{Mul}_I([\![\Phi \vdash \epsilon_2 : \mathtt{Effect}]\!], [\![\Phi \vdash \epsilon_1 : \mathtt{Effect}]\!]) : I \to U$$

Figure 3.5: The denotations of effects in the base category

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. These type denotations are given in Figure 3.6

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 $\mathtt{1} + \mathtt{1}$.

$$[\![\Phi \vdash \mathtt{Unit} : \mathtt{Type}]\!] = \mathtt{1} \qquad [\![\Phi \vdash \mathtt{Bool} : \mathtt{Type}]\!] = \mathtt{1} + \mathtt{1} \qquad [\![\Phi \vdash \gamma : \mathtt{Type}]\!] = [\![\gamma]\!]$$

$$[\![\Phi \vdash A \to B : \mathtt{Type}]\!] = ([\![\Phi \vdash B : \mathtt{Type}]\!])^{([\![\Phi \vdash A : \mathtt{Type}]\!])}$$

$$[\![\Phi \vdash \mathtt{M}_\epsilon A : \mathtt{Type}]\!] = T_{[\![\Phi \vdash \epsilon : \mathtt{Effect}]\!]}[\![\Phi \vdash A : \mathtt{Type}]\!] \qquad [\![\Phi \vdash \forall \alpha . A : \mathtt{Type}]\!] = \forall_I([\![\Phi, \alpha \vdash A : \mathtt{Type}]\!])$$

Figure 3.6: The denotations of type expressions within the appropriate fibre.

By using the terminal objects and products present in each fibre, we can now derive denotations of type environments. $[\![\Phi \vdash \Gamma \ \mathtt{Ok}]\!]$ should be an object in the fibre $\mathbb{C}(I)$ induced by Φ. Specific denotations are given in Figure 3.7

$$[\![\Phi \vdash \diamond \ \mathtt{Ok}]\!] = 1 \qquad [\![\Phi \vdash \Gamma, x\colon A \ \ \mathtt{Ok}]\!] = ([\![\Phi \vdash \Gamma \ \ \mathtt{Ok}]\!] \times [\![\Phi \vdash A\colon \mathtt{Type}]\!])$$

Figure 3.7: Denotations of type environments within the appropriate fibre

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. $[\![A \leq:_\Phi B]\!] \in \mathbb{C}(I)(A, B)$. 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 $[\![\epsilon_1 \leq_\Phi \epsilon_2]\!] : T_{\epsilon_1} \to T_{\epsilon_2}$. Specific subtyping denotations are given in Figure 3.8.

$$\begin{aligned}
[\![\gamma_1 \leq:_\Phi \gamma_2]\!] &= [\![\gamma_1 \leq:_\gamma \gamma_2]\!] \\
[\![A \to B \leq:_\Phi A' \to B']\!] &= [\![B \leq:_\Phi B']\!]^{A'} \circ B^{[\![A' \leq:_\Phi A]\!]} \\
[\![\mathtt{M}_{\epsilon_1} A \leq:_\Phi \mathtt{M}_{\epsilon_2} B]\!] &= [\![\epsilon_1 \leq_\Phi \epsilon_2]\!] \circ T_{\epsilon_1} [\![A \leq:_\Phi B]\!] \\
[\![\forall \alpha.A \leq:_\Phi \forall \alpha.B]\!] &= \forall_I [\![A \leq:_{\Phi,\alpha} B]\!]
\end{aligned}$$

Figure 3.8: The denotations of subtyping relations

This finally gives us the ability to express the denotations of well-typed terms in an effect-variable environment, Φ as morphisms in the fibre induced by Φ, $\mathbb{C}(I)$. Writing Γ_I and A_I for $[\![\Phi \vdash \Gamma \ \ \mathtt{Ok}]\!]$ and $[\![\Phi \vdash A\colon \mathtt{Type}]\!]$, we can derive $[\![\Phi \mid \Gamma \vdash v\colon A]\!]$ as a morphism in $\mathbb{C}(I)(\Gamma_I, A_I)$. Since each fibre is an S-category, for each constant, $\mathtt{k}^A$, there exists $[\![\mathtt{k}^A]\!] : 1 \to A_I$ in $\mathbb{C}(I)$. These denotations can be seen in Figure 3.9.

$$\text{(Unit)}\ \frac{\Phi \vdash \Gamma\ \texttt{Ok}}{[\![\Phi \mid \Gamma \vdash \texttt{()}:\texttt{Unit}]\!] = \langle\rangle_\Gamma : \Gamma \to \mathtt{1}} \qquad \text{(Const)}\ \frac{\Phi \vdash \Gamma\ \texttt{Ok}}{[\![\Phi \mid \Gamma \vdash \mathtt{k}^A : A]\!] = [\![\mathtt{k}^A]\!] \circ \langle\rangle_\Gamma : \Gamma \to [\![A]\!]}$$

$$\text{(True)}\ \frac{\Phi \vdash \Gamma\ \texttt{Ok}}{[\![\Phi \mid \Gamma \vdash \texttt{true}:\texttt{Bool}]\!] = \texttt{inl} \circ \langle\rangle_\Gamma : \Gamma \to [\![\texttt{Bool}]\!] = \mathtt{1} + \mathtt{1}}$$

$$\text{(False)}\ \frac{\Phi \vdash \Gamma\ \texttt{Ok}}{[\![\Phi \mid \Gamma \vdash \texttt{false}:\texttt{Bool}]\!] = \texttt{inr} \circ \langle\rangle_\Gamma : \Gamma \to [\![\texttt{Bool}]\!] = \mathtt{1} + \mathtt{1}}$$

$$\text{(Var)}\ \frac{\Phi \vdash \Gamma\ \texttt{Ok}}{[\![\Phi \mid \Gamma, x{:}\,A \vdash x{:}\,A]\!] = \pi_2 : \Gamma \times A \to A} \qquad \text{(Weaken)}\ \frac{f = [\![\Phi \mid \Gamma \vdash x{:}\,A]\!] : \Gamma \to A}{[\![\Phi \mid \Gamma, y{:}\,B \vdash x{:}\,A]\!] = f \circ \pi_1 : \Gamma \times B \to A}$$

$$\text{(Fn)}\ \frac{f = [\![\Phi \mid \Gamma, x{:}\,A \vdash v{:}\,B]\!] : \Gamma \times A \to B}{[\![\Phi \mid \Gamma \vdash \lambda x{:}\,A.v : A \to B]\!] = \texttt{cur}(f) : \Gamma \to (B)^A}$$

$$\text{(Subtype)}\ \frac{f = [\![\Phi \mid \Gamma \vdash v{:}\,A]\!] : \Gamma \to A \qquad g = [\![A \leq :_\Phi B]\!]}{[\![\Phi \mid \Gamma \vdash v{:}\,B]\!] = g \circ f : \Gamma \to B} \qquad \text{(Return)}\ \frac{f = [\![\Phi \mid \Gamma \vdash v{:}\,A]\!]}{[\![\Phi \mid \Gamma \vdash \texttt{return}\ v : \mathtt{M}_1\,A]\!] = \eta_A \circ f}$$

$$\text{(If)}\ \frac{f = [\![\Phi \mid \Gamma \vdash v{:}\,\texttt{Bool}]\!] : \Gamma \to \mathtt{1} + \mathtt{1} \qquad g = [\![\Phi \mid \Gamma \vdash v_1 : \mathtt{M}_\epsilon A]\!] \qquad h = [\![\Phi \mid \Gamma \vdash v_2 : \mathtt{M}_\epsilon A]\!]}{[\![\Phi \mid \Gamma \vdash \texttt{if}_{\epsilon,A}\ v\ \texttt{then}\ v_1\ \texttt{else}\ v_2 : \mathtt{M}_\epsilon A]\!] = \texttt{app} \circ (([\texttt{cur}(g \circ \pi_2), \texttt{cur}(h \circ \pi_2)] \circ f) \times \texttt{Id}_\Gamma) \circ \delta_\Gamma : \Gamma \to T_\epsilon A}$$

$$\text{(Bind)}\ \frac{f = [\![\Phi \mid \Gamma \vdash v_1 : \mathtt{M}_{\epsilon_1} A : \Gamma \to T_{\epsilon_1} A]\!] \qquad g = [\![\Phi \mid \Gamma, x{:}\,A \vdash v_2 : \mathtt{M}_{\epsilon_2} B]\!] : \Gamma \times A \to T_{\epsilon_2} B}{[\![\Phi \mid \Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B]\!] = \mu_{\epsilon_1,\epsilon_2,B} \circ T_{\epsilon_1} g \circ \mathtt{t}_{\Gamma,A,\epsilon_1} \circ \langle \texttt{Id}_\Gamma, f\rangle : \Gamma \to T_{\epsilon_1 \cdot \epsilon_2} B}$$

$$\text{(Apply)}\ \frac{f = [\![\Phi \mid \Gamma \vdash v_1 : A \to B]\!] : \Gamma \to (B)^A \qquad g = [\![\Phi \mid \Gamma \vdash v_2 : A]\!] : \Gamma \to A}{[\![\Phi \mid \Gamma \vdash v_1\ v_2 : B]\!] = \texttt{app} \circ \langle f, g\rangle : \Gamma \to B}$$

$$\text{(Effect-Gen)}\ \frac{f = [\![\Phi, \alpha \mid \Gamma \vdash v{:}\,A]\!] : \mathbb{C}(I \times U, W)(\Gamma, A)}{[\![\Phi \mid \Gamma \vdash \Lambda\alpha.A : \forall\epsilon.A]\!] = \overline{f} : \mathbb{C}(I)(\Gamma, \forall_I(A))}$$

$$\text{(Effect-Spec)}\ \frac{g = [\![\Phi \mid \Gamma \vdash v : \forall\alpha.A]\!] : \mathbb{C}(I)(\Gamma, \forall_I(A)) \qquad h = [\![\Phi \vdash \epsilon : \texttt{Effect}]\!] : \mathbb{C}(I, U)}{[\![\Phi \mid \Gamma \vdash v\ \epsilon : A[\epsilon/\alpha]]\!] = \langle \texttt{Id}_I, h\rangle^*(\boldsymbol{\epsilon}_{[\![\Phi,\beta \vdash A[\beta/\alpha]:\texttt{Type}]\!]}) \circ g : \mathbb{C}(I)(\Gamma, A[\epsilon/\alpha])} \tag{3.6}$$

Figure 3.9: The denotations of terms in PEC

The least intuitive of these denotations is that of effect-application (*Effect-Spec*) in Equation 3.6. As explained later in Section 3.5 the re-indexing functor represents application of the substitution $[\epsilon/\beta]$. By doing type analysis on the co-unit morphism $\boldsymbol{\epsilon}_{[\![\Phi,\beta \vdash A[\beta/\alpha]:\texttt{Type}]\!]}$, as in Figure 3.10 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-variable environment. By applying the substitution re-indexing functor in Figure 3.11, 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 3.12.

$$
\begin{aligned}
\boldsymbol{\epsilon}_{[\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]} = {} & \overbrace{\mathtt{Id}_{\forall_I([\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!])}} \\
: \quad & \pi_1^*\forall_I([\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]) \to [\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!] \\
: \quad & \pi_1^*[\![\Phi\vdash\forall\beta.A[\beta/\alpha]:\mathtt{Type}]\!] \to [\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!] \\
: \quad & \pi_1^*[\![\Phi\vdash\forall\alpha.A:\mathtt{Type}]\!] \to [\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!] \\
: \quad & [\![\Phi,\beta\vdash\forall\alpha.A:\mathtt{Type}]\!] \to [\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]
\end{aligned}
$$

Figure 3.10: Analysis of the type of the co-unit natural transformation

$$
\begin{aligned}
\langle\mathtt{Id}_I,h\rangle^*(\boldsymbol{\epsilon}_{[\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]}) : \quad & [\![\Phi\vdash\forall\alpha.A:\mathtt{Type}]\!] \to [\![\Phi\vdash A[\beta/\alpha][\epsilon/\beta]:\mathtt{Type}]\!] \\
: \quad & [\![\Phi\vdash\forall\alpha.A:\mathtt{Type}]\!] \to [\![\Phi\vdash A[\epsilon/\alpha]:\mathtt{Type}]\!]
\end{aligned}
$$

Figure 3.11: Analysis of the type of the re-indexed co-unit

$$
\begin{array}{ccccc}
[\![\Phi,\beta\vdash\forall\alpha.A:\mathtt{Type}]\!] & & \xrightarrow{\boldsymbol{\epsilon}_{[\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]}} & & [\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!] \\
\Gamma & \xrightarrow{\quad g\quad} & \forall_I(A) & \xrightarrow{\langle\mathtt{Id}_I,h\rangle^*(\boldsymbol{\epsilon}_{[\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]})} & A[\epsilon/\alpha]
\end{array}
$$

Figure 3.12: Composition of the effect-specialisation denotation

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

Substitutions and weakenings are two distinct ways of manipulating effect or type environments. 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 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 define and state the denotations of specific substitutions and weakenings, upon both the effect-variable environment and the type environments.

In this dissertation, substitutions and weakenings come in two flavours: weakening and substitution of the effect-variable environment and weakening and substitution of the type environments. 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.

Aside 3.5.1 (Weakening Proofs)**.** *The structure of weakening theorems and their proofs are often rather similar to their respective substitution theorem. In these cases, I have placed the example cases of the weakening proof in Appendix A.*

3.5.1 Substitution and Weakening on Effects

The first family of theorems is that of weakening and substitution of the effect-variable environment. Weakenings are a relation between effect-variable environments $\omega\colon \Phi' \triangleright \Phi$, that are defined in Figure 3.13. We can define the denotation of an effect-environment weakening as a morphism in the base category: $[\![\omega\colon \Phi' \triangleright \Phi]\!] : I' \to I$. The denotations are defined in Figure 3.21.

$$\text{(E-Id)}\frac{\Phi\ \texttt{Ok}}{\iota\colon \Phi \triangleright \Phi} \qquad \text{(E-Project)}\frac{\omega\colon \Phi' \triangleright \Phi}{\omega\pi\colon (\Phi', \alpha) \triangleright \Phi}(\text{if } \alpha \notin \Phi') \qquad \text{(E-Extend)}\frac{\omega\colon \Phi' \triangleright \Phi}{\omega\times\colon (\Phi', \alpha) \triangleright (\Phi, \alpha)}(\text{if } \alpha \notin \Phi')$$

Figure 3.13: Effect weakening definitions

$$\begin{aligned}
[\![\iota\colon \Phi \triangleright \Phi]\!] &= \texttt{Id}_I : I \to I \\
[\![\omega\pi\colon \Phi', \alpha \triangleright \Phi]\!] &= [\![\omega\colon \Phi' \triangleright \Phi]\!] \circ \pi_1 : I' \times U \to I \\
[\![\omega\times\colon \Phi', \alpha \triangleright \Phi, \alpha]\!] &= ([\![\omega\colon \Phi' \triangleright \Phi]\!] \times \texttt{Id}_U) : I' \times U \to I \times U
\end{aligned}$$

Figure 3.14: Effect weakening denotations

Substitutions are also an inductively defined relation between effect-variable environments, with inductively defined denotations. Finite substitutions may be represented as a snoc-list of variable-effect pairs. The substitution relation matches substitutions to the environments it operates between. The relation instance $\Phi' \vdash \sigma\colon \Phi$ means that σ is a substitution from Φ to Φ'. It is defined inductively using rules in Figure 3.15.

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

$$\text{(E-Nil)}\frac{\Phi' \ \texttt{Ok}}{\Phi' \vdash \diamond : \diamond} \qquad \text{(E-Extend)}\frac{\Phi' \vdash \sigma : \Phi \qquad \Phi' \vdash \epsilon : \texttt{Effect}}{\Phi' \vdash (\sigma, \alpha := \epsilon) : (\Phi, \alpha)} (\text{if } \alpha \notin \Phi)$$

Figure 3.15: Effect substitution definition

We can define the action of substitutions on effects, as in Figure 3.16. Furthermore, Figure 3.16 also gives the denotation of substitutions. The denotation of an effect-environment substitutions, $[\![\Phi' \vdash \sigma : \Phi]\!]$, is a morphism $I' \to I$ in the base category.

Action

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

Denotations

$$\begin{aligned}
[\![\Phi' \vdash \diamond : \diamond]\!] &= \langle \rangle_I : \mathbb{C}(I', 1) \\
[\![\Phi' \vdash (\sigma, \alpha := \epsilon) : \Phi, \alpha]\!] &= \langle [\![\Phi' \vdash \sigma : \Phi]\!], [\![\Phi \vdash \epsilon : \texttt{Effect}]\!] \rangle \\
&\quad : \mathbb{C}(I', I \times U)
\end{aligned}$$

Figure 3.16: The action on effects and denotations of an effect substitution

The general construction for effect-variable substitution allows us to define, in particular, the identity substitution, $\Phi \vdash \texttt{Id}_\Phi : \Phi$ and the singleton substitution, $\Phi \vdash [\epsilon/\alpha] : \Phi, \alpha$ in Figure 3.17. By inspection, we can also obtain the denotations of these special-case substitutions. Note that in particular, the denotation of the identity substitution completes our intuition for the denotation of the (*Effect-Spec*) rule in Section 3.4. These substitutions will form a useful shorthand later. Finally, it will be useful to examine how the substitutions generated by extending the effect-variable environment, such as in the case of polymorphic types, relate to the original substitution.

$$\begin{aligned}
\texttt{Id}_\diamond &= \diamond \\
\texttt{Id}_{\Phi,\alpha} &= (\texttt{Id}_\Phi, \alpha := \alpha) \\
[\epsilon/\alpha] &= (\texttt{Id}_\Phi, \alpha := \epsilon)
\end{aligned}$$

$$\begin{aligned}
[\![\Phi \vdash \texttt{Id}_\Phi : \Phi]\!] &= \texttt{Id}_I \\
[\![\Phi \vdash [\epsilon/\alpha] : (\Phi, \alpha)]\!] &= \langle \texttt{Id}_I, [\![\Phi \vdash \epsilon : \texttt{Effect}]\!] \rangle
\end{aligned}$$

Figure 3.17: Special case substitutions and their denotations.

