Foundations of Type

Programming in Mathematics

Edward O'Callaghan

"Foundations of Type"

Book version: `0.1`

ISBN: `???-???-???.`

This book is freely available at http://victoredwardocallaghan.github.io/book/

Acknowledgment

Thanks to github...

Preface

Book development

The following people participated in the development of this text.

Edward O'Callaghan

About this book

This book had its genesis from lecture notes I compiled myself while at university. I wrote many notes as I felt things were ether too fragmented, disordered or just poorly explained. With the modern advances in mathematics, type theory in particular, we are on the precious of a new era in computer science, where computer science will be given back to mathematicians. A merger of disciplines on seemly two different trajectories into a more unified discipline of rigours computational linguistics.

However, computer science and mathematics are separately taught fields and so, as a result, has lead to this artificial barrier.This systemic problem has been further aggravated by computer science actively avoiding the underlaying mathematics and mathematics paying little to no interest in undergraduate computational theory. A number of topics that have a deep unexpected connection that are prevalent in both disciplines are covered. In particular, *dependent types*, *category theory* and *homotopy type theory* are some of the quint essential topics of interest. This text serves to introduce the reader to programming and the foundations of mathematics from scratch by way of *homotopy type theory* and *category theory* and explores aspects of these topics to lead to a new era in high assurance software design and implementations by programs as proofs. We complement this with the concept of propositions as types as finally consider the connection between *category theory* and *homotopy type theory*.

This text serves my interest to once more marry these fields and provide a standard introductory text for undergraduates. My hope is to make this text accessible as to serve to support development of a modern rigours treatment from first to third year undergraduate.

Foundations of Type
Edward O'Callaghan
Australia, Jun 2013

Contents

Introduction

In this text we explore the essential material required in the pursuit of praxis in the concept ideas of *computational trinitarianism*.

We consider notions such as;

- propositions as types,
- programs as proofs,
- and relations between type and category theory.

This deep unexpected connection is referred to as computational trinitarianism

> The central dogma of computational trinitarianism holds that Logic, Languages, and Categories are but three manifestations of one divine notion of computation. There is no preferred route to enlightenment: each aspect provides insights that comprise the experience of computation in our lives.
>
> Computational trinitarianism entails that any concept arising in one aspect should have meaning from the perspective of the other two. If you arrive at an insight that has importance for logic, languages, and categories, then you may feel sure that you have elucidated an essential concept of computation you have made an enduring scientific discovery.

.. foo bar

Part I

Foundations in Mathematics and Computation

Chapter 1

Prelude

1.1 Predicate logic

..

1.2 Modal logic

..

1.3 Set theory

..

1.4 Equivalences

..

1.5 Morphisms

A morphism is a map between two objects in an abstract category from *dom*ain to *codom*ain, more on this later. A general morphism is called a homomorphism. A homomorphism is a term typically used in category theory. The term itself derives from the Greek $o\mu o$ (omo), "alike" and $\mu o\rho\varphi\omega\sigma\iota\varsigma$ (morphosis), "to form" or "to shape".

Remark. The similarity in meaning and form of the words "homomorphism" and "homeomorphism" is unfortunate and typically a source of common confusion.

Morphisms can have any of the following properties.

1.5.1 Monomorphism

Definition 1.5.1 (Monomorphism)**.** A morphism $\varphi : X \to Y$ is a *monomorphism* if $\varphi \circ \psi = \varphi \circ \phi \implies \psi = \phi$ for all morphisms $\psi, \phi : Z \to X$.

$$Z \underset{\phi}{\overset{\psi}{\rightrightarrows}} X \xrightarrow{\varphi} Y$$

1.5.2 Epimorphism

Definition 1.5.2 (Epimorphism)**.** A morphism $\varphi : X \to Y$ is a *epimorphism* if $\psi \circ \varphi = \phi \circ \varphi \implies \psi = \phi$ for all morphisms $\psi, \phi : Y \to Z$.

$$X \xrightarrow{\varphi} Y \underset{\phi}{\overset{\psi}{\rightrightarrows}} Z$$

1.5.3 Bimorphism

Definition 1.5.3 (Bimorphism)**.** A morphism $\varphi : X \to Y$ is a *bimorphism* if φ is both a monomorphism and epimorphism.

1.5.4 Isomorphism

Definition 1.5.4 (Isomorphism)**.** A morphism $\varphi : X \to Y$ is a *isomorphism* if there exists a morphism $\psi : Y \to X : \varphi \circ \psi = id_Y$ and $\psi \circ \varphi = id_X$.