Definition 3.5.1 (Freshness)**.** *We define $\alpha \# \sigma$ to mean that α does not occur in the domain or any of the substitution expression of σ.*

Theorem 3.5.1 (Effect Substitution on Effects). *If* $\Phi \vdash \epsilon : \mathit{Effect}$ *and* $\Phi' \vdash \sigma : \Phi$ *then:*

i. $\Phi' \vdash \sigma(\epsilon) : \mathit{Effect}$

ii. Writing σ *for* $[\![\Phi' \vdash \sigma : \Phi]\!]$, $[\![\Phi' \vdash \sigma(\epsilon) : \mathit{Effect}]\!] = [\![\Phi \vdash \epsilon : \mathit{Effect}]\!] \circ \sigma$

Proof: The proof of this depends on the naturality of $\mathtt{Mul}_I$ and inversion to narrow down case splitting on the structure of the effect-variable environments. It proceeds by induction on the definition of denotations of effects given in Figure 3.5.

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

$$\begin{aligned}
[\![\Phi \vdash e : \mathtt{Effect}]\!] \circ \sigma &= [\![e]\!] \circ \langle\rangle_I \circ \sigma \\
&= [\![e]\!] \circ \langle\rangle_{I'} \\
&= [\![\Phi' \vdash e : \mathtt{Type}]\!]
\end{aligned}$$

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

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

Case E-Compose: We make use of the naturality of $\mathtt{Mul}_I$ and induction upon the subterms.

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

□

The weakening theorem proceeds similarly.

Theorem 3.5.2 (Effect Weakening on Effects). *If* $\Phi \vdash \epsilon : \mathit{Effect}$ *and* $\omega : \Phi' \triangleright \Phi$ *then:*

i. $\Phi' \vdash \epsilon : \mathit{Effect}$

ii. *Writing* ω *for* $[\![\omega : \Phi' \triangleright \Phi]\!]$, $[\![\Phi' \vdash \epsilon : \mathit{Effect}]\!] = [\![\Phi \vdash \epsilon : \mathit{Effect}]\!] \circ \omega$.

Proof: This proof also depends on the naturality of $\mathtt{Mul}_I$ and case splitting on the structure of ω. Some cases can be found in Appendix A.1

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* $[\![(\Phi', \alpha) \vdash (\sigma, \alpha := \alpha) : (\Phi, \alpha)]\!] = ([\![\Phi' \vdash \sigma : \Phi]\!] \times \mathit{Id}_U)$

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

□

3.5.2 Substitution and Weakening on Typing

We can now 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 action of a morphism $\theta : I' \to I$ between objects in the base category on structure in the fibres $\mathbb{C}(I)$, we should simply apply the associated re-indexing functor (As defined in Section 3.2) $\theta^* : \mathbb{C}(I) \to \mathbb{C}(I')$ 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 (σ) have the actions given in Figure 3.18 on types and type-environments.

Action of Effect Substitution

$$\gamma[\sigma] = \gamma$$
$$(A \to B)[\sigma] = (A[\sigma]) \to (B[\sigma])$$
$$(\mathtt{M}_\epsilon A)[\sigma] = \mathtt{M}_{\sigma(\epsilon)}(A[\sigma])$$
$$(\forall \alpha . A)[\sigma] = \forall \alpha . (A[\sigma]) \quad \text{If } \alpha \# \sigma$$

$$\diamond[\sigma] = \diamond$$
$$(\Gamma, x : A)[\sigma] = (\Gamma[\sigma], x : (A[\sigma]))$$

Figure 3.18: The action of effect substitutions on types and type environments.

Theorem 3.5.4 (Effect Substitution on Types)**.** *If* $\Phi \vdash A : \mathit{Type}$ *and* $\Phi' \vdash \sigma : \Phi$*, then:*

i. $\Phi' \vdash A[\sigma] : \mathit{Type}$

ii. $[\![\Phi' \vdash A[\sigma] : \mathit{Type}]\!] = \sigma^* [\![\Phi \vdash A : \mathit{Type}]\!]$

Proof: By induction on the derivation of $[\![\Phi \vdash A : \mathtt{Type}]\!]$ (Figure 3.6) and the fact that σ^* is S-preserving. These specific cases show the dependencies on Theorem 3.5.1, the Beck-Chevalley condition and the extension lemma (Lemma 3.5.3).

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

$$\begin{aligned}
\sigma^* [\![\Phi \vdash \mathtt{M}_\epsilon A : \mathtt{Type}]\!] &= \sigma^* (T_{[\![\Phi \vdash \epsilon : \mathtt{Effect}]\!]} [\![\Phi \vdash A : \mathtt{Type}]\!]) \\
&= T_{[\![\Phi \vdash \epsilon : \mathtt{Effect}]\!] \circ \sigma} \sigma^* ([\![\Phi \vdash A : \mathtt{Type}]\!]) \\
&= T_{[\![\Phi' \vdash \sigma(\epsilon) : \mathtt{Effect}]\!]} [\![\Phi' \vdash A[\sigma] : \mathtt{Type}]\!] \\
&= [\![\Phi' \vdash (\mathtt{M}_\epsilon A)[\sigma] : \mathtt{Type}]\!]
\end{aligned}$$

Case T-Quantification: This case makes use of the Beck-Chevalley condition and the fact that $[\![(\Phi', \alpha) \vdash (\sigma, \alpha := \alpha) : (\Phi, \alpha)]\!] = \sigma \times \mathtt{Id}_U$, which we can induct upon.

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

□

Similarly, we can extend the effect-substitution theorem to type environments.

Theorem 3.5.5 (Effect Substitution on Type Environments)**.** *If* $\Phi \vdash \Gamma \ \mathit{Ok}$*, then:*

i. $\Phi' \vdash \Gamma[\sigma] \ \mathit{Ok}$

ii. $[\![\Phi' \vdash \Gamma[\sigma] \ \mathit{Ok}]\!] = \sigma^* [\![\Phi \vdash \Gamma \ \mathit{Ok}]\!]$.

Proof: By induction on the derivation of $[\![\Phi \vdash \Gamma \ \mathtt{Ok}]\!]$ (Figure 3.7) whilst making use of the S-preserving property of the re-indexing functor.

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

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

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

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

□

The effect-weakening theorem on types and type environments is formulated analogously.

Theorem 3.5.6 (Effect Weakening on Types and Type Environments)**.** *If* $\omega{:}\,\Phi' \triangleright \Phi$ *and* $\Phi \vdash A{:}\,\mathtt{Type}$ *then:*

i. $\Phi' \vdash A{:}\,\mathtt{Type}\ [\![\Phi' \vdash A{:}\,\mathtt{Type}]\!] = \omega^*[\![\Phi \vdash A{:}\,\mathtt{Type}]\!]$

ii. $\omega{:}\,\Phi' \triangleright \Phi$ *and* $\Phi \vdash \Gamma \ \ \mathtt{Ok}$ *imply* $[\![\Phi' \vdash \Gamma \ \ \mathtt{Ok}]\!] = \omega^*[\![\Phi \vdash \Gamma \ \ \mathtt{Ok}]\!]$

Proof: The proof for types proceeds in a similar fashion the proof of effect substitution on types (Theorem 3.5.4). That is, by inducting over the derivation of $[\![\Phi \vdash A{:}\,\mathtt{Type}]\!]$, and making use of the Beck-Chevalley condition and the S-preserving property of ω^*.

The proof for type environments follows the same steps as the effect-substitution proof (Theorem 3.5.5).

□

The above theorems confirm that to model the action of an effect weakening or substitution on the denotation of a type or type environment, we simply apply the suitable re-indexing functor to the denotation object of the type.

Next, we consider the action of weakening and substitution on subtyping relations. The denotations of the subtyping relations, given in Figure 3.8, are morphisms as opposed to objects. However, the action of substitution and weakening can still be achieved by applying the appropriate re-indexing functor.

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:*

i. $A[\sigma]$ *is also a subtype of* $B[\sigma]$ *under* Φ' *(*$A[\sigma] \leq :_{\Phi'} B[\sigma]$*)*

ii. $[\![A[\sigma] \leq :_{\Phi'} B[\sigma]]\!] = \sigma^*[\![A \leq :_\Phi B]\!]$.

Proof: By rule induction over the definition of the subtype relation (Figures 2.28, 3.8), making use of S-preserving property and the effect-substitution theorem on types.

Case S-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 S-Fn: This case holds by several S-preserving properties of σ^*. σ^* preserves the exponential morphisms: $\sigma^*(\mathtt{cur}(f)) = \mathtt{cur}(\sigma^*(f))$ and $\sigma^*(\mathtt{app}) = \mathtt{app}$. 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 $\sigma^*(f \times g) = (\sigma^* f \times \sigma^* g)$.

$$\begin{aligned}
\sigma^*[\![(A \to B) \leq:_\Phi A' \to B']\!] &= \sigma^*([\![B \leq:_\Phi B']\!]^{A'} \circ B^{[\![A' \leq:_\Phi A]\!]}) \\
&= \sigma^*(\mathtt{cur}([\![B \leq:_\Phi B']\!] \circ \mathtt{app})) \circ \sigma^*(\mathtt{cur}(\mathtt{app} \circ (\mathtt{Id}_B \times [\![A' \leq:_\Phi A]\!]))) \\
&= \mathtt{cur}(\sigma^*([\![B \leq:_\Phi B']\!]) \circ \mathtt{app}) \circ \mathtt{cur}(\mathtt{app} \circ (\mathtt{Id}_B \times \sigma^*([\![A' \leq:_\Phi A]\!]))) \\
&= \mathtt{cur}([\![B[\sigma] \leq:_{\Phi'} B'[\sigma]]\!] \circ \mathtt{app}) \circ \mathtt{cur}(\mathtt{app} \circ (\mathtt{Id}_{B[\sigma]} \times [\![A'[\sigma] \leq:_{\Phi'} A[\sigma]]\!])) \\
&= [\![(A[\sigma]) \to (B[\sigma]) \leq:_{\Phi'} (A'[\sigma]) \to (B'[\sigma])]\!] \\
&= [\![(A \to B)[\sigma] \leq:_{\Phi'} (A' \to B')[\sigma]]\!]
\end{aligned}$$

□

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* $[\![A \leq:_{\Phi'} B]\!] = \omega^*[\![A \leq:_\Phi B]\!]$.

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

□

These subtyping theorems complete our understanding of how the type system is affected by the application of substitutions and weakenings.

3.5.3 Effect Substitution and Weakening on Terms

Now that we have defined the action of effect substitutions on all other components of the PEC type system, 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, as in Figure 3.19.

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

Figure 3.19: Action of effect substitution on terms

We can now formulate the substitution theorem on terms. We need to be more careful when dealing with denotations of terms. Due to subtyping, typing relations on terms may have multiple derivations. As denotations are defined inductively on these derivations, each denotation carries around an implicit derivation. In the next section, I prove that all derivations for a given type relation yield equal denotations. However, for now, we need to take more care to refer to the derivation. In the theorems in this section, I use the symbol Δ to indicate both a typing relation derivation and its denotation.

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

Proof: This proof makes use of the previous effect-substitution theorems in section 3.5, and the adjunction of quantification and the re-indexing functor of projection, π_1^*. It proceeds by induction on the typing relation (Figure 2.29) and references term denotations (Figure 3.9).

Case Effect-Gen: This case makes use of the naturality condition 3.1 and simple reductions. If Δ derives $\Phi \mid \Gamma \vdash \Lambda\alpha.v\colon \forall\alpha.A$ then by inversion, there exists Δ_1 deriving $\Phi, \alpha \mid \Gamma \vdash v\colon A$ such that $\Delta = \overline{\Delta_1}$. By the extension lemma, we can pick $\alpha \notin \Phi' \wedge \alpha \notin \Phi$ so that we have: $(\Phi', \alpha) \vdash (\sigma, \alpha\colon = \alpha)\colon (\Phi, \alpha)$. By induction, there exists Δ_1' deriving $(\Phi', \alpha) \mid \Gamma[\sigma, \alpha\colon = \alpha] \vdash v[\sigma, \alpha\colon = \alpha]\colon A[\sigma, \alpha\colon = \alpha]$, which we can simplify to $\Phi', \alpha \mid \Gamma[\sigma] \vdash v[\sigma]\colon A[\sigma]$ since α does not occur in Φ or Φ'. Hence $\Phi' \mid \Gamma[\sigma] \vdash v[\sigma]\colon (\forall\alpha.A)[\sigma]$ by the tree given in Equation 3.7 (Δ').

$$
\Delta' = (\text{Effect-Gen})\frac{\dfrac{\Delta_1'}{\Phi', \alpha \mid \Gamma[\sigma] \vdash v[\sigma]\colon A[\sigma]}}{\Phi' \mid \Gamma[\sigma] \vdash v[\sigma]\colon (\forall\alpha.A)[\sigma]} \tag{3.7}
$$

It is also the case that:

$$
\sigma \times \mathtt{Id} = [\![(\Phi', \alpha) \vdash (\sigma, \alpha\colon = \epsilon)\colon (\Phi, \alpha)]\!] \tag{3.8}
$$

So

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

Case Effect-Spec: 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 $\Phi \mid \Gamma \vdash v\ \epsilon\colon A[\epsilon/\alpha]$ then there exists Δ_1 deriving $\Phi \mid \Gamma \vdash v\colon \forall\alpha.A$ and $h = [\![\Phi \vdash \epsilon\colon \mathtt{Effect}]\!]$ such that Δ is derived from Δ_1 and h as in equations 3.9 and 3.10.

$$\Delta = (\text{Effect-Spec})\frac{\dfrac{\Delta_1}{\Phi \mid \Gamma \vdash v\colon \forall\alpha.A} \qquad \dfrac{h}{\Phi \vdash \epsilon\colon \mathtt{Effect}}}{\Phi \mid \Gamma \vdash v\ \epsilon\colon A[\epsilon/\alpha]} \tag{3.9}$$

$$\Delta = \langle \mathtt{Id}_\Gamma, h\rangle^*(\epsilon_{[\![\Phi,\beta \vdash A[\beta/\alpha]\colon \mathtt{Type}]\!]}) \circ \Delta_1 \tag{3.10}$$

So by induction and effect-substitution preserving well-formedness of effects, we have Δ_1' deriving $\Phi' \mid \Gamma[\sigma] \vdash v[\sigma]\colon (\forall\alpha.A)[\sigma]$ and $\Phi' \vdash \sigma(\epsilon)\colon \mathtt{Effect}$. Using the Effect-Spec type rule, we can construct Δ' deriving $\Phi' \mid \Gamma[\sigma] \vdash (v[\sigma])\ (\sigma(\epsilon))\colon A[\sigma][\sigma(\epsilon)/\alpha]$ from Δ_1', as in Equation 3.11. Since $\alpha\#\sigma$, we can commute the applications of substitution. As a result, Δ' also derives $\Phi' \mid \Gamma[\sigma] \vdash (v\ \epsilon)[\sigma]\colon A[\epsilon/\alpha][\sigma]$.

$$\Delta' = (\text{Effect-Spec})\frac{\dfrac{\Delta_1'}{\Phi' \mid \Gamma[\sigma] \vdash v[\sigma]\colon (\forall\alpha.A)[\sigma]} \qquad \Phi' \vdash \sigma(\epsilon)\colon \mathtt{Effect}}{\Phi' \mid \Gamma[\sigma] \vdash (v[\sigma])\ (\sigma(\epsilon))\colon A[\sigma][\sigma(\epsilon)/\alpha]} \tag{3.11}$$

So, due to the substitution theorem on effects,

$$h \circ \sigma = [\![\Phi \vdash \epsilon\colon \mathtt{Effect}]\!] \circ \sigma = [\![\Phi' \vdash \sigma(\epsilon)\colon \mathtt{Effect}]\!] = h' \tag{3.12}$$

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

$$\begin{aligned}\sigma^*\Delta &= \sigma^*(\langle \mathtt{Id}_\Gamma, h\rangle^*(\epsilon_{[\![\Phi,\beta \vdash A[\beta/\alpha]\colon \mathtt{Type}]\!]}) \circ \Delta_1) && (3.13)\\ &= (\langle \mathtt{Id}_\Gamma, h\rangle \circ \sigma)^*(\epsilon_{[\![\Phi,\beta \vdash A[\beta/\alpha]\colon \mathtt{Type}]\!]}) \circ \sigma^*(\Delta_1) && (3.14)\\ &= ((\sigma \times \mathtt{Id}_U) \circ \langle \mathtt{Id}_\Gamma, h \circ \sigma\rangle)^*(\epsilon_{[\![\Phi,\beta \vdash A[\beta/\alpha]\colon \mathtt{Type}]\!]}) \circ \Delta_1' && (3.15)\\ &= (\langle \mathtt{Id}_\Gamma, h'\rangle)^*((\sigma \times \mathtt{Id}_U)^*\epsilon_{[\![\Phi,\beta \vdash A[\beta/\alpha]\colon \mathtt{Type}]\!]}) \circ \Delta_1' && (3.16)\end{aligned}$$

Looking at the inner part of the functor application: Let

$$A = [\![\Phi, \beta \vdash A[\beta/\alpha]\colon \mathtt{Type}]\!]$$

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

Going back to the expression in Equation 3.16

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

□

Similarly, we can derive the weakening theorem on terms.