1.5.5 Endomorphism

Definition 1.5.5 (Endomorphism)**.** A morphism $\varphi : X \to X$ is a *endomorphism*. The class of endomorphisms of φ is denoted by $end(\varphi)$.

1.5.6 Automorphism

Definition 1.5.6 (Automorphism)**.** A morphism $\varphi : X \to Y$ is a *automorphism* if φ is both an endomorphism and an isomorphism. The class of automorphisms of φ is denoted by $aut(\varphi)$.

1.5.7 Retractions and Sections

Definition 1.5.7 (Retraction)**.** A morphism $\varphi : X \to Y$ has a *retraction* if there exists a morphism $\psi : Y \to X$ with $\varphi \circ \psi = id_Y$ - "if a right inverse of φ exists".

Definition 1.5.8 (Section)**.** A morphism $\varphi : X \to Y$ has a *section* if there exists a morphism $\psi : Y \to X$ with $\psi \circ \varphi = id_X$ - "if a left inverse of φ exists".

Lemma 1.5.9 (Retractions and Sections)**.** *Every retraction is an epimorphism and every section is a monomorphism. We have the following three equivalent statements:*

- *φ is a monomorphism and a retraction;*
- *φ is an epimorphism and a section;*
- *φ is an isomorphism.*

Chapter 2

Algorithms I

Chapter 3

Type Theory I

Chapter 4

Algebra

Chapter 5

Data Structures

Chapter 6

Algorithms I

Part II

Higher Mathematics

Chapter 7

Toplogy

Chapter 8

Category theory

8.1 Prelude

First a word on how to study category theory. Category theory is extremely heavy on new terminology at first sight. It is typically misconceived as abstractly and arbitrarily renaming things to no real gain and a generally poorly understood topic in undergraduate mathematics. Consider taking the time to highlight these notes in assortment of coloured pens, with each new term given a corresponding colour. To get over the initial shock of, perhaps, new terms and to consolidate concepts. Also note that everything may be represented as a diagram and that we make the notion of a diagram rigous later. However, a categorial diagram is simply a directed graph, where the nodes are objects from a category while morphism are the edges. I highly recommend you copy each new term with a corresponding diagram to anecdote the idea at hand. If all else fails, be sure to remember to break the term down into prefix and suffix and reason it. Typically terms are fairly simple to reason out (e.g., mono = "one", auto = "self", morphism = "to shape"). Category theory is actually a generalisation of set theory and so, we 'stress' that a class of objects is not nessarily a set and is considered more abstract. Keep this in mind when we look at 'small' and 'large' categories we shall see later.

Note (Class). A *class* is a collection of mathematical objects (e.g., sets) which can be unambiguously defined by a property that all its members share.

Note (Morphism). A *morphism*, from Greek, is an abstraction derived from structure-preserving mappings between two mathematical structures. The notion of morphism recurs in much of contemporary mathematics.

8.2 Introduction

In each area of mathematics (e.g., sets, groups, topological spaces) there are available many definitions and constructions. It turns out, however, that there are a number of notions (e.g. that of a product) that occur naturally in various areas of mathematics, with only slight changes from one area to another. It is convenient to take advantage of this observation. Category theory can be described as that branch of mathematics in which one studies certain definitions in a broader context - without reference to the particular area to which the definition might be applied.

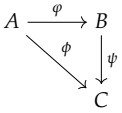
8.3 Categories

A category consists of three things: a collection of objects, for each pair of objects a collection of morphisms from one to another, and a binary operation defined on compatible pairs of morphisms called composition denoted by $\circ$. The category must satisfy an identity axiom and an associative axiom which is analogous to the monoid axioms, seen later. We require that morphisms preserve the mathematical structure of the objects (i.e., for vector spaces as objects one would choose linear maps as morphisms, and so on).

Definition 8.3.1 (Category)**.** A *category* $\mathcal{K}$ consists of the following three mathematical entities:

1. A *class* $\mathrm{Ob}(\mathcal{K})$ of objects
2. A class $\mathrm{Hom}(A, B)$ of *morphisms*, from $A \longrightarrow B$ such that $A, B \in \mathrm{Ob}(\mathcal{K})$.
   e.g. $\varphi : A \to B$ to mean $\varphi \in \mathrm{Hom}(A, B)$.
   Remark. The class of *all* morphisms of $\mathcal{K}$ is denoted $\mathrm{Hom}(\mathcal{K})$.
3. Given $A, B, C \in \mathrm{Ob}(\mathcal{K})$, a binary operation $\circ : Hom(A, B) \times Hom(B, C) \to Hom(A, C)$ called *composition*, satisfying:

(a) *(closed associativity)* Given $\varphi : A \to B$, $\psi : B \to C$ and $\phi : A \to C$ we have $\phi \circ (\psi \circ \varphi) = (\phi \circ \psi) \circ \varphi$.