Theorem 3.5.10 (Effect Weakening on Terms)**.** *If* $\omega \colon \Phi' \triangleright \Phi$ *and* Δ *derives* $\Phi \mid \Gamma \vdash v \colon A$ *then:*

i. there exists Δ' *deriving* $\Phi' \mid \Gamma \vdash v \colon A$

ii. $\Delta' = \omega^* \Delta$.

Proof: This theorem is proved in a similar fashion to the effect substitution theorem (Theorem 3.5.9) and many of its cases are the same. Some cases of this proof can be found in Appendix A.3.

This concludes our treatment of effect-environment substitution and weakening.

3.5.4 Term Substitution and Weakening

Having analysed the manipulation of the effect-variable environment, we are now at a point to start considering weakenings and substitution of the type environments. Type environment weakenings are inductively defined inf Figure 3.20 with respect to an effect-variable environment.

$$\text{(T-Id)}\frac{\Phi \vdash \Gamma \ \texttt{Ok}}{\Phi \vdash \iota : \Gamma \triangleright \Gamma} \qquad \text{(T-Project)}\frac{\Phi \vdash \omega : \Gamma' \triangleright \Gamma \qquad \Phi \vdash A : \texttt{Type}}{\Phi \vdash \omega\pi : \Gamma, x : A \triangleright \Gamma}(\text{if } x \notin \texttt{dom}(\Gamma'))$$

$$\text{(T-Extend)}\frac{\Phi \vdash \omega : \Gamma' \triangleright \Gamma \qquad A \leq : B}{\Phi \vdash \omega\times : \Gamma', x : A \triangleright \Gamma, x : B}(\text{if } x \notin \texttt{dom}(\Gamma'))$$

Figure 3.20: Definition of the term weakening relation

The denotation of such a weakening is a morphism within the fibre category derived from the effect-variable environment: $[\![\Phi \vdash \omega : \Gamma' \triangleright \Gamma]\!] : \Gamma' \to \Gamma \in \mathbb{C}(I)$. These denotations are defined in Figure 3.21.

$$[\![\Phi \vdash \iota : \Gamma \triangleright \Gamma]\!] = \texttt{Id}_\Gamma : \Gamma \to \Gamma \in \mathbb{C}(I) \tag{3.17}$$
$$[\![\Phi \vdash \omega\pi : \Gamma', ax \triangleright \Gamma]\!] = [\![\Phi \vdash \omega : \Gamma' \triangleright \Gamma]\!] \circ \pi_1 : \Gamma' \times A \to \Gamma \tag{3.18}$$
$$[\![\Phi \vdash \omega\times : \Gamma', x : A \triangleright \Gamma, x : B]\!] = [\![\Phi \vdash \omega : \Gamma' \triangleright \Gamma]\!] \times [\![A \leq :_\Phi B]\!] : \Gamma' \times A \to \Gamma \times B \tag{3.19}$$

Figure 3.21: Denotations of Term Weakening

Similarly to the definition of effect-environment substitutions (Section 3.5.1, type-environment substitutions are also snoc-lists derived inductively with respect to an effect-variable environment inf Figure 3.22.

$$\sigma ::= \diamond \mid \sigma, x := v$$

$$\text{(T-Nil)}\frac{\Phi \vdash \Gamma' \ \texttt{Ok}}{\Phi \mid \Gamma' \vdash \diamond : \diamond} \qquad \text{(T-Extend)}\frac{\Phi \mid \Gamma' \vdash \sigma : \Gamma \qquad \Phi \mid \Gamma' \vdash v : A}{\Phi \mid \Gamma' \vdash (\sigma, x := v) : (\Gamma, x : A)}(\text{if } x \notin \texttt{dom}(\Gamma'))$$

Figure 3.22: Definition of Term Substitutions

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 $v[\sigma]$.

Definition 3.5.2 (Freshness)**.** *We write $x\#\sigma$ to indicate that x does not occur in the domain of σ or as a free variable in any of its substituted terms.*

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

Denotations of type-environment substitutions are morphisms in the appropriate fibre category: $[\![\Phi \mid \Gamma' \vdash \sigma{:}\,\Gamma]\!] : \Gamma' \to \Gamma \in \mathbb{C}(I)$ and are defined in Figure 3.23.

$$
\begin{aligned}
[\![\Phi \mid \Gamma' \vdash \diamond{:}\,\diamond]\!] &= \langle\rangle_{\Gamma'} : \quad \Gamma' \to \mathtt{1} \\
[\![\Phi \mid \Gamma' \vdash (\sigma, x{:=}v){:}\,\Gamma, x{:}\,A]\!] &= \quad \langle [\![\Phi \mid \Gamma' \vdash \Gamma{:}\,]\!], [\![\Phi \mid \Gamma' \vdash v{:}\,A]\!] \rangle : \Gamma' \to \Gamma \times A
\end{aligned}
$$

Figure 3.23: Denotations for term substitutions

In order to prove the quantification case of term-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-variable 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:*

i. $\Phi' \vdash \omega{:}\,\Gamma' \triangleright \Gamma$

ii. $[\![\Phi' \vdash \omega{:}\,\Gamma' \triangleright \Gamma]\!] = \omega_1^*[\![\Phi \vdash \omega{:}\,\Gamma' \triangleright \Gamma]\!]$.

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

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

So

$$\omega_1^*(\mathtt{Id}_{\Gamma_I}) = \mathtt{Id}_{\Gamma_{I'}} = [\![\Phi' \vdash \iota{:}\,\Gamma \triangleright \Gamma]\!] \tag{3.20}$$

Case T-Project: Then $\omega = \omega'\pi$

$$(\text{T-Project})\frac{\Phi \vdash \omega' : \Gamma' \triangleright \Gamma}{\Phi \vdash \omega\pi : \Gamma', x : A \triangleright \Gamma} \tag{3.21}$$

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

Hence

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

Case T-Extend: Then $\omega = \omega' \times$

$$(\text{T-Extend})\frac{\Phi \vdash \omega' : \Gamma' \triangleright \Gamma \qquad A \leq :_\Phi B}{\Phi \vdash \omega \times : (\Gamma', x : A) \triangleright (\Gamma, x : B)} \tag{3.22}$$

So $\omega = \omega' \times [\![A \leq :_\Phi B]\!]$

Hence

$$\begin{aligned}\omega_1^*(\omega) &= (\omega_1^*(\omega') \times \omega_1^*([\![A \leq :_\Phi B]\!]) \\ &= ([\![\Phi' \vdash \omega' : \Gamma' \triangleright \Gamma]\!] \times [\![A \leq :_{\Phi'} B]\!]) \\ &= [\![\Phi' \vdash \omega : (\Gamma', x : A) \triangleright (\Gamma, x : B)]\!]\end{aligned}$$

□

Secondly, we can state and prove 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:*

i. $\Phi' \mid \Gamma' \vdash \sigma : \Gamma$

ii. $[\![\Phi' \mid \Gamma' \vdash \sigma : \Gamma]\!] = \omega^*[\![\Phi \mid \Gamma' \vdash \sigma : \Gamma]\!]$

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

Case T-Nil: Then $\sigma = \langle\rangle_{\Gamma'_I}$, so $\omega^*(\sigma) = \langle\rangle_{\Gamma'_{I'}} = [\![\Phi' \mid \Gamma' \vdash \sigma : \Gamma]\!]$

Case T-Extend: Then $\sigma = (\sigma', x := v)$

$$
\begin{aligned}
\omega^*\sigma &= \omega * \langle \sigma', [\![\Phi \mid \Gamma \vdash v\colon A]\!] \rangle \\
&= \langle \omega^*\sigma', \omega^*[\![\Phi \mid \Gamma \vdash v\colon A]\!] \rangle \\
&= \langle [\![\Phi' \mid \Gamma' \vdash \sigma'\colon \Gamma]\!], [\![\Gamma' \mid \Phi' \vdash v\colon A]\!] \rangle \\
&= [\![\Phi' \mid \Gamma' \vdash \sigma\colon \Gamma, x\colon A]\!]
\end{aligned}
$$

□

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 in Figure 3.24.

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

$$
\begin{aligned}
[\![\Phi \mid \Gamma \vdash \mathtt{Id}_\Gamma\colon \Gamma]\!] &: \quad \Gamma \to \Gamma \\
[\![\Phi \mid \Gamma \vdash \mathtt{Id}_\Gamma\colon \Gamma]\!] &= \quad \mathtt{Id}_\Gamma \\
[\![\Phi \mid \Gamma \vdash [v/x]\colon \Gamma, x\colon A]\!] &: \quad \Gamma \to \Gamma \times A \\
[\![\Phi \mid \Gamma \vdash [v/x]\colon \Gamma, x\colon A]\!] &= \quad \langle \mathtt{Id}_\Gamma, [\![\Phi \mid \Gamma \vdash v\colon A]\!] \rangle
\end{aligned}
$$

Figure 3.24: Definition and denotation of identity and single term substitutions

By inspection, these also have simple denotations, similar to those seen in the case of effect substitutions. 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* $\Phi \mid \Gamma' \vdash \sigma\colon \Gamma$ *and* $x\#\sigma$ *then* $\Phi \mid (\Gamma', x : A) \vdash (\sigma, x\colon = x)\colon (\Gamma, x\colon A)$ *with denotation*

$$[\![\Phi \mid (\Gamma', x : A) \vdash (\sigma, x\colon = x)\colon (\Gamma, x\colon A)]\!] = [\![\Phi \mid \Gamma' \vdash \sigma\colon \Gamma]\!] \times \mathit{Id}_A$$

Proof: Makes use of the weakening on terms (theorem 3.5.15). If $\Phi \mid \Gamma' \vdash \sigma\colon \Gamma$ then $\Phi \mid (\Gamma', x : A) \vdash \sigma\colon \Gamma$ with denotation $\sigma \circ \pi_1$. □

Finally, we can move on to 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 we 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-substitution theorem is formulated as so:

Theorem 3.5.14 (Term Substitution)**.** *If* $\Phi \mid \Gamma' \vdash \sigma\colon \Gamma$ *and* Δ *is a derivation of* $\Phi \mid \Gamma \vdash v\colon A$*, then we can construct* Δ' *deriving* $\Phi \mid \Gamma' \vdash v[\sigma]\colon A$ *with denotation* $\Delta' = \Delta \circ \sigma$*.*

Proof: By induction on the derivation of Δ (Figure 2.29) using the denotations of terms (Figure 3.9). Making use of the weakening of effect-variable environments on term substitutions in the case of (*Effect-Gen*)

Case Weaken:

$$(\text{Weaken})\frac{\dfrac{\Delta_1}{\Phi \mid \Gamma'' \vdash x\colon A}}{\Phi \mid \Gamma'', y : B \vdash x\colon A} \tag{3.23}$$

By inversion, $\Gamma = \Gamma', y : B$ and we have Δ_1 deriving the tree in Equation 3.23. In addition, by inversion of the well-formedness of $\Phi \mid \Gamma' \vdash \sigma\colon \Gamma$, we can decompose σ into $(\sigma', y := v)$. Here σ' satisfies $\Phi \mid \Gamma' \vdash \sigma'\colon \Gamma''$, and the denotation of σ depends on that of σ': $[\![\Phi \mid \Gamma' \vdash \sigma\colon \Gamma]\!] = \langle [\![\Phi \mid \Gamma' \vdash \sigma'\colon \Gamma'']\!], [\![\Phi \mid \Gamma' \vdash v\colon B]\!] \rangle$. Hence by induction on Δ_1 we have Δ_1' deriving $(\Phi \mid \Gamma' \vdash x[\sigma]\colon A)$. So we can take Δ' to be Δ_1'.

Hence

$$\begin{aligned}
\Delta' &= \Delta_1' \quad \text{By definition} \\
&= \Delta_1 \circ \sigma' \quad \text{By induction} \\
&= \Delta_1 \circ \pi_1 \circ \langle \sigma', [\![\Phi \mid \Gamma' \vdash v\colon B]\!] \rangle \quad \text{By product property} \\
&= \Delta_1 \circ \pi_1 \circ \sigma \quad \text{By defintion of the denotation of } \sigma \\
&= \Delta \circ \sigma \quad \text{By defintion.}
\end{aligned}$$

Case Fn: By inversion, we have Δ_1 such that the derivation tree in Equation 3.24 holds. By induction on $(\Gamma, x\colon A)$ and Δ_1 we have Δ_1' deriving $\Phi \mid \Gamma', x : A \vdash (v[\sigma])\colon B$. Also by induction, and making use of the extension lemma, we can derive the denotation of Δ_1' from Δ_1: $\Delta_1' = \Delta_1 \circ (\sigma \times \mathtt{Id}_A)$. Furthermore, Δ_1' allows us to construct the derivation Δ', in Equation 3.25.

$$\Delta = (\text{Fn})\frac{\dfrac{D_1}{\Phi \mid \Gamma, x : A \vdash v\colon B}}{\Phi \mid \Gamma \vdash \lambda x\colon A.v : A \to B} \tag{3.24}$$

$$\Delta' = (\text{Fn})\frac{\dfrac{\Delta_1'}{\Phi \mid \Gamma', x : A \vdash (v[\sigma])\colon B}}{\Phi \mid \Gamma \vdash (\lambda x\colon A.v\ )[\sigma]\colon A \to B} \tag{3.25}$$

Hence:

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

□

The term-weakening theorem is formulated similarly.

Theorem 3.5.15 (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: This proof follows similarly to the proof of the substitution theorem (Theorem 3.5.14). Some cases can be found in Appendix A.4

3.5.5 Summary

In this section, I have stated and proved a number of theorems to do with the substitution of effect and term variables within an expression. In particular, these theorems give us a way of manipulating the denotations of terms when the effect-variable and type environments are changed. As a result, these theorems provide us with a set of tools to tackle problems such as soundness and the uniqueness do denotations.

3.6 Uniqueness of Denotations

Up until this point, due to the subtyping rule, we have had to be careful about the implicit typing derivation for every term denotation $[\![\Phi \mid \Gamma \vdash v : A]\!]$. This is because typing relations have multiple derivations, as seen in Figure 3.25. 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.

$$\text{(Subtype)}\frac{\text{(Apply)}\dfrac{\Phi \mid \Gamma \vdash v_1 : A \to B \qquad \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash v_1\ v_2 : B} \qquad B \leq :_\Phi B'}{\Phi \mid \Gamma \vdash v_1\ v_2 : B'} \qquad \text{(Apply)}\frac{\text{(Subtype)}\dfrac{\Phi \mid \Gamma \vdash v_1 : A \to B \qquad A \to B \leq :_\Phi A \to B'}{\Phi \mid \Gamma \vdash v : A \to B'} \qquad \Phi \mid \Gamma \vdash v_2 : A}{\Phi \mid \Gamma \vdash v_1\ v_2 : B'}$$

Figure 3.25: Two different derivations of the same typing relation.

To prove that all typing derivations have the same denotation, I first introduce the concept of a *reduced* typing derivation that is unique to each term and type in each effect and type 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 $[\![\Phi \mid \Gamma \vdash v : A]\!] = [\![reduce(\Phi \mid \Gamma \vdash v : A)]\!]$. 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 subtyping rule only occurs 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. Secondly, the subtyping rule must 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, the subtyping rule must 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 subtyping rule instances may be the identity subtyping rule, making use of the reflexive relation $A \leq:_\Phi A$. These rules have the effect of only introducing subtyping rule uses when it is necessary maintain syntactic correctness of the derivation.

Aside 3.6.1 (Syntactic Subtyping)**.** *If subtyping were a syntactic feature, that is if subtyping were 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 type environment. Let the trees in equations 3.26, 3.27 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 3.28).

$$\text{(Subtype)}\frac{\dfrac{\Delta}{\Phi \mid \Gamma \vdash v_1 : \mathtt{M}_{\epsilon_1} A} \qquad \text{(T-Effect)}\dfrac{A \leq:_\Phi A' \qquad \epsilon_1 \leq_\Phi \epsilon_1'}{\mathtt{M}_{\epsilon_1} A \leq:_\Phi \mathtt{M}_{\epsilon_1'} A'}}{\Phi \mid \Gamma \vdash v_1 : \mathtt{M}_{\epsilon_1'} A'} \tag{3.26}$$

$$\text{(Subtype)}\frac{\dfrac{\Delta'}{\Phi \mid \Gamma, x : A' \vdash v_2 : \mathtt{M}_{\epsilon_2} B} \qquad \text{(T-Effect)}\dfrac{B \leq:_\Phi B' \qquad \epsilon_2 \leq_\Phi \epsilon_2'}{\mathtt{M}_{\epsilon_2} B \leq:_\Phi \mathtt{M}_{\epsilon_2'} B'}}{\Phi \mid \Gamma, x : A' \vdash v_2 : \mathtt{M}_{\epsilon_2'} B'} \tag{3.27}$$

$$\text{(Subtype)}\frac{\dfrac{\Delta''}{\Phi \mid (\Gamma, x : A) \vdash v_2 : \mathtt{M}_{\epsilon_2} B} \qquad \text{(T-Effect)}\dfrac{B \leq:_\Phi B' \qquad \epsilon_2 \leq_\Phi \epsilon_2'}{\mathtt{M}_{\epsilon_2} B \leq:_\Phi \mathtt{M}_{\epsilon_2'} B'}}{\Phi \mid (\Gamma, x : A) \vdash v_2 : \mathtt{M}_{\epsilon_2'} B'} \tag{3.28}$$

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 $\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1' \cdot \epsilon_2'} B'$ can be seen in Equation 3.29.

$$\text{(Subtype)}\frac{\text{(Bind)}\dfrac{\dfrac{\Delta}{\Phi \mid \Gamma \vdash v_1 : \mathtt{M}_{\epsilon_1} A} \qquad \dfrac{\Delta''}{\Phi \mid \Gamma, x : A \vdash v_2 : \mathtt{M}_{\epsilon_2} B}}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B} \qquad \text{(T-Effect)}\dfrac{\epsilon_1 \cdot \epsilon_2 \leq_\Phi \epsilon_1' \cdot \epsilon_2' \qquad B \leq:_\Phi B'}{\mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B \leq:_\Phi \mathtt{M}_{\epsilon_1' \cdot \epsilon_2'} B'}}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1' \cdot \epsilon_2'} B'} \tag{3.29}$$

Case Effect-Gen: If Equation 3.31 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 3.30.

$$\text{(Subtype)}\frac{\text{(Effect-Gen)}\dfrac{\dfrac{\Delta}{\Phi, \alpha \mid \Gamma \vdash v : A}}{\Phi \mid \Gamma \vdash \Lambda\alpha.v : \forall\alpha.A} \qquad \forall\alpha.A \leq_{\Phi} \forall\alpha.B}{\Phi \mid \Gamma \vdash \Lambda\alpha.B : \forall\alpha.B} \tag{3.30}$$

$$\text{(Subtype)}\frac{\dfrac{\Delta}{\Phi, \alpha \mid \Gamma \vdash v : A} \qquad A \leq :_{\Phi,\alpha} B}{\Phi, \alpha \mid \Gamma \vdash v : B} \tag{3.31}$$

□

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 3.26.

The Reduce Function

Case Apply: To find the reduction of a tree ending in (*Apply*) as seen in Equation 3.32, we pattern match on the reductions of the subtrees to find trees Δ_1', Δ_2', as seen in equations 3.33 and 3.34. These trees allow us to build a reduced derivation tree as in Equation 3.35.

$$reduce((\text{Apply})\frac{\dfrac{\Delta_1}{\Phi \mid \Gamma \vdash v_1 \colon A \to B} \qquad \dfrac{\Delta_2}{\Phi \mid \Gamma \vdash v_2 \colon A}}{\Phi \mid \Gamma \vdash v_1\ v_2 \colon B}) \tag{3.32}$$

$$(\text{Subtype})\frac{\dfrac{\Delta_1'}{\Phi \mid \Gamma \vdash v_1 \colon A' \to B'} \qquad A' \to B' \leq :_\Phi A \to B}{\Phi \mid \Gamma \vdash v_1 \colon A \to B} = reduce(\Delta_1) \tag{3.33}$$

$$(\text{Subtype})\frac{\dfrac{\Delta_2'}{\Phi \mid \Gamma \vdash v \colon A'} \qquad A' \leq :_\Phi A}{\Phi \mid \Gamma \vdash v_1 \colon A} = reduce(\Delta_2) \tag{3.34}$$

$$(\text{Subtype})\frac{(\text{Apply})\dfrac{\dfrac{\Delta_1'}{\Phi \mid \Gamma \vdash v_1 \colon A' \to B'} \qquad (\text{Subtype})\dfrac{\dfrac{\Delta_2'}{\Phi \mid \Gamma \vdash v_2 \colon A''} \qquad A'' \leq :_\Phi A \leq :_\Phi A'}{\Phi \mid \Gamma \vdash v_2 \colon A'}}{\Phi \mid \Gamma \vdash v_1\ v_2 \colon B'} \qquad B' \leq :_\Phi B}{\Phi \mid \Gamma \vdash v_1\ v_2 \colon B} \tag{3.35}$$

Case Bind: To find the reduction of a tree ending in a use of the (*Bind*) rule as seen in Equation 3.36, we first reduce the left-most subtree, Δ_1 to find Δ_1', as seen in Equation 3.37.

$$reduce((\text{Bind})\frac{\dfrac{\Delta_1}{\Phi \mid \Gamma \vdash v_1 \colon \mathtt{M}_{\epsilon_1} A} \qquad \dfrac{\Delta_2}{\Phi \mid \Gamma, x \colon A \vdash v_2 \colon \mathtt{M}_{\epsilon_2} B}}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B}) \tag{3.36}$$

$$(\text{Subtype})\frac{\dfrac{\Delta_1'}{\Phi \mid \Gamma \vdash v_1 \colon \mathtt{M}_{\epsilon_1'} A'} \qquad (\text{T-Effect})\dfrac{\epsilon_1' \leq_\Phi \epsilon_1 \qquad A' \leq :_\Phi A}{\mathtt{M}_{\epsilon_1'} A' \leq :_\Phi \mathtt{M}_{\epsilon_1} A}}{\Phi \mid \Gamma \vdash v_1 \colon \mathtt{M}_{\epsilon_1} A} = reduce(\Delta_1) \tag{3.37}$$

Since $\Phi \vdash (i, \times) \colon (\Gamma, x : A') \triangleright (\Gamma, x \colon A)$ if $A' \leq :_\Phi A$, and because Δ_2 derives $\Phi \mid (\Gamma, x \colon A) \vdash v_2 \colon \mathtt{M}_{\epsilon_2} B$, there also exists a derivation Δ_3 of $\Phi \mid (\Gamma, x : A') \vdash v_2 \colon \mathtt{M}_{\epsilon_2} B$. 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 3.38.

$$(\text{Subtype})\frac{\dfrac{\Delta_3'}{\Phi \mid \Gamma, x : A' \vdash v_2 \colon \mathtt{M}_{\epsilon_2'} B'} \qquad (\text{T-Effect})\dfrac{\epsilon_1' \leq_\Phi \epsilon_2 \qquad B' \leq :_\Phi B}{\mathtt{M}_{\epsilon_1'} B' \leq :_\Phi \mathtt{M}_{\epsilon_2} B}}{\Phi \mid \Gamma, x : A' \vdash v_2 \colon \mathtt{M}_{\epsilon_2} B} = reduce(\Delta_3) \tag{3.38}$$

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:

$$(\text{Subtype})\frac{(\text{T-Effect})\dfrac{\epsilon_1' \cdot \epsilon_2' \leq_\Phi \epsilon_1 \cdot \epsilon_2 \qquad B' \leq :_\Phi B}{\mathtt{M}_{\epsilon_1' \cdot \epsilon_2'} B' \leq :_\Phi \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B} \qquad (\text{Bind})\dfrac{\dfrac{\Delta_1'}{\Phi \mid \Gamma \vdash v_1 \colon \mathtt{M}_{\epsilon_1'} A'} \qquad \dfrac{\Delta_3'}{\Phi \mid \Gamma, x : A' \vdash v_2 \colon \mathtt{M}_{\epsilon_2'} B'}}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1' \cdot \epsilon_2'} B}}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B} \tag{3.39}$$

Figure 3.26: Some cases of the reduce function.

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 $\Delta' = reduce(\Delta) \implies \Delta' = \Delta$.*

Proof: We proceed by induction over the structure of Δ, making use of the substitution and weakening theorems. We make use of the the definition of the *reduce* function as defined in Figure 3.26. We also use the definitions of Δ_1, Δ_2... from the same figure.

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

$$\begin{aligned} f &= [\![A \leq :_\Phi A']\!] : A \to A' \\ f' &= [\![A'' \leq :_\Phi A]\!] : A'' \to A \\ g &= [\![B' \leq :_\Phi B]\!] : B' \to B \end{aligned}$$

Hence

$$\begin{aligned} [\![A' \to B' \leq :_\Phi A \to B]\!] &= (g)^A \circ (B')^f \\ &= \mathtt{cur}(\mathtt{app} \circ \mathtt{app}) \circ \mathtt{cur}(\mathtt{app} \circ (\mathtt{Id} \times f)) \\ &= \mathtt{cur}(g \circ \mathtt{app} \circ (\mathtt{Id} \times f)) \end{aligned}$$

Then

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

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

Let us define the following helper morphisms:

$$\begin{aligned} f &= [\![A' \leq :_\Phi A]\!] : A' \to A \\ g &= [\![B' \leq :_\Phi B]\!] : B' \to B \\ h_1 &= [\![\epsilon_1' \leq_\Phi \epsilon_1]\!] : T_{\epsilon_1'} \to T_{\epsilon_1} \\ h_2 &= [\![\epsilon_2' \leq_\Phi \epsilon_2]\!] : T_{\epsilon_2'} \to T_{\epsilon_2} \\ h &= [\![\epsilon_1' \cdot \epsilon_2' \leq_\Phi \epsilon_1 \cdot \epsilon_2]\!] : T_{\epsilon_1' \cdot \epsilon_2'} \to T_{\epsilon_1 \cdot \epsilon_2} \end{aligned}$$

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

$$\Delta_3 = \Delta_2 \circ (\mathtt{Id}_\Gamma \times f) \tag{3.40}$$

Due to the reduction of Δ_3, we have

$$\Delta_3 = h_{2,B} \circ T_{\epsilon_2'} g \circ \Delta_3' \tag{3.41}$$

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 \mathtt{t}_{\epsilon_1,\Gamma,A} \circ \langle \mathtt{Id}_\Gamma, \Delta_1 \rangle && \text{By definition.} \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ T_{\epsilon_1} \Delta_2 \circ \mathtt{t}_{\epsilon_1,\Gamma,A} \circ \langle \mathtt{Id}_\Gamma, h_{1,A} \circ T_{\epsilon_1'} f \circ \Delta_1' \rangle && \text{By reduction of } \Delta_1. \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ T_{\epsilon_1} \Delta_2 \circ \mathtt{t}_{\epsilon_1,\Gamma,A} \circ (\mathtt{Id}_\Gamma \times h_{1,A}) \circ \langle \mathtt{Id}_\Gamma, T_{\epsilon_1'} f \circ \Delta_1' \rangle && \text{Factor out } h_1 \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ T_{\epsilon_1} \Delta_2 \circ h_{1,(\Gamma \times A)} \circ \mathtt{t}_{\epsilon_1',\Gamma,A} \circ \langle \mathtt{Id}_\Gamma, T_{\epsilon_1'} f \circ \Delta_1' \rangle && \text{Tensor strength and subeffecting } h_1 \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ h_{1,B} \circ T_{\epsilon_1'} \Delta_2 \circ \mathtt{t}_{\epsilon_1',\Gamma,A} \circ \langle \mathtt{Id}_\Gamma, T_{\epsilon_1'} f \circ \Delta_1' \rangle && \text{Naturality of } h_1 \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ h_{1,B} \circ T_{\epsilon_1'} \Delta_2 \circ \mathtt{t}_{\epsilon_1',\Gamma,A} \circ (\mathtt{Id}_\Gamma \times T_{\epsilon_1'} f) \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \text{Factor out pairing again} \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ h_{1,B} \circ T_{\epsilon_1'} (\Delta_2 \circ (\mathtt{Id}_\Gamma \times f)) \circ \mathtt{t}_{\epsilon_1',\Gamma,A'} \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \text{Tensorstrength} \\
&= \mu_{\epsilon_1,\epsilon_2,B} \circ h_{1,B} \circ T_{\epsilon_1'} (\Delta_3) \circ \mathtt{t}_{\epsilon_1',\Gamma,A'} \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \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 \mathtt{t}_{\epsilon_1',\Gamma,A'} \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \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 \mathtt{t}_{\epsilon_1',\Gamma,A'} \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \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 \mathtt{t}_{\epsilon_1',\Gamma,A'} \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \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 \mathtt{t}_{\epsilon_1',\Gamma,A'} \circ \langle \mathtt{Id}_\Gamma, \Delta_1' \rangle && \text{By naturality of } \mu \\
&= \Delta' && \text{By definition}
\end{aligned}
$$

□

$$
\text{(Fn)}\frac{\text{(Subtype)}\dfrac{\Phi \mid \Gamma, x\colon A \vdash v\colon B \qquad B \leq :_\Phi B'}{\Phi \mid \Gamma, x\colon A \vdash v\colon B'}}{\Phi \mid \Gamma \vdash \lambda x\colon A.v : A \to B'} \qquad (3.42) \qquad \text{(Subtype)}\frac{\text{(Fn)}\dfrac{\Phi \mid \Gamma, x\colon A \vdash v\colon B}{\Phi \mid \Gamma \vdash \lambda x\colon A.v : A \to B} \qquad A \to B \leq : A \to B'}{\Phi \mid \Gamma \vdash \lambda x\colon A.v : A \to B'} \qquad (3.43)
$$

Figure 3.27: Two derivations of the same type relation. The right derivation is in reduced form.

We can sum up the importance of these theorems in the corollary below.

Corollary 3.6.2.1 (Denotations are Unique)**.** *For any two derivations* Δ_1, Δ_2 *deriving* $\Phi \mid \Gamma \vdash v\colon A$*, the denotations of* Δ_1 *and* Δ_2 *are equal.*

Proof: Since reduced derivations of the typing relation are unique, the derivations $reduce(\Delta_1), reduce(\Delta_2)$ are equivalent and so have the same denotation. Since reduction preserves denotations, $\Delta_1 = reduce(\Delta_1) = reduce(\Delta_2) = \Delta_2$.

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. The desire for 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 3.28, there are the reductions which formalise how we expect the program to execute given an appropriate implementation. We give $\beta\eta$-reductions 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 3.29, 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 3.30.

$$\text{(Eq-Lambda-Beta)}\frac{\Phi \mid \Gamma, x{:}\, A \vdash v_2{:}\, B \qquad \Phi \mid \Gamma \vdash v_1{:}\, A}{\Phi \mid \Gamma \vdash (\lambda x{:}\, A.v_1\ )\ v_2 \approx v_1[v_2/x]{:}\, B} \quad \text{(Eq-Lambda-Eta)}\frac{\Phi \mid \Gamma \vdash v{:}\, A \to B}{\Phi \mid \Gamma \vdash \lambda x{:}\, A.(v\ x\ ) \approx v{:}\, A \to B}$$

$$\text{(Eq-Left-Unit)}\frac{\Phi \mid \Gamma \vdash v_1{:}\, A \qquad \Phi \mid \Gamma, x{:}\, A \vdash v_2{:}\, \mathtt{M}_\epsilon B}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow \mathtt{return}\ v_1\ \ \mathtt{in}\ v_2\ \approx v_2[v_1/x]{:}\, \mathtt{M}_\epsilon B} \quad \text{(Eq-Right-Unit)}\frac{\Phi \mid \Gamma \vdash v{:}\, \mathtt{M}_\epsilon A}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v\ \mathtt{in}\ \mathtt{return}\ x\ \ \approx v{:}\, \mathtt{M}_\epsilon A}$$

$$\text{(Eq-Associativity)}\frac{\Phi \mid \Gamma \vdash v_1{:}\, \mathtt{M}_{\epsilon_1} A \qquad \Phi \mid \Gamma, x{:}\, A \vdash v_2{:}\, \mathtt{M}_{\epsilon_2} B \qquad \Phi \mid \Gamma, y{:}\, B \vdash v_3{:}\, \mathtt{M}_{\epsilon_3} C}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ (\mathtt{do}\ y \leftarrow v_2\ \mathtt{in}\ v_3\ )\ \approx \mathtt{do}\ y \leftarrow (\mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2\ )\ \mathtt{in}\ v_3 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2 \cdot \epsilon_3} C}$$

$$\text{(Eq-Unit)}\frac{\Phi \mid \Gamma \vdash v{:}\, \mathtt{Unit}}{\Phi \mid \Gamma \vdash v \approx ()\,{:}\, \mathtt{Unit}}$$

$$\text{(Eq-If-True)}\frac{\Phi \mid \Gamma \vdash v_1{:}\, A \qquad \Phi \mid \Gamma \vdash v_2{:}\, A}{\Phi \mid \Gamma \vdash \mathtt{if}_A\ \mathtt{true}\ \mathtt{then}\ v_1\ \mathtt{else}\ v_2\ \approx v_1{:}\, A} \quad \text{(Eq-If-False)}\frac{\Phi \mid \Gamma \vdash v_2{:}\, A \qquad \Phi \mid \Gamma \vdash v_1{:}\, A}{\Phi \mid \Gamma \vdash \mathtt{if}_A\ \mathtt{false}\ \mathtt{then}\ v_1\ \mathtt{else}\ v_2\ \approx v_2{:}\, A}$$

$$\text{(Eq-If-Eta)}\frac{\Phi \mid \Gamma, x : \mathtt{Bool} \vdash v_2{:}\, A \qquad \Phi \mid \Gamma \vdash v_1{:}\, \mathtt{Bool}}{\Phi \mid \Gamma \vdash \mathtt{if}_A\ v_1\ \mathtt{then}\ v_2[\mathtt{true}/x]\ \mathtt{else}\ v_2[\mathtt{false}/x]\ \approx v_2[v_1/x]{:}\, A}$$

$$\text{(Eq-Effect-Beta)}\frac{\Phi \vdash \epsilon{:}\, \mathtt{Effect} \qquad \Phi, \alpha \mid \Gamma \vdash v{:}\, A}{\Phi \mid \Gamma \vdash (\Lambda\alpha.v\ \epsilon) \approx v[\epsilon/\alpha]{:}\, A[\epsilon/\alpha]} \quad \text{(Eq-Effect-Eta)}\frac{\Phi \mid \Gamma \vdash v{:}\, \forall\alpha.A}{\Phi \mid \Gamma \vdash \Lambda\alpha.(v\ \alpha) \approx v{:}\, \forall\alpha.A}$$