(b) *(identity)* For any object X there is an identity morphism $id_X : X \to X$ such that for any $\varphi : A \to B$ we have $id_B \circ \varphi = \varphi = \varphi \circ id_A$.

It is also worth noting about what we mean by 'small' and 'large' categories.

Definition 8.3.2 (Small Category)**.** A category $\mathcal{K}$ is called small if both $\mathrm{Ob}(\mathcal{K})$ and $\mathrm{Hom}(\mathcal{K})$ are sets. If $\mathcal{K}$ is not small, then it is called large. $\mathcal{K}$ is called locally small if $\mathrm{Hom}(A, B)$ is a set for all $A, B \in \mathrm{Ob}(\mathcal{K})$.

Remark. Most important categories in mathematics are not small however, are locally small.

Most basic categories have as objects certain mathematical structures, and the structure-preserving functions as morphisms. A common list is given;

- *Pos* is the category of partially ordered sets and monotonic functions.
- *Top* is the category of topological spaces and continuous functions.
- *Rng* is the category of rings and ring homomorphisms.
- *Grp* is the category of groups and group homomorphisms.
- *Grph* is the category of graphs and graph homomorphisms.

8.3.1 Product

Let $A, B \in \mathrm{Ob}(\mathcal{K})$ for some category $\mathcal{K}$. A *product* of A and B is an object C, together with morphisms from $\alpha : C \to A$ and $\beta : C \to B$, such that the following property holds: if C' is any object, and any morphisms $\alpha' : C' \to A$ and $\beta' : C' \to B$, there is a unique morphism $\gamma : C' \to C$ such that the following diagram commutes:

The morphisms α and β are called the *canonical projections*. We denote the product of A and B as: $C = A \prod B$ or sometimes written $C = A \bigotimes B$.

8.4 Dual Category

Duality is a correpsondence between properties of a category $\mathcal{K}$ and the notion of *dual properties* of the "*op*posite category" $\mathcal{K}^{op}$. Suppose some proposition regarding a category $\mathcal{K}$, by interchanging the domain and codomain of each morphism as well as the order of composition, a corresponding dual proposition is obtained of $\mathcal{K}^{op}$.

Duality, as such, is the assertion that truth is invariant under this operation on propositions.

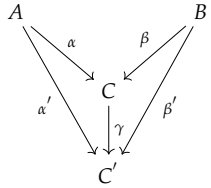
Lemma 8.4.1. $(\mathcal{K}^{op})^{op} = \mathcal{K}$.

Example 8.4.2. Suppose a monomorphism in a category $\mathcal{K}$ then an epimorphism is the categorial dual in the dual category $\mathcal{K}^{op}$. Prove this by diagram.

Remark. We make the notion of diagram and hence the idea of commutative diagrams rigous later.

8.4.1 Coproduct

Let $A, B \in \mathrm{Ob}(\mathcal{K})$ for some category $\mathcal{K}$. A *co*product (sum) of A and B is an object C, together with morphisms from $\alpha : A \to C$ and $\beta : B \to C$, such that the following property holds: if C' is any object, and any morphisms $\alpha' : A \to C'$ and $\beta' : B \to C'$, there is a unique morphism $\gamma : C \to C'$ such that the following diagram commutes:



Once again, the morphisms α and β are called the *canonical projections*. We denote the coproduct, or 'sum', of A and B as: $C = A \coprod B$ or sometimes written $C = A \bigoplus B$. Notice that all we have done is reverse all the morphisms (arrows) in a product to obtain the coproduct. This is, ofcourse, the usual method of finding a categorial dual.

8.5 Functors

A category is itself a type of mathematical structure and so, one can generalise the notion of a morphism thus preserve this structure by the notion of a functor. A functor associates to every object of one category an object of another category, and to every morphism in the first category a morphism in the second. Hence, functors are structure-preserving maps between categories and can be thought of as morphisms in the category of all (small) categories.