Figure 3.28: The reduction rules for PEC

$$\text{(Eq-Effect-Gen)}\frac{\Phi, \alpha \mid \Gamma \vdash v_1 \approx v_2 : A}{\Phi \mid \Gamma \vdash \Lambda\alpha.v_1 \approx \Lambda\alpha.v_2 : \forall\alpha.A} \quad \text{(Eq-Effect-Spec)}\frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : \forall\alpha.A \qquad \Phi \vdash \epsilon : \texttt{Effect}}{\Phi \mid \Gamma \vdash v_1\ \epsilon \approx v_2\ \epsilon : A[\epsilon/\alpha]}$$

$$\text{(Eq-Fn)}\frac{\Phi \mid \Gamma, x : A \vdash v_1 \approx v_2 : B}{\Phi \mid \Gamma \vdash \lambda x : A.v_1 \ \approx \lambda x : A.v_2 : A \to B} \quad \text{(Eq-Return)}\frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : A}{\Phi \mid \Gamma \vdash \texttt{return}\ v_1 \ \approx \texttt{return}\ v_2 : \texttt{M}_1 A}$$

$$\text{(Eq-Apply)}\frac{\Phi \mid \Gamma \vdash v_1 \approx v_1' : A \to B \qquad \Phi \mid \Gamma \vdash v_2 \approx v_2' : A}{\Phi \mid \Gamma \vdash v_1\ v_2 \approx v_1'\ v_2' : B} \quad \text{(Eq-Bind)}\frac{\Phi \mid \Gamma \vdash v_1 \approx v_1' : \texttt{M}_{\epsilon_1} A \qquad \Phi \mid \Gamma, x : A \vdash v_2 \approx v_2' : \texttt{M}_{\epsilon_2} B}{\Phi \mid \Gamma \vdash \texttt{do}\ x \leftarrow v_1\ \texttt{in}\ v_2 \ \approx \texttt{do}\ x \leftarrow v_1'\ \texttt{in}\ v_2' : \texttt{M}_{\epsilon_1 \cdot \epsilon_2} B}$$

$$\text{(Eq-If)}\frac{\Phi \mid \Gamma \vdash v \approx v' : \texttt{Bool} \qquad \Phi \mid \Gamma \vdash v_1 \approx v_1' : A \qquad \Phi \mid \Gamma \vdash v_2 \approx v_2' : A}{\Phi \mid \Gamma \vdash \texttt{if}_A\ v\ \texttt{then}\ v_1\ \texttt{else}\ v_2 \ \approx \texttt{if}_A\ v'\ \texttt{then}\ v_1'\ \texttt{else}\ v_2' : A} \quad \text{(Eq-Subtype)}\frac{\Phi \mid \Gamma \vdash v \approx v' : A \qquad A \leq :_\Phi B}{\Phi \mid \Gamma \vdash v \approx v' : B}$$

Figure 3.29: The congruence rules for PEC

$$\text{(Eq-Reflexive)}\frac{\Phi \mid \Gamma \vdash v : A}{\Phi \mid \Gamma \vdash v \approx v : A} \quad \text{(Eq-Symmetric)}\frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : A}{\Phi \mid \Gamma \vdash v_2 \approx v_1 : A}$$

$$\text{(Eq-Transitive)}\frac{\Phi \mid \Gamma \vdash v_1 \approx v_2 : A \qquad \Phi \mid \Gamma \vdash v_2 \approx v_3 : A}{\Phi \mid \Gamma \vdash v_1 \approx v_3 : A}$$

Figure 3.30: Rules expanding the reduction and congruence relation to an equivalence relation

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* $[\![\Phi \mid \Gamma \vdash v_1 : A]\!] = [\![\Phi \mid \Gamma \vdash v_2 : A]\!]$.

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, 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 Eq-Right-Unit: This case makes use of the right-unit monad law. Let $f = [\![\Phi \mid \Gamma \vdash v : \texttt{M}_\epsilon A]\!]$.

$$\begin{aligned} [\![\Phi \mid \Gamma \vdash \texttt{do}\ x \leftarrow v\ \texttt{in}\ \texttt{return}\ x \ : \texttt{M}_\epsilon A]\!] &= \mu_{\epsilon,\mathbf{1},A} \circ T_\epsilon(\eta_A \circ \pi_2) \circ \mathtt{t}_{\epsilon,\Gamma,A} \circ \langle \mathtt{Id}_\Gamma, f \rangle \\ &= T_\epsilon \pi_2 \circ \mathtt{t}_{\epsilon,\Gamma,A} \circ \langle \mathtt{Id}_\Gamma, f \rangle \\ &= \pi_2 \circ \langle \mathtt{Id}_\Gamma, f \rangle \\ &= f \end{aligned} \tag{3.44}$$

Case Eq-If-True: This case makes use of the co-product diagram on $\mathtt{1}+\mathtt{1}$. Let $f = [\![\Phi \mid \Gamma \vdash v_1 \colon A]\!]$ and $g = [\![\Phi \mid \Gamma \vdash v_2 \colon A]\!]$. Then we can simplify the denotation of the whole expression.

$$\begin{aligned}
[\![\Phi \mid \Gamma \vdash \mathtt{if}_A\ \mathtt{true}\ \mathtt{then}\ v_1\ \mathtt{else}\ v_2 : A]\!] &= \mathtt{app} \circ (([\mathtt{cur}(f \circ \pi_2), \mathtt{cur}(g \circ \pi_2)] \circ \mathtt{inl} \circ \langle\rangle_\Gamma) \times \mathtt{Id}_\Gamma) \circ \delta_\Gamma \\
&= \mathtt{app} \circ ((\mathtt{cur}(f \circ \pi_2) \circ \langle\rangle_\Gamma) \times \mathtt{Id}_\Gamma) \circ \delta_\Gamma \\
&= \mathtt{app} \circ (\mathtt{cur}(f \circ \pi_2) \times \mathtt{Id}_\Gamma) \circ (\langle\rangle_\Gamma \times \mathtt{Id}_\Gamma) \circ \delta_\Gamma \\
&= f \circ \pi_2 \circ \langle \langle\rangle_\Gamma, \mathtt{Id}_\Gamma \rangle \\
&= f \\
&= [\![\Phi \mid \Gamma \vdash v_1 \colon A]\!]
\end{aligned} \tag{3.45}$$

Case Eq-Effect-Beta: This case makes use of the adjunction properties of $\forall_I, \pi_1^*$. Let $h = [\![\Phi \vdash \epsilon \colon \mathtt{Effect}]\!]$, $f = [\![\Phi, \alpha \mid \Gamma \vdash v \colon A]\!]$, and $A = [\![\Phi, \alpha \vdash A[\alpha/\alpha] \colon \mathtt{Type}]\!]$. Then we can find the sub-term denotation $[\![\Phi \mid \Gamma \vdash \Lambda\alpha.v \colon \forall\alpha.A]\!] = \overline{f}$, which allows us to express the denotation of the whole expression.

$$\begin{aligned}
[\![\Phi \mid \Gamma \vdash (\Lambda\alpha.v)\ \epsilon \colon \forall\alpha.A]\!] &= \langle \mathtt{Id}_I, h \rangle^* (\boldsymbol{\epsilon}_A) \circ \overline{f} \\
&= \langle \mathtt{Id}_I, h \rangle^* (\boldsymbol{\epsilon}_A) \circ \langle \mathtt{Id}_I, h \rangle^* (\pi_1^* (\overline{f})) \quad \text{Identity functor} \\
&= \langle \mathtt{Id}_I, h \rangle^* (\boldsymbol{\epsilon}_A \circ \pi_1^* (\overline{f})) \\
&= \langle \mathtt{Id}_I, h \rangle^* (f) \quad \text{By adjunction} \\
&= [\![\Phi \mid \Gamma \vdash v[\epsilon/\alpha] \colon A[e/\alpha]]\!] \quad \text{By substitution theorem}
\end{aligned}$$

Case Eq-Effect-Eta: Let $f = [\![\Phi \mid \Gamma \vdash v \colon \forall\alpha.A]\!]$, and $A = [\![\Phi, \alpha \vdash A \colon \mathtt{Type}]\!]$. We can now express the denotation of the entire expression.

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

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

$$\begin{aligned}
\forall_{I \times U}([\![\Phi, \alpha, \beta \vdash A[\beta/\alpha] \colon \mathtt{Type}]\!]) &= \forall_I ((\pi_1 \times \mathtt{Id}_U)^* [\![\Phi, \beta \vdash A[\beta/\alpha] \colon \mathtt{Type}]\!]) \\
&= \pi_1^* \forall_I ([\![\Phi, \beta \vdash A[\beta/\alpha] \colon \mathtt{Type}]\!])
\end{aligned}$$

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

$$
\begin{aligned}
\boldsymbol{\epsilon}_{[\![\Phi,\alpha,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!]} &= \widehat{\mathtt{Id}_{\forall_{I\times U}([\![\Phi,\alpha,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!])}} \\
&= \widehat{\mathtt{Id}_{\pi_1^*\forall_I([\![\Phi,\beta\vdash A[\beta/\alpha]:\mathtt{Type}]\!])}} \\
&= \widehat{\mathtt{Id}_{\pi_1^*\forall_I A}} \\
&= \widehat{\pi_1^*(\mathtt{Id}_{\forall_I A})} \\
&= \widehat{\pi_1^*(\overline{\boldsymbol{\epsilon}_A})} \\
&= \widehat{\overline{(\pi_1 \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon}_A)}} \\
&= (\pi_1 \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon}_A)
\end{aligned}
$$

This rearrangement can now be used in conjunction with the contravariant composition of re-indexing functors to show the denotation $[\![\Phi \mid \Gamma \vdash \Lambda\alpha.(v\ \alpha):\forall\alpha.A]\!]$ is equal to the morphism f.

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

□

This completes the proof of soundness for this semantics.

3.8 Summary

In this chapter, I have introduced the indexed S-category structure, given a denotational semantics for PEC with respect to this structure, and proved that these semantics soundly model PEC. However, I have not shown whether it's possible to construct an indexed S-category, or how complex such categories might be. These questions will be addressed in the next chapter.

Chapter 4

Constructing a Model of PEC

Now we have proved that an appropriate categorical structure can model PEC, it remains to explicitly 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 is a graded monad $\mathtt{T}^0, \mu^0, \eta^0, \mathtt{t}^0$ on `Set`, and there exist subtyping functions $[\![A \leq :_\gamma B]\!] : A \to B$ for each instance of the ground subtyping relation, and natural transformations $[\![\epsilon_1 \leq_0 \epsilon_2]\!] : \mathtt{T}^0_{\epsilon_1} \to \mathtt{T}^0_{\epsilon_2}$ for each instance of the subeffecting relation. Furthermore, `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 the partially ordered monoid on ground effects: $(E, \leq_0, \mathtt{1}, \cdot)$. 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-variable environments of the same length are equivalent. For example, the terms in Equation 4.1 are α-equivalent. Hence, we can reduce an effect-variable environment to the natural number n indicating its length.

$$((\diamond, \alpha, \beta) \mid \Gamma \vdash \lambda f\colon \mathtt{Int} \to \mathtt{M}_{\alpha\cdot\beta}\mathtt{Unit}.(f\ 0) : \mathtt{M}_{\alpha\cdot\beta}\mathtt{Unit}) =_\alpha ((\diamond, \gamma, \delta) \mid \Gamma \vdash \lambda f\colon \mathtt{Int} \to \mathtt{M}_{\gamma\cdot\delta}\mathtt{Unit}.(f\ 0) : \mathtt{M}_{\gamma\cdot\delta}\mathtt{Unit}) \tag{4.1}$$

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

$$E^0 = \mathtt{1} = \{*\}$$
$$E^{n+1} = E^n \times E$$

Morphisms $E^n \to E$ 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 $e : \mathtt{1} \to E$, so the denotation of a ground effect e is the constant valued function $[\![\Phi \vdash e\colon \mathtt{Effect}]\!] = \vec{\epsilon} \mapsto e$. We also need to define a monoidal operation $\mathtt{Mul}_n : \mathtt{Eff}(E^n, E) \times \mathtt{Eff}(E^n, E) \to \mathtt{Eff}(E^n, E)$ that is natural as described in Section 3.2. A logical choice for $\mathtt{Mul}_n$ 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 $\mathbb{C}(E^n)$ to be the functor category $[E^n, \mathtt{Set}]$, where E^n is a discrete category (a category that only

The Monoidal Mul Operator

$$\texttt{Mul}_n(f,g)(\vec{\epsilon}) = (f\vec{\epsilon}) \cdot (g\vec{\epsilon})$$

This satisfies naturality of `Mul`.

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

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

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

Cartesian Closed Category

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

$$\begin{aligned}
(A \times B)\vec{\epsilon} &= (A\vec{\epsilon}) \times (B\vec{\epsilon}) \\
\texttt{1}\vec{\epsilon} &= \texttt{1} \\
(B^A)\vec{\epsilon} &= (B\vec{\epsilon})^{(A\vec{\epsilon})} \\
\pi_1\vec{\epsilon} &= \pi_1 \\
\pi_2\vec{\epsilon} &= \pi_2 \\
\texttt{app}\vec{\epsilon} &= \texttt{app} \\
\texttt{cur}(f)\vec{\epsilon} &= \texttt{cur}(f\vec{\epsilon}) \\
\langle f, g\rangle\vec{\epsilon} &= \langle f\vec{\epsilon}, g\vec{\epsilon}\rangle
\end{aligned}$$

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

contains the identity morphisms). That is, the category of functions returning an object in `Set` given n ground effects. Morphisms between objects are functions which, given a collection of ground effects, yield a morphism (function) in C. If $m \in [E^n, \texttt{Set}](A, B)$ then $m\vec{\epsilon} \in \texttt{Set}(A\vec{\epsilon}, B\vec{\epsilon})$. The fibres, $[E^n, C]$ are S-categories. This can be proved by constructing the S-category structures pointwise with respect to their parameter $\vec{\epsilon} \in E^n$ as seen in Figures 4.2 - 4.7. Full proofs of the naturality and monad laws can be found in Appendix C.2.4.

Now we need to define the required morphisms between fibres. Firstly, for any function $\theta : E^m \to E^n$ in `Eff`, there should exist an S-preserving re-indexing functor $\theta^* : [E^n, \texttt{Set}] \to [E^m, \texttt{Set}]$. 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.

The Co-Product on the Terminal Set

We can define the co-product pointwise.

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

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

Ground Types and Terms

Each ground type in the non-polymorphic calculus has a fixed denotation $[\![\gamma]\!] \in \mathtt{obj\ Set}$. The ground type in the polymorphic calculus hence has a denotation represented by the constant function.

$$\begin{aligned}[\![\gamma]\!] : \quad & E^n \to \mathtt{obj\ Set} \\ \vec{\epsilon} \mapsto \quad & [\![\gamma]\!]\end{aligned}$$

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

$$\begin{aligned}[\![\mathtt{k}^A]\!] : \quad & [E^n, \mathtt{Set}](\mathtt{1}, A) \\ \vec{\epsilon} \mapsto \quad & [\![\mathtt{k}^{A\vec{\epsilon}}]\!]\end{aligned}$$

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

Strong Graded Monad

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

$$
\begin{aligned}
\mathtt{T}^n : \quad & (E^n, \cdot, \leq_n, \mathtt{1}_n) \to [[E^n, \mathtt{Set}], [E^n, \mathtt{Set}]] \\
(\mathtt{T}^n_f A)\vec{\epsilon} = \quad & \mathtt{T}^0_{(f\vec{\epsilon})} A\vec{\epsilon} \\
(\eta^n_A)\vec{\epsilon} = \quad & \eta^0_{A\vec{\epsilon}} \\
(\mu^n_{f,g,A})\vec{\epsilon} = \quad & \mu^0_{(f\vec{\epsilon}),(g\vec{\epsilon}),(A\vec{\epsilon})} \\
(\mathtt{t}^n_{f,A,B})\vec{\epsilon} = \quad & \mathtt{t}^0_{(f\vec{\epsilon}),(A\vec{\epsilon}),(B\vec{\epsilon})}
\end{aligned}
$$

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

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

Subeffecting

Given a collection of subeffecting natural transformations in $\mathtt{Set}$,

$$[\![\epsilon_1 \leq_0 \epsilon_2]\!] : \quad \mathtt{T}^0_{\epsilon_1} \to \mathtt{T}^0_{\epsilon_2}$$

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

$$
\begin{aligned}
[\![f \leq_n g]\!] : \quad & \mathtt{T}^n_f \to \mathtt{T}^n_g \\
([\![f \leq_n g]\!]A)\vec{\epsilon} : \quad & \mathtt{T}^n_{f\vec{\epsilon}}(A\vec{\epsilon}) \to \mathtt{T}^n_{g\vec{\epsilon}}(B\vec{\epsilon}) \\
= \quad & [\![f\vec{\epsilon} \leq_0 g\vec{\epsilon}]\!](A\vec{\epsilon})
\end{aligned}
$$

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

Subtyping

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

$$
\begin{aligned}
[\![A \leq:_n B]\!] : \quad & A \to B \\
[\![A \leq:_n B]\!]\vec{\epsilon} = \quad & [\![A\vec{\epsilon} \leq:_0 B\vec{\epsilon}]\!]
\end{aligned}
$$

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

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

The Pre-composition Functor

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

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* $[E^n, \mathtt{Set}]$ *are preserved by* θ^* *in* $[E^m, \mathtt{Set}]$*. For a full list of features and their proofs, please see Appendix C.3.*

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

Case Graded Monad Functor:

$$\begin{aligned}(\theta^* \mathtt{T}^n_f A)\vec{\epsilon} &= \mathtt{T}^n_f A(\theta\vec{\epsilon}) \\ &= \mathtt{T}^0_{(f(\theta\vec{\epsilon}))}(A(\theta\vec{\epsilon})) \\ &= (\mathtt{T}^m_{(f\circ\theta)}\theta^* A)\vec{\epsilon}\end{aligned}$$

Case Terminal Object:

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

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

□

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

The Quantification Functor

$$\begin{aligned}\forall_{E^n} &: [E^{n+1}, \mathtt{Set}] \to [E^n, \mathtt{Set}] \\ \forall_{E^n}(A)\vec{\epsilon_n} &= \Pi_{\epsilon\in E} A(\vec{\epsilon_n}, \epsilon) \\ \forall_{E^n}(f)\vec{\epsilon_n} &= \Pi_{\epsilon\in E} f(\vec{\epsilon_n}, \epsilon)\end{aligned}$$

Figure 4.11: Definition of the quantification functor

The Adjunction Operations

$$
\begin{aligned}
f: &\quad \pi_1^* A \to B \\
\overline{f}: &\quad A \to \forall_{E^n} B \\
\overline{f}(\vec{\epsilon_n}) = &\quad \langle f(\vec{\epsilon_n}, \epsilon) \rangle_{\epsilon \in E}
\end{aligned}
$$

$$
\begin{aligned}
g: &\quad A \to \forall_{E^n} B \\
\widehat{g}: &\quad \pi_1^* A \to B \\
\widehat{g}(\vec{\epsilon_n}, \epsilon_{n+1}) = &\quad \pi_{\epsilon_{n+1}} \circ g(\vec{\epsilon_n})
\end{aligned}
$$

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

Unit

$$
\begin{aligned}
\boldsymbol{\eta}_A: &\quad A \to \forall_{E^n} \pi_1^* A \\
\boldsymbol{\eta}_A(\vec{\epsilon_n}) = &\quad \langle \mathtt{Id}_{A(\vec{\epsilon_n}, \epsilon)} \rangle_{\epsilon \in E}
\end{aligned}
$$

Co-Unit

$$
\begin{aligned}
\boldsymbol{\epsilon}_B: &\quad \pi_1^* \forall_{E^n} B \to B \\
\boldsymbol{\epsilon}_B(\vec{\epsilon_n}, \epsilon') = &\quad \pi_{\epsilon'} : \Pi_{\epsilon \in E} B(\vec{\epsilon_n}, \epsilon) \to B(\vec{\epsilon_n}, \epsilon')
\end{aligned}
$$

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

We can now prove that $\pi_1^* \dashv \forall_{E^n}$. To do this, we need a natural bijection between morphisms in the functor categories $[E^n, \mathtt{Set}]$ and $[E^{n+1}, \mathtt{Set}]$.

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

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 construct and that our requirements do not overconstrain potential models to the point that they are not useful for doing actual analysis. This contrasts with models for System F, which cannot be `Set`based[4].

Verifying We Have an Adjunction

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

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

So $\boldsymbol{\epsilon}_B \circ \pi_1^*(\overline{g}) = g$.

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

$$\begin{aligned}((\forall_{E^n}(\widehat{f})) \circ \boldsymbol{\eta}_A)(\vec{\epsilon_n}) &= \Pi_{\epsilon \in E}(\widehat{f}(\vec{\epsilon_n}, \epsilon)) \circ \langle \mathtt{Id}_{A(\vec{\epsilon_n}, \epsilon)} \rangle_{\epsilon \in E} \\ &= \Pi_{\epsilon \in E}(\pi_\epsilon \circ f(\vec{\epsilon_n})) \circ \langle \mathtt{Id}_{A(\vec{\epsilon_n}, \epsilon)} \rangle_{\epsilon \in E} \\ &= \langle \pi_\epsilon \circ f(\vec{\epsilon_n}) \rangle_{\epsilon \in E} \\ &= \langle \pi_\epsilon \rangle_{\epsilon \in E} \circ f(\vec{\epsilon_n}) \\ &= f(\vec{\epsilon_n})\end{aligned}$$

□

Figure 4.14: Proof that the π_1^* and $\forall_{E^n}$ functors form an adjunction, as required in 3.2

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

Proof: Firstly,

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

Secondly,

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

□

Figure 4.15: Proof that the $\pi_1^* \dashv \forall_{E^n}$ 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 constructed in simple categories such as `Set`. It is reasonable to conjecture that the reason that PEC's semantics are significantly simpler than those of other polymorphic languages, such as System F, is that PEC's polymorphism is not impredicative. Its polymorphic types do not quantify over themselves. This hypothesis could be tested by attempting to construct simple, `Set`-based models for other predicatively polymorphic languages. An example of such a language might be the lambda calculus extended with types that depend on the natural numbers.

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 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.

Appendix A

Additional Proofs

In this appendix, I provide introductions to some proofs which are too similar to proofs already in the dissertation to include. **TODO: Explain better**

A.1 Effect Weakening on Effects

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 E-Var: We do a case split on ω.

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

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

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

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

$$\llbracket \Phi, \alpha \vdash \alpha : \mathtt{Effect} \rrbracket = \pi_2 \circ \omega' \circ \pi_1 \tag{A.1}$$

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

So

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

Case E-Weaken:

$$\llbracket \Phi, \beta \vdash \alpha : \mathtt{Effect} \rrbracket \circ \omega = \llbracket \Phi \vdash \alpha : \mathtt{Effect} \rrbracket \circ \pi_1 \circ \omega \tag{A.2}$$

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

Case: $\omega = \iota$ Then $\Phi' = \Phi, \beta$ so $\omega = \mathtt{Id}_I$ So $[\![\Phi, \beta \vdash \alpha\colon \mathtt{Effect}]\!] \circ \omega = [\![\Phi' \vdash \alpha\colon \mathtt{Effect}]\!]$

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

$$\begin{aligned}[\![\Phi, \beta \vdash \alpha\colon \mathtt{Effect}]\!] \circ \omega &= [\![\Phi, \beta \vdash \alpha\colon \mathtt{Effect}]\!] \circ \omega' \circ \pi_1 \\ &= \Phi'' \vdash \alpha\colon \mathtt{Effect} \circ \pi_1 \\ &= \Phi'', \gamma \vdash \alpha\colon \mathtt{Effect} \\ &= \Phi' \vdash \alpha\colon \mathtt{Effect}\end{aligned}$$

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

So

$$\begin{aligned}[\![\Phi, \beta \vdash \alpha\colon \mathtt{Effect}]\!] \circ \omega &= [\![\Phi \vdash \alpha\colon \mathtt{Effect}]\!] \circ \pi_1 \circ (\omega' \times \mathtt{Id}_U) \\ &= [\![\Phi \vdash \alpha\colon \mathtt{Effect}]\!] \circ \omega' \circ \pi_1 \\ &= [\![\Phi'' \vdash \alpha\colon \mathtt{Effect}]\!] \circ \pi_1 \\ &= [\![\Phi' \vdash \alpha\colon \mathtt{Effect}]\!]\end{aligned}$$

□

A.2 Effect Weakening on Types

This proof proceeds in a similar fashion the proof of effect substitution on types (Theorem 3.5.4). That is, by induction over the derivation of $[\![\Phi \vdash A\colon \mathtt{Type}]\!]$, and making use of the Beck-Chevalley condition and the S-preserving property of ω^*.

Case T-Quantification: This case uses the Beck-Chevalley condition and the fact that $[\![\omega\times\colon (\Phi', \alpha) \triangleright (\Phi, \alpha)]\!] = \omega \times \mathtt{Id}_U$. This property is used in conjunction with induction to change the environment from (Φ, α) to (Φ', α).

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

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

$$\begin{aligned}
\omega^*[\![\Phi \vdash A \to B : \mathtt{Type}]\!] &= \omega^*([\![\Phi \vdash B : \mathtt{Type}]\!]^{[\![\Phi \vdash A : \mathtt{Type}]\!]}) \\
&= \omega^*([\![\Phi \vdash B : \mathtt{Type}]\!])^{\omega^*([\![\Phi \vdash A : \mathtt{Type}]\!])} \\
&= [\![\Phi' \vdash B : \mathtt{Type}]\!]^{[\![\Phi' \vdash A : \mathtt{Type}]\!]} \\
&= [\![\Phi' \vdash A \to B : \mathtt{Type}]\!]
\end{aligned}$$

□

A.3 Effect Weakening on Terms

This theorem is proved in a similar fashion to the effect substitution theorem (Theorem 3.5.9) 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 $\Delta = [\![A \leq :_\Phi B]\!] \circ \Delta_1$, as shown in Equation A.3.

$$\Delta = (\text{Subtype}) \frac{\dfrac{\Delta_1}{\Phi \mid \Gamma \vdash v : A} \qquad A \leq :_\Phi B}{\Phi \mid \Gamma \vdash v : B} \tag{A.3}$$

By induction there exists Δ_1' deriving $\Phi' \mid \Gamma \vdash v : A$, so we can construct Δ' using the subtyping rule in Equation A.4. So, using the weakening of the subtyping morphism, we can derive the denotation of Δ'.

$$\Delta' = (\text{Subtype}) \frac{\dfrac{\Delta_1'}{\Phi' \mid \Gamma \vdash v : A} \qquad A \leq :_\Phi B}{\Phi' \mid \Gamma \vdash v : B} \tag{A.4}$$

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

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

If Δ derives $\Phi \mid \Gamma \vdash \lambda x : A.v : A \to B$ 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 A.5 and A.6

$$\Delta = (\text{Fn}) \frac{\dfrac{\Delta_1}{\Phi \mid \Gamma, x : A \vdash v : B}}{\Phi \mid \Gamma \vdash \lambda x : A.v : A \to B} \tag{A.5}$$

$$\Delta = \mathtt{cur}(\Delta_1) \tag{A.6}$$

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 A.7 and A.8.

$$\Delta' = (\text{Fn})\frac{\dfrac{\Delta_1'}{\Phi' \mid \Gamma, x{:}\, A \vdash v{:}\, B}}{\Phi' \mid \Gamma \vdash \lambda x{:}\, A.v : A \to B} \tag{A.7}$$

$$\Delta' = \texttt{cur}(\Delta_1') \tag{A.8}$$

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

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

□

A.4 Term Weakening Theorem

Proceeds by induction on the derivation of Δ. Making use of the weakening of effect-variable environments on term weakenings in the case of $(Effect - Gen)$.

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

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

$$\Delta = (\text{Effect-Gen})\frac{\dfrac{\Delta_1}{\Phi, \alpha \mid \Gamma \vdash v{:}\, A}}{\Phi \mid \Gamma \vdash \Lambda\alpha.v{:}\, \forall\epsilon.A} \tag{A.9}$$

By induction, we derive Δ_1' such that it completes the tree in Equation A.10. By induction, its denotation can be related to that of Δ_1, as in Equation A.11. Hence we can show that the denotation of Δ is preserved in Equation A.12.

$$\Delta' = (\text{Effect-Gen})\frac{\dfrac{\Delta_1'}{\Phi, \alpha \mid \Gamma' \vdash v{:}\, A}}{\Phi \mid \Gamma' \vdash (\Lambda\alpha.v){:}\, \forall\epsilon.A} \tag{A.10}$$

$$\begin{aligned}
\Delta_1' &= \Delta_1 \circ [\![\Phi, \alpha \vdash \omega{:}\, \Gamma' \triangleright \Gamma]\!] \\
&= \Delta_1 \circ [\![\iota\pi{:}\, \Phi, a \triangleright \Phi]\!]^*(\omega) \\
&= \Delta_1 \circ \pi_1^*(\omega)
\end{aligned} \tag{A.11}$$

$$\begin{aligned}\Delta \circ \omega &= \overline{\Delta_1} \circ \omega \\ &= \overline{\Delta_1 \circ \pi_1^*(\omega)} \\ &= \overline{\Delta_1'} \\ &= \Delta'\end{aligned} \tag{A.12}$$

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

$$\Delta = (\text{Bind})\frac{\dfrac{\Delta_1}{\Phi \mid \Gamma \vdash v_1 : \mathtt{M}_{\mathbb{E}_1} A} \qquad \dfrac{\Delta_2}{\Phi \mid \Gamma, x : A \vdash v_2 : \mathtt{M}_{\epsilon_2} B}}{\Phi \mid \Gamma \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B} \tag{A.13}$$

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:

$$\Delta' = (\text{Bind})\frac{\dfrac{\Delta_1'}{\Phi \mid \Gamma' \vdash v_1 : \mathtt{M}_{\epsilon_1} A} \qquad \dfrac{\Delta_2'}{\Phi \mid \Gamma', x : A \vdash v_2 : \mathtt{M}_{\epsilon_2} B}}{\Phi \mid \Gamma' \vdash \mathtt{do}\ x \leftarrow v_1\ \mathtt{in}\ v_2 : \mathtt{M}_{\epsilon_1 \cdot \epsilon_2} B} \tag{A.14}$$

This preserves denotations:

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

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

$$\Delta = (\text{Return})\frac{\dfrac{\Delta_1}{\Phi \mid \Gamma \vdash v : A}}{\Phi \mid \Gamma \vdash \mathtt{return}\ v : \mathtt{M}_{\mathbf{1}} A} \tag{A.15}$$

Hence, by induction, with $\Phi \vdash \omega : \Gamma' \triangleright \Gamma$, we find the derivation Δ_1' such that:

$$\Delta' = (\text{Return})\frac{\dfrac{\Delta_1'}{\Phi \mid \Gamma' \vdash v : A}}{\Phi \mid \Gamma' \vdash \mathtt{return}\ v : \mathtt{M}_{\mathbf{1}} A} \tag{A.16}$$

This preserves denotations:

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

□

Appendix B

S-Category Functors

This appendix gives a terse introduction to the properties of functors between S-categories. In particular, it covers re-indexing functors f^* and the quantification functor $\forall_I$.

B.1 Re-Indexing Functors

For each morphism $f : I' \to I$ in $\mathbb{C}$, there should be a co-variant, re-indexing functor $f^* : \mathbb{C}(I) \to \mathbb{C}(I')$, which is S-preserving. That is, it preserves the S-category structure of $\mathbb{C}(I)$. (See below). $(-)^*$ should be a contra-variant functor in its $\mathbb{C}$ argument and co-variant in its right argument.

- $(g \circ f)^*(a) = f^*(\gamma^*(a))$
- $\mathtt{Id}_I^*(a) = a$
- $f^*(\mathtt{Id}_A) = \mathtt{Id}_{f^*(A)}$
- $f^*(a \circ b) = f^*(a) \circ f^*(b)$

B.1.1 Preserves Ground Types

If $[\![\gamma]\!] \in \mathtt{obj}\ \mathbb{C}(I)$ then $f^*[\![\gamma]\!] = [\![\gamma]\!] \in \mathtt{obj}\ \mathbb{C}(I')$

B.1.2 f^* Preserves Products

If $\langle g, h\rangle : \mathbb{C}(I)(Z, X \times Y)$ Then

$$\begin{aligned} f^*(X \times Y) &= f^*(X) \times f^*(Y) \\ f^*(\langle g, h\rangle) &= \langle f^*(g), f^*h\rangle && : \mathbb{C}(I')(f^*Z, f^*(X) \times f^*(Y)) \\ f^*(\pi_1) &= \pi_1 && : \mathbb{C}(I')(f^*(X) \times f^*(Y), f^*(X)) \\ f^*(\pi_2) &= \pi_2 && : \mathbb{C}(I')(f^*(X) \times f^*(Y), f^*(Y)) \end{aligned}$$

B.1.3 f^* Preserves Terminal Object

If $\mathtt{Id}_A : \mathbb{C}(I)(A, \mathtt{1})$ Then

$$\begin{aligned} f^*(\mathtt{1}) &= \mathtt{1} \\ f^*(\langle\rangle_A) &= \langle\rangle_{f^*(A)} && : \mathbb{C}(I')(f^*A, \mathtt{1}) \end{aligned}$$

B.1.4 f^* Preserves Exponentials

$$\begin{aligned} f^*(Z^X) &= (f^*(Z))^{(f^*(X))} \\ f^*(\mathtt{app}) &= \mathtt{app} && : \mathbb{C}(I')(f^*(Z^X) \times f^*(X), f^*(Z)) \\ f^*(\mathtt{cur}(g)) &= \mathtt{cur}(f^*(g)) && : \mathbb{C}(I')(f^*(Y) \times f^*(X), f^*(Z)^{f^*(X)}) \end{aligned}$$

B.1.5 f^* Preserves Co-product on Terminal

$$\begin{aligned} f^*(\mathtt{1} + \mathtt{1}) &= \mathtt{1} + \mathtt{1} \\ f^*(\mathtt{inl}) &= \mathtt{inl} && : \mathbb{C}(I')(\mathtt{1}, \mathtt{1} + \mathtt{1}) \\ f^*(\mathtt{inr}) &= \mathtt{inr} && : \mathbb{C}(I')(\mathtt{1}, \mathtt{1} + \mathtt{1}) \\ f^*([g, h]) &= [f^*(g), f^*(h)] && : \mathbb{C}(I')(\mathtt{1} + \mathtt{1}, f^*(Z)) \end{aligned}$$

B.1.6 f^* Preserves Graded Monad

$$\begin{aligned} f^*(T_\epsilon A) &= T_{f^*(\epsilon)} f^*(A) && : \mathbb{C}(I') \\ f^*(\eta_A) &= \eta_{f^*(A)} && : \mathbb{C}(I')(f^*(A), f^*(T_{\mathtt{1}} A)) \\ f^*(\mu_{\epsilon_1, \epsilon_2, A}) &= \mu_{f^*(\epsilon_1), f^*(\epsilon_2), f^*(A)} && : \mathbb{C}(I')(f^*(T_{\epsilon_1} T_{\epsilon_2} A), f^*(T_{\epsilon_1 \cdot \epsilon_2} A)) \end{aligned}$$

B.1.7 f^* and Effects

$$\begin{aligned} f^*(\mathtt{1}) &= \mathtt{1} \quad \text{The unit effect} \\ f^*(\epsilon_1 \cdot \epsilon_2) &= f^*(\epsilon_1) \cdot f^*(\epsilon_2) \quad \text{Multiplication} \end{aligned}$$

This is done By

$$f^* [\![\Phi \vdash \epsilon : \mathtt{Effect}]\!] = [\![\Phi \vdash \epsilon : \mathtt{Effect}]\!] \circ f$$

B.1.8 f^* Preserves Tensor Strength

$$f^*(\mathtt{t}_{\epsilon, A, B}) = \mathtt{t}_{f^*(\epsilon), f^*(A), f^*(B)} \qquad : \mathbb{C}(I')(f^*(A \times T_\epsilon B), f^*(T_\epsilon(A \times B)))$$

B.1.9 f^* Preserves Ground Constants

For each ground constant $[\![\mathtt{k}^A]\!]$ in $\mathbb{C}(I)$,

$$f^*([\![\mathtt{k}^A]\!]) = \mathtt{k}^{f^*(A)} : \mathbb{C}(I')(\mathtt{1}, f^*(A))$$

B.1.10 f^* Preserves Ground Subeffecting

For ground effects e_1, e_2 such that $e_1 \leq e_2$

$$\begin{aligned} f^*(e) &= e : \mathbb{C}(I') \\ f^*[\![\epsilon_1 \leq e_2]\!]_A &= [\![e_1 \leq e_2]\!]_{f^*(A)} : \mathbb{C}(I') f^*(T_{e_1} A), f^*(T_{e_2} A) \end{aligned}$$

B.1.11 f^* Preserves Ground Subtyping

For ground types γ_1, γ_2 such that $\gamma_1 \leq:_\gamma \gamma_2$

$$\begin{aligned} f^*\gamma &= \gamma : \mathbb{C}(I')(\mathtt{1}, \gamma) \\ f^*([\![\gamma_1 \leq:_\gamma \gamma_2]\!]) &= [\![\gamma_1 \leq:_\gamma \gamma_2]\!] && : \mathbb{C}(I')(\gamma_1, \gamma_2) \end{aligned}$$

B.2 The $\forall_I$ functor

We expand $\forall_I : \mathbb{C}(I \times U) \to \mathbb{C}(I)$ to be a functor which is right adjoint to the re-indexing functor π_1^*.

$$\overline{(\_)} : \mathbb{C}(I \times U)(\pi_1^* A, B) \leftrightarrow \mathbb{C}(I)(A, \forall_I B) : \widehat{(\_)} \tag{B.1}$$

For $A \in \mathtt{obj}\ \mathbb{C}(I)$, $B \in \mathtt{obj}\ \mathbb{C}(I \times U)$.