In particular, what we have done is define a category of categories and functors - the objects are categories, and the morphisms (between categories) are functors. By studying categories and functors, we are not merely studying a class of mathematical structures and the morphisms between them, we are studying the relationships between various classes of mathematical structures.

Definition 8.5.1 (Functor)**.** Let $\mathcal{C}$ and $\mathcal{K}$ be categories. A *functor* F from $\mathcal{C}$ to $\mathcal{K}$ is a mapping that:

1. associates to each object $X \in \mathcal{C}$ an object $F(X) \in \mathcal{K}$
2. associates to each morphism $\varphi : X \to Y \in \mathcal{C}$ a morphism $F(\varphi) : F(X) \to F(Y) \in \mathcal{K}$ satisfying:
   (a) $F(id_X) = id_{F(X)}$ for every object $X \in \mathcal{C}$
   (b) $F(\psi \circ \varphi) = F(\psi) \circ F(\varphi)$ for all morphisms $\varphi : X \to Y$ and $\psi : Y \to Z$

   Remark. That is, functors must preserve identity morphisms and composition of morphisms.

8.5.1 Types of Functors

Like many things in category theory, functors come in a kind of "dual" type in concepts; that of the *contravariant* and *covariant* functors, defined as follows:

Definition 8.5.2 (Covariant Functor)**.** A *covariant* functor F from a category $\mathcal{C}$ to a category $\mathcal{K}$, $F : \mathcal{C} \to \mathcal{K}$, consists of:

- for each object $X \in \mathcal{C}$, an object $F(X) \in \mathcal{K}$
- for each morphism $\varphi : X \to Y \in \mathcal{C}$, a morphism $F(\varphi) : F(X) \to F(Y)$

provided the following two properties hold:

- For every object $X \in \mathcal{C}$, $F(id_X) = id_{F(X)}$
- For all morphisms $\varphi : X \to Y$ and $\psi : Y \to Z$, $F(\psi \circ \varphi) = F(\psi) \circ F(\varphi)$

Definition 8.5.3 (Contravariant Functor)**.** A *contravariant* functor $F : \mathcal{C} \to \mathcal{K}$, is a 'reversed' covariant functor. In particular, for every morphism $\varphi : X \to Y \in \mathcal{C}$ must be assigned to a morphism $F(\varphi) : F(Y) \to F(X) \in \mathcal{K}$. Alternatively, contravariant functors act as covariant functors from the opposite category $\mathcal{C}^{op} \to \mathcal{K}$.

Functors have particular properties that essentially are the same as morphisms. This may seem obvious given that functors are a generalisation of the notion of a morphism. So, the following definitions should mostly be obvious if you reflect back to the corresponding morphism definitions above or simply break the word down grammatically.

Definition 8.5.4 (Endofunctor)**.** A *endofunctor* is a functor that maps a category to itself, sometimes called a *identity functor*.

Definition 8.5.5 (Bifunctor)**.** A *bifunctor* is a functor in *two* arguments. More formally, a bifunctor is a functor whose domain is a *product category*. The *Hom* functor is a natural example; it is contravariant in one argument while covariant in the other. Hence, the *Hom* functor is of the type $\mathcal{C}^{op} \times \mathcal{C} \to \mathbf{Set}$.

Definition 8.5.6 (Multifunctor)**.** A *multifunctor* is a generalisation of the functor concept to n variables, (e.g., A *bifunctor* is when $n = 2$)

8.5.2 Properties of Functors

However, some properties are more particular to functors and we review them here to families ourselves. A functor $F : \mathcal{A} \to \mathcal{B}$ is

1. *faithful* if for every parallel pair of morphisms $f, g : A \rightrightarrows A' \in \mathcal{A}$, one has $f = g$ whenever $F(f) = F(g)$. (Recall the definition of a monomorphism).
2. *full* if for every morphism $b : F(A) \to F(A') \in \mathcal{B}$, there exists a morphism $a : A \to A' \in \mathcal{A}$ such that $F(a) = b$ (Recall the definition of a epimorphism).
3. essentially *surjective* if for every object $B \in \mathcal{B}$, there exists an object $A \in \mathcal{A}$ with B isomorphic to $F(A)$

4. an *equivalence* if there exists a functor $F' : \mathcal{B} \to \mathcal{A}$ such that both $F \circ F'$ and $F' \circ F$ are naturally isomorphic to the identity functors; such a functor F' is called a *quasi-inverse* of F
5. an *isomorhpism* if there exists a functor $F' : \mathcal{B} \to \mathcal{A}$ such that both $F \circ F'$ and $F' \circ F$ are equal to the identity functors
6. *conserative* if it reflects isomorphisms; that is, $a : A \to A'$ is an isomorphism whenever $F(a) : F(A) \to F(A')$ is