Hence the action of $\forall_I$ on a morphism $l : A \to A'$ is as follows:

$$\forall_I(l) = \overline{l \circ \boldsymbol{\epsilon}_A} \tag{B.2}$$

Where $\epsilon_A : \mathbb{C}(I \times U)(\pi_1^* \forall_I A \to A)$ is the co-unit of the adjunction.

B.2.1 Beck Chevalley Condition

We need to be able to commute the $\forall_I$ functor with re-indexing functors. A natural way to do this is:

$$\theta^* \circ \forall_I = \forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*$$

We shall also require that the canonical natural-transformation between these functors is the identity.

That is, $\overline{(\theta \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon})} = \mathtt{Id} : \theta^* \circ \forall_I \to \forall_{I'} \circ (\theta \times \mathtt{Id}_U)^* \in \mathbb{C}(I')$

This shall be called the Beck-Chevalley condition.

B.3 Naturality Corollaries

Here are some simple corollaries of the adjunction between π_1^* and $\forall_I$.

B.3.1 Naturality

By the definition of an adjunction:

$$\overline{f \circ \pi_1^*(n)} = \overline{f} \circ n \tag{B.3}$$

B.3.2 $\overline{(-)}$ and Re-indexing Functors

By assuming the Beck-Chevalley condition that:

$$\overline{(\theta \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon})} = \mathtt{Id} : \theta^* \circ \forall_I \to \forall_{I'} \circ (\theta \times \mathtt{Id}_U)^* \tag{B.4}$$

We then have:

$$\begin{aligned}
\theta^*\boldsymbol{\eta}_A : \quad & \theta^* A \to \theta^* \circ \forall_I \circ \pi_1^* A \\
\theta^*\boldsymbol{\eta} = & \overline{(\theta \times \mathtt{Id}_U)^*(\boldsymbol{\epsilon}_{\pi_1^*})} \circ \theta^*\boldsymbol{\eta} \\
= & (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)(\boldsymbol{\epsilon}_{\pi_1^*}) \circ \boldsymbol{\eta}_{(\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*) \circ \pi_1^*} \circ \theta^*\boldsymbol{\eta} \\
= & (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)(\boldsymbol{\epsilon}_{\pi_1^*}) \circ \boldsymbol{\eta}_{\theta^* \circ \forall_I \circ \pi_1^*} \circ \theta^*\boldsymbol{\eta} \\
= & (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*)(\boldsymbol{\epsilon}_{\pi_1^*}) \circ (\theta^* \circ \forall_I \circ \pi_1^*)\boldsymbol{\eta} \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*} \\
= & (\theta^* \circ \forall_I)(\boldsymbol{\epsilon}_{\pi_1^*} \circ \pi_1^*\boldsymbol{\eta}) \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*} \\
= & (\theta^* \circ \forall_I)(\mathtt{Id}) \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*} \\
= & \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^*}
\end{aligned}$$

$$\begin{aligned}
\theta^*(\overline{f}) &= \theta^*(\forall_I(f) \circ \eta_A) \\
&= \theta^*(\forall_I(f)) \circ \theta^*(\eta_A) \\
&= (\forall_{I'} \circ (\theta \times \mathtt{Id}_U)^*) f \circ \boldsymbol{\eta}_{(\theta \times \mathtt{Id}_U)^* A} \\
&= \overline{(\theta \times \mathtt{Id}_U)^* f}
\end{aligned}$$

B.3.3 $\hat{(-)}$ and Re-Indexing Functors

$$\begin{aligned}
\theta^*(\langle \mathtt{Id}_I, \rho \rangle^*(\widehat{m})) &= (\langle \mathtt{Id}_I, \rho \rangle \circ \theta)^*(\widehat{m}) \\
&= ((\theta \times \mathtt{Id}_U) \circ \langle \mathtt{Id}_I, \rho \rangle)^*(\widehat{m}) \\
&= \langle \mathtt{Id}_I, \rho \circ \theta \rangle^*(\theta \times \mathtt{Id}_U)^*(\widehat{m}) \\
&= \langle \mathtt{Id}_I, \theta^*\rho \rangle^*(\theta^*(\widehat{m}))
\end{aligned}$$

B.3.4 Pushing Morphisms into f^*

$$\begin{aligned}\langle \mathtt{Id}_I, \rho\rangle^*(\widehat{m}) \circ n &= \langle \mathtt{Id}_I, \rho\rangle^*(\widehat{m}) \circ \langle \mathtt{Id}_I, \rho\rangle^* \pi_1^*(n) \\ &= \langle \mathtt{Id}_I, \rho\rangle^*(\widehat{m} \circ \pi_1^*(n)) \\ &= \langle \mathtt{Id}_I, \rho\rangle^*(\widehat{m \circ n})\end{aligned}$$

Appendix C

Functorial S-Categories

This appendix provides a full, although terse, exposition of S-category structure of functor categories $[E^n, \mathtt{Set}]$ and the S-preserving nature of the re-indexing functors between them.

C.1 Base Category is Monoidal

We construct the base category, `Eff`, as follows:

- $U = E$, the set of ground effects in the non-polymorphic language.
- `1` is a singleton set.
- $U^n = E^n$, set of n-wide tuples of effects, $\vec{\epsilon}$

Hence when we treat effects that are well formed in Φ as morphisms, $E^n \to E$ in `Eff`, we should treat them as functions $f : E^n \to E$. Ground effects become point functions: $e : \mathtt{1} \to E$, so the denotation of a ground effect is the constant value function: $[\![\Phi \vdash e{:}\mathtt{Effect}]\!] = \vec{\epsilon} \mapsto e$. We extend the multiplication of ground effects to multiplication on effect functions, giving us our $\mathtt{Mul}_n$ operation $\mathtt{Mul}_n(f,g)(\vec{\epsilon}) = (f\vec{\epsilon})\cdot(g\vec{\epsilon})$. This $\mathtt{Mul}_n$ satisfies the naturality and monoidal requirements, as seen in figure C.1. It is also trivially the case that $\mathtt{Mul}_n([\![\Phi \vdash e_1{:}\mathtt{Effect}]\!], [\![\Phi \vdash e_2{:}\mathtt{Effect}]\!]) = [\![\Phi \vdash e_1 \cdot e_2{:}\mathtt{Effect}]\!]$.

C.2 S-Categories

The semantic category, $[E^0, \mathtt{Set}]$ of the effect-environment $\diamond$ is isomorphic to `Set`. Since each effect-environment is alpha equivalent to a natural number, the semantic category for Φ shall be represented as $\mathbb{C}(\Phi) = \mathbb{C}(n) = [E^n, \mathtt{Set}]$, the category of functions $E^n \to \mathtt{Set}$. Objects in $[E^n, \mathtt{Set}]$ are functions and we describe them by their actions on a generic vector of ground effects, $\vec{\epsilon}$. Morphisms in $[E^n, \mathtt{Set}]$ are natural transformations between the functions. So:

$$\begin{aligned} m : &\quad A \to B \quad \text{In } [E^n, \mathtt{Set}] \\ m\vec{\epsilon} : &\quad A\vec{\epsilon} \to B\vec{\epsilon} \quad \text{In } \mathtt{Set} \\ (f \circ g)\vec{\epsilon} = &\quad (f\vec{\epsilon}) \circ (g\vec{\epsilon}) \\ \mathtt{Id}_A(\vec{\epsilon}) = &\quad \mathtt{Id}_{A\vec{\epsilon}} \end{aligned}$$

So morphisms are dependently typed functions from a vector of ground effects to morphisms in `Set`.

Naturality

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

Left and Right Identity

$$\begin{aligned}\texttt{Mul}_n([\![\Phi\vdash \texttt{1}:\texttt{Effect}]\!],f)(\vec{\epsilon}) &= (\texttt{1}\vec{\epsilon})\cdot(f\vec{\epsilon})\\ &= (\texttt{1})\cdot(f\vec{\epsilon})\\ &= f\vec{\epsilon}\end{aligned}\qquad\begin{aligned}\texttt{Mul}_n(f,[\![\Phi\vdash \texttt{1}:\texttt{Effect}]\!])(\vec{\epsilon}) &= (f\vec{\epsilon})\cdot(\texttt{1}\vec{\epsilon})\\ &= (f\vec{\epsilon})\cdot(\texttt{1})\\ &= f\vec{\epsilon}\end{aligned}$$

Monoid Associativity

$$\begin{aligned}\texttt{Mul}_n(f,\texttt{Mul}_n(g,h))(\vec{\epsilon}) &= (f\vec{\epsilon})\cdot((g\vec{\epsilon})\cdot(h\vec{\epsilon}))\\ &= ((f\vec{\epsilon})\cdot(g\vec{\epsilon}))\cdot(h\vec{\epsilon})\\ &= \texttt{Mul}_n(\texttt{Mul}_n(f,g),h)(\vec{\epsilon})\end{aligned}$$

Figure C.1: Our pick for `Mul` is natural and is a monoid at each n.

C.2.1 Each S-Category is a CCC

Since $\mathtt{Set}$ is complete and a CCC, and E^n is small, since E is small, $[E^n, \mathtt{Set}]$ is a CCC.

$$\begin{aligned}
(A \times B)\vec{\epsilon} &= (A\vec{\epsilon}) \times (B\vec{\epsilon}) \\
\mathtt{1}\vec{\epsilon} &= \mathtt{1} \\
(B^A)\vec{\epsilon} &= (B\vec{\epsilon})^{(A\vec{\epsilon})} \\
\pi_1\vec{\epsilon} &= \pi_1 \\
\pi_2\vec{\epsilon} &= \pi_2 \\
\mathtt{app}\vec{\epsilon} &= \mathtt{app} \\
\mathtt{cur}(f)\vec{\epsilon} &= \mathtt{cur}(f\vec{\epsilon}) \\
\langle f, g\rangle\vec{\epsilon} &= \langle f\vec{\epsilon}, g\vec{\epsilon}\rangle
\end{aligned}$$

C.2.2 The Terminal Co-Product

We can define the co-product point-wise.

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

This preserves the co-product diagram.

$$\begin{aligned}
([f, g] \circ \mathtt{inl})\vec{\epsilon} &= [f\vec{\epsilon}, g\vec{\epsilon}] \circ \mathtt{inl} \\
&= f\vec{\epsilon} \\
&\square \\
([f, g] \circ \mathtt{inr})\vec{\epsilon} &= [f\vec{\epsilon}, g\vec{\epsilon}] \circ \mathtt{inr} \\
&= f\vec{\epsilon} \\
&\square
\end{aligned}$$

$[f, g]$ is also unique in $[E^n, \mathtt{Set}]$. Suppose $l \circ \mathtt{inl} = f$ and $l \circ \mathtt{inr} = g$ in $[E^n, \mathtt{Set}]$. Then $l\vec{\epsilon} \circ \mathtt{inl} = f\vec{\epsilon}$ and $l\vec{\epsilon} \circ \mathtt{inr} = g\vec{\epsilon}$. Hence by the co-product in $\mathtt{Set}$, $l = [f\vec{\epsilon}, g\vec{\epsilon}]$ so $l = [f, g]$.

C.2.3 Ground Types and Terms

Each ground type in the non-polymorphic calculus has a fixed denotation $[\![\gamma]\!] \in \mathtt{obj\ Set}$. The ground type in the polymorphic calculus hence has a denotation represented by the constant function.

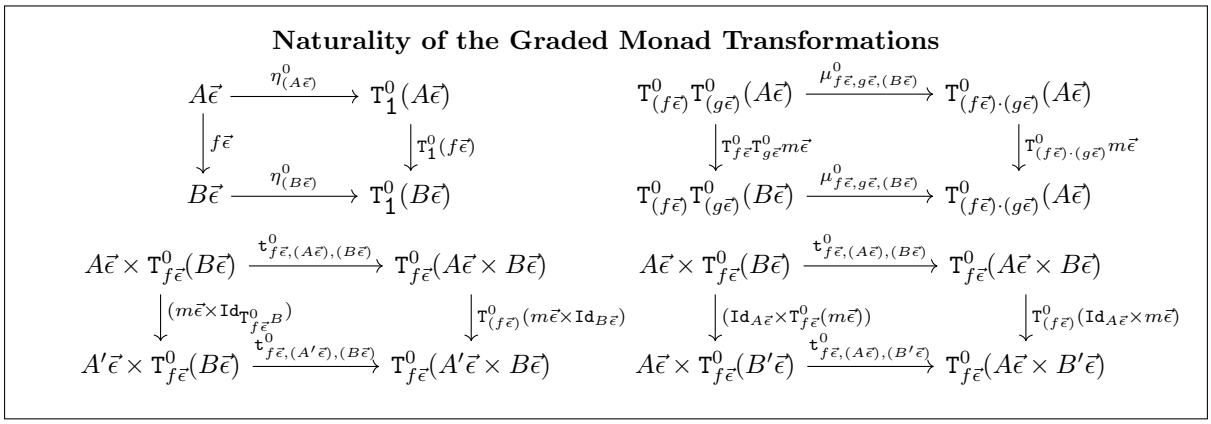


Figure C.2: Naturality Squares for the graded monad natural transformations. The naturality of each transformation α in $[E^n, \mathtt{Set}]$ derives from the naturality of each of its component transformations $\alpha\vec{\epsilon}$ in $\mathtt{Set}$

$$\begin{aligned} [\![\gamma]\!] : &\quad E^n \to \mathtt{obj\ Set} \\ \vec{\epsilon} \mapsto &\quad [\![\gamma]\!] \end{aligned}$$

Each constant term $\mathtt{k}^A$ in the non-polymorphic calculus has a fixed denotation $[\![\mathtt{k}^A]\!] \in \mathtt{Set}(\mathtt{1}, A)$. So the morphism $[\![\mathtt{k}^A]\!]$ in $[E^n, \mathtt{Set}]$ is the corresponding constant dependently typed morphism returning the $[\![\mathtt{k}^A]\!]$ function in $\mathtt{Set}$.

$$\begin{aligned} [\![\mathtt{k}^A]\!] : &\quad [E^n, \mathtt{Set}](\mathtt{1}, A) \\ \vec{\epsilon} \mapsto &\quad [\![\mathtt{k}^A]\!] \end{aligned}$$

C.2.4 Graded Monad

Given the strong graded monad $(\mathtt{T}^0, \eta^0, \mu^0, \mathtt{t}^0)$ on $\mathtt{Set}$, we can construct an appropriate graded monad $(\mathtt{T}^n, \eta^n, \mu^n, \mathtt{t}^n)$ on $[E^n, \mathtt{Set}]$. Through some mechanical proof and the naturality of the $\mathtt{Set}$ strong graded monad, these morphisms are natural in their type parameters and form a strong graded monad in $[E^n, \mathtt{Set}]$.

$$\begin{aligned} \mathtt{T}^n : &\quad (E^n, \cdot, \leq_n, \mathtt{1}_n) \to [[E^n, \mathtt{Set}], [E^n, \mathtt{Set}]] \\ (\mathtt{T}^n_f A)\vec{\epsilon} = &\quad \mathtt{T}^0_{(f\vec{\epsilon})} A\vec{\epsilon} \\ (\eta^n_A)\vec{\epsilon} = &\quad \eta^0_{A\vec{\epsilon}} \\ (\mu^n_{f,g,A})\vec{\epsilon} = &\quad \mu^0_{(f\vec{\epsilon}),(g\vec{\epsilon}),(A\vec{\epsilon})} \\ (\mathtt{t}^n_{f,A,B})\vec{\epsilon} = &\quad \mathtt{t}^0_{(f\vec{\epsilon}),(A\vec{\epsilon}),(B\vec{\epsilon})} \end{aligned}$$

The required naturality, monad, and tensor-strength laws hold, as proved in in figures C.2-C.4

Monad Laws

Left Unit

$$
\begin{aligned}
(\mu^n_{\mathtt{1},g,A} \circ \eta^n_{\mathtt{T}^n_f A})\vec{\epsilon} &= \mu^0_{\mathtt{1},(f\vec{\epsilon}),(A\vec{\epsilon})} \circ (\eta^0_{\mathtt{T}^0_{f\vec{\epsilon}} A\vec{\epsilon}}) \\
&= \mathtt{Id}_{\mathtt{T}^0_{f\vec{\epsilon}} A\vec{\epsilon}} \\
&= (\mathtt{Id}_{\mathtt{T}^n_f A})\vec{\epsilon}
\end{aligned}
$$

Right Unit

$$
\begin{aligned}
(\mu^n_{f,\mathtt{1},A} \circ \mathtt{T}^n_f \eta^n_A)\vec{\epsilon} &= \mu^0_{(f\vec{\epsilon}),\mathtt{1},(A\vec{\epsilon})} \circ \mathtt{T}^0_{f\vec{\epsilon}}(\eta^0_{A\vec{\epsilon}}) \\
&= \mathtt{Id}_{\mathtt{T}^0_{f\vec{\epsilon}} A\vec{\epsilon}} \\
&= (\mathtt{Id}_{\mathtt{T}^n_f A})\vec{\epsilon}
\end{aligned}
$$

Monad Associativity

$$
\begin{aligned}
((\mu^n_{f,(g\cdot h),A}) \circ \mathtt{T}^n_f(\mu^n_{g,h,A}))\vec{\epsilon} &= \mu^0_{(f\vec{\epsilon}),((g\vec{\epsilon})\cdot(h\vec{\epsilon})),(A\vec{\epsilon})} \circ \mathtt{T}^0_{f\vec{\epsilon}} \mu^0_{(h\vec{\epsilon}),(g\vec{\epsilon}),A\vec{\epsilon}} \\
&= \mu^0_{((f\vec{\epsilon})\cdot(g\vec{\epsilon})),(h\vec{\epsilon}),(A\vec{\epsilon})} \circ \mu^0_{(f\vec{\epsilon}),(g\vec{\epsilon}),(\mathtt{T}^0_{h\vec{\epsilon}}(A\vec{\epsilon}))} \\
&= (\mu^n_{f\cdot g,h,A} \circ \mu^n_{f,g,\mathtt{T}^0_h A})\vec{\epsilon}
\end{aligned}
$$

Figure C.3: The monad laws for the graded monad $(\mathtt{T}^n, \eta^n, \mu^n, \mathtt{t}^n)$ can be proved component-wise from the graded monad $(\mathtt{T}^0)$ on $\mathtt{Set}$.

C.2.5 Subeffecting

Given a collection of subeffecting natural transformation in $\mathtt{Set}$, $[\![\epsilon_1 \leq_0 \epsilon_2]\!] : \mathtt{T}^0_{\epsilon_1} \to \mathtt{T}^0_{\epsilon_2}$ we can form subeffect natural transformations in $[E^n, \mathtt{Set}]$ as seen in figure C.5. This natural transformation has all the required properties. These are proved in figure C.6.

C.3 Re-Indexing Functors

For a function $\theta : E^m \to E^n$, the re-indexing functor θ^* is defined as follows:

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

This functor preserves all the S-category properties. These can be seen in figures C.7-C.10

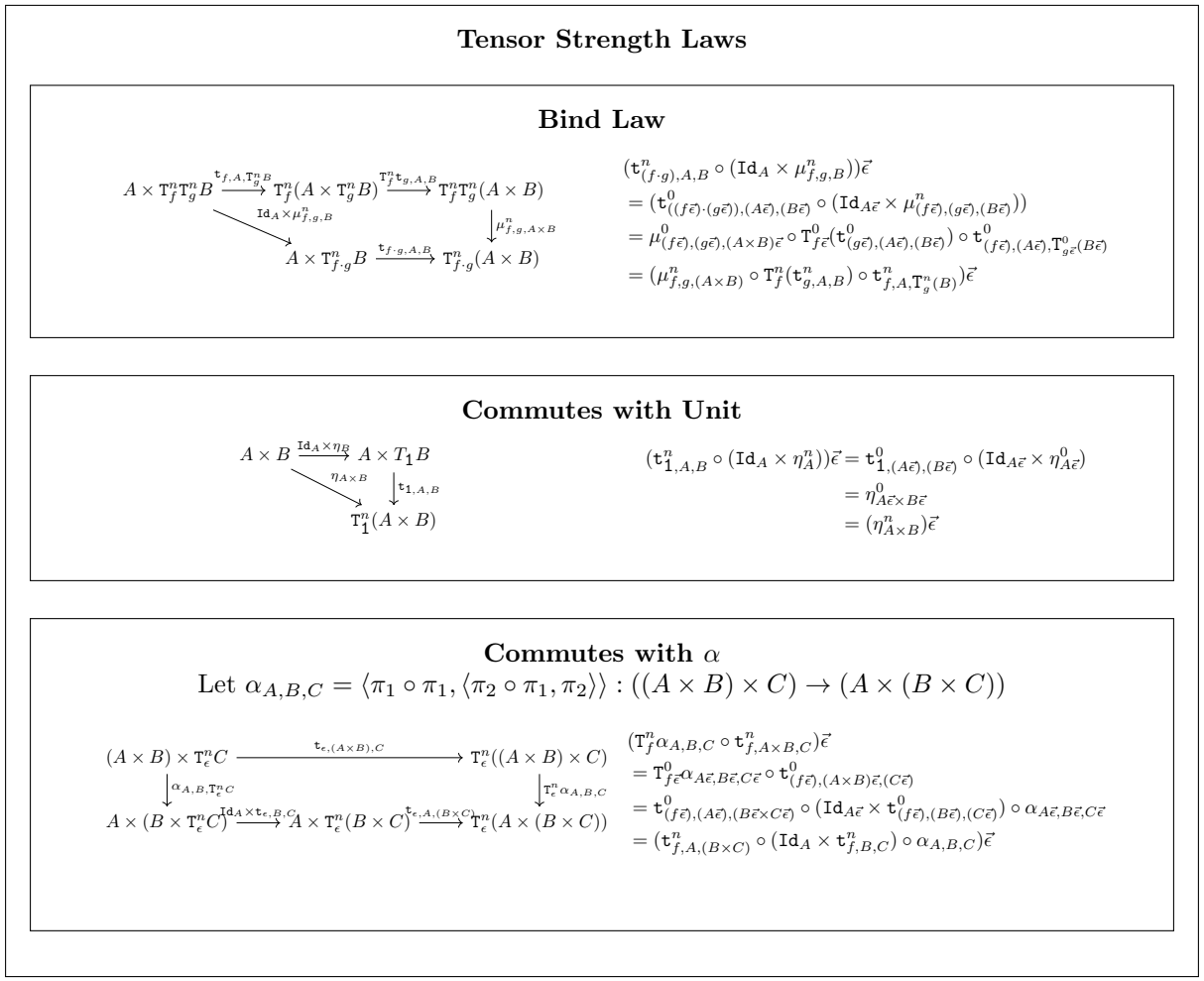


Figure C.4: The tensor strength laws can be proved component wise from the strength of the monad $\mathtt{T}^0$

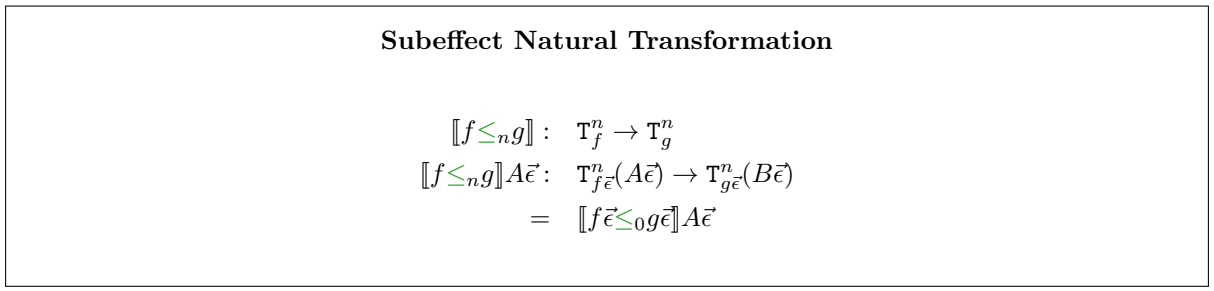


Figure C.5: Definition of subeffecting natural transformations

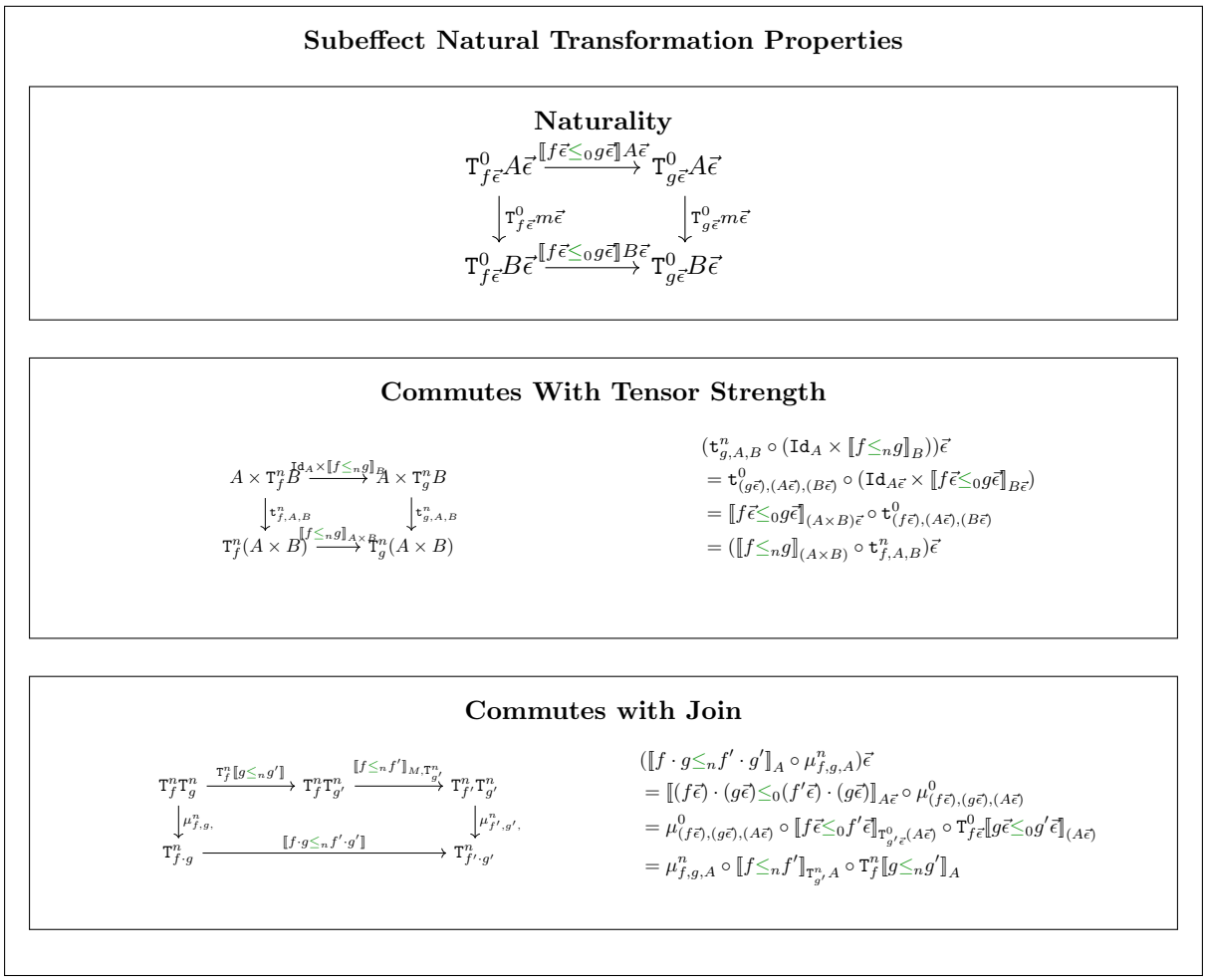


Figure C.6: The required properties of the subeffect natural transformations can be proved componentwise from the appropriate of the subeffect natural transformation on `Set`.

θ^* is Cartesian Closed

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

$$\begin{aligned}(\theta^*\pi_1)\vec{\epsilon} &= \pi_1(\theta\vec{\epsilon}) \\ &= \pi_1 \quad \text{Constant function} \\ &= \pi_1\vec{\epsilon}\end{aligned}$$

$$\begin{aligned}(\theta^*\pi_2)\vec{\epsilon} &= \pi_2(\theta\vec{\epsilon}) \\ &= \pi_2 \quad \text{Constant function} \\ &= \pi_2\vec{\epsilon}\end{aligned}$$

$$\begin{aligned}(\theta^*\langle f, g\rangle)\vec{\epsilon} &= (\langle f, g\rangle)(\theta\vec{\epsilon}) \\ &= \langle f(\theta\vec{\epsilon}), g(\theta\vec{\epsilon})\rangle \\ &= \langle\theta^*f, \theta^*g\rangle\vec{\epsilon}\end{aligned}$$

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

$$\begin{aligned}(\theta^*\mathtt{app})\vec{\epsilon} &= \mathtt{app}(\theta\vec{\epsilon}) \\ &= \mathtt{app} \quad \text{Constant fn} \\ &= \mathtt{app}\vec{\epsilon}\end{aligned}$$

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

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

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

Figure C.7: Proof of the CCC-preserving property of re-indexing functors.

θ^* Preserves Co-products

$$\begin{aligned}(\theta^*(\mathtt{1}+\mathtt{1}))\vec{\epsilon} &= (\mathtt{1}+\mathtt{1})(\theta\vec{\epsilon}) \\ &= (\mathtt{1}+\mathtt{1}) \quad \text{Constant function} \\ &= (\mathtt{1}+\mathtt{1})\vec{\epsilon}\end{aligned}$$

$$\begin{aligned}(\theta^*\mathtt{inl})\vec{\epsilon} &= \mathtt{inl}(\theta\vec{\epsilon}) \\ &= \mathtt{inl} \quad \text{Constant Fn} \\ &= \mathtt{inl}\vec{\epsilon}\end{aligned}$$

$$\begin{aligned}(\theta^*\mathtt{inr})\vec{\epsilon} &= \mathtt{inr}(\theta\vec{\epsilon}) \\ &= \mathtt{inr} \quad \text{Constant Fn} \\ &= \mathtt{inr}\vec{\epsilon}\end{aligned}$$

$$\begin{aligned}(\theta^*[f, g])\vec{\epsilon} &= [f, g](\theta\vec{\epsilon}) \\ &= [f(\theta\vec{\epsilon}), g(\theta\vec{\epsilon})] \\ &= [\theta^*f, \theta^*g]\vec{\epsilon}\end{aligned}$$

Figure C.8: Proof that re-indexing functors preserve the $\mathtt{1}+\mathtt{1}$ co-product.

θ^* Preserves the Graded Monad

$$\begin{aligned}(\theta^*\mathtt{T}^n_f A)\vec{\epsilon} &= \mathtt{T}^n_f A(\theta\vec{\epsilon}) \\ &= \mathtt{T}^0_{(f(\theta\vec{\epsilon}))}(A(\theta\vec{\epsilon})) \\ &= (\mathtt{T}^m_{(f\circ\theta)}\theta^* A)\vec{\epsilon}\end{aligned} \qquad \begin{aligned}(\theta^*\eta^n_A)\vec{\epsilon} &= \eta^n_A(\theta\vec{\epsilon}) \\ &= \eta^0_{A(\theta\vec{\epsilon})} \\ &= \eta^m_{\theta^* A}\vec{\epsilon}\end{aligned}$$

$$\begin{aligned}(\theta^*\mu^n_{f,g,A})\vec{\epsilon} &= \mu^n_{f,g,A}(\theta\vec{\epsilon}) \\ &= \mu^0_{f(\theta\vec{\epsilon}),g(\theta\vec{\epsilon}),A(\theta\vec{\epsilon})} \\ &= \mu^m_{f\circ\theta,g\circ\theta,\theta^*(A)}(\vec{\epsilon})\end{aligned} \qquad \begin{aligned}(\theta^*\mathtt{t}^n_{f,A,B})\vec{\epsilon} &= \mathtt{t}^n_{f,A,B}(\theta\vec{\epsilon}) \\ &= \mathtt{t}^0_{(f(\theta\vec{\epsilon})),(A(\theta\vec{\epsilon})),(B(\theta\vec{\epsilon}))} \\ &= \mathtt{t}^m_{f\circ\theta,\theta^* A,\theta^* B}\vec{\epsilon}\end{aligned}$$

Figure C.9: Re-indexing functors preserve the graded monad structure

θ^* preserves Ground Subtyping and Subeffecting

$$\begin{aligned}\theta^*([\![A{\leq:}_\gamma B]\!])\vec{\epsilon} &= [\![A{\leq:}_\gamma B]\!](\theta\vec{\epsilon}) \\ &= [\![A{\leq:}B]\!] \quad \text{Constant Function} \\ &= [\![A{\leq:}B]\!]\vec{\epsilon}\end{aligned} \qquad \begin{aligned}(\theta^*([\![f{\leq_n}g]\!]A))\vec{\epsilon} &= ([\![f{\leq_n}g]\!]A)(\theta\vec{\epsilon}) \\ &= ([\![f(\theta\vec{\epsilon}){\leq_n}g(\theta\vec{\epsilon})]\!](A(\theta\vec{\epsilon}))) \\ &= ([\![\theta^* f{\leq_m}\theta^* g]\!](\theta^* A))\vec{\epsilon}\end{aligned}$$

Figure C.10: Re-indexing functors preserve the ground subtyping and subeffecting morphisms.