Definition 8.5.7 (Functor category and the Yoneda embedding)**.** The Yoneda lemma suggests that instead of studying the *locally small* category $\mathcal{K}$, one should study the category of all functors of $\mathcal{K}$ into **Set**. Since the category **Set** is well understood, a functor of $\mathcal{K}$ into **Set** maybe seen as a *representation* of $\mathcal{K}$ in terms of known structures. See the *category of diagrams* later.

1. Given a category $\mathcal{K}$ and a small category $\mathcal{J}$, we denote by $\mathcal{K}^{\mathcal{J}}$ the category of functors from $\mathcal{J} \to \mathcal{K}$ and natural transformations of each functor from $\mathcal{J} \to \mathcal{K}$.
2. In case $\mathcal{K} = \mathbf{Set}$, we have the *Yoneda embedding*:
   $Y_{\mathcal{J}} : \mathcal{J}^{op} \to \mathbf{Set}^{\mathcal{J}}$ $Y_{\mathcal{J}}(X) = \mathcal{J}(X, -)$, which is full and faithful (Recall the definition of a bimorphism).

8.5.3 F-algebra

F-algebras have various applications, such as in computer science, creating concrete definitions of various algebraic data types (e.g., lists, trees, ...). A F-algebra is a structure defined according to some endofunctor $F : \mathcal{K} \to \mathcal{K}$ for some category of functors $\mathcal{K}$. That is,

Definition 8.5.8 (F-algebra)**.** A F-algebra for a endofunctor F is some object A in the category $\mathcal{K}$, coupled with a $\mathcal{K}$-morphism $\alpha : F(A) \to A$. We denote this F-algebra by the pair (A, α).

F-algebras also form a category as expected. Since, The homomorphism from a F-algebra (A, α) to F-algebra (B, β) is given by $\varphi : A \to B \in \mathcal{K}$ such that $\varphi \circ \alpha = \beta \circ F(\varphi)$. Hence, we have the following commutative diagram:

$$\begin{array}{ccc} F(A) & \xrightarrow{\alpha} & A \\ \downarrow{\scriptstyle F(\varphi)} & & \downarrow{\scriptstyle \varphi} \\ F(B) & \xrightarrow{\beta} & B \end{array}$$

Example 8.5.9 (Initial algebra)**.** Consider a functor $F : \mathbf{Set} \to \mathbf{Set}$, in particular, $F : X \to 1 \bigoplus X$. Then the set $\mathbb{N}$ endowered with the function $[zero, succ] : 1 \bigoplus \mathbb{N} \to N$, is a F-algebra. This initial algebra is denoted by the pair $(\mathbb{N}, [zero, succ])$.

Here 1 denotes the *terminal object* taken to be the singleton set $\{\ \}$ and $\bigoplus$ as the *coproduct*.

Remark. The categorial dual are called *F-co*algebras.

8.6 Monoids

A *monoid* is an abstract mathematical structure normally associated with that of a semigroup with identity. A semigroup with identity, or just monoid, is some set M together with a law of associative composition, $\forall x, y, z \in M : (x \circ y) \circ z = x \circ (y \circ z)$, that is both closed, $M \circ M \to M$, and has idenity, $\exists id_M \in M : id_M \circ x = x = x \circ id_M \, \forall x \in M$. That is,

Definition 8.6.1. A *monoid* is a set X together with a law of composition $\circ$ which is closed associative, and has identity $id_X \in X$.

More compactly in categorial terms,

Definition 8.6.2 (Monoid)**.** A category with exactly one object is called a *monoid*.

Example 8.6.3. Let $M = \mathbb{Z}$ and the law of composition $+$ as arithmetic addition with 0 as identity.

Example 8.6.4. A list is an example of a monoid with the empty list $()$ as identity and the law of composition $++$ as appending to the list.

8.7 Natural Transformations

A natural transformation provides a way of transforming one functor into another while respecting the internal structure (i.e. the composition of morphisms) of the categories involved. Hence, a natural transformation can be considered to be a "morphism of functors". Indeed this intuition can be formalized to define so-called functor categories. Natural transformations are, after categories and functors, one of the most basic notions of category theory and consequently appear in the majority of its applications.

Definition 8.7.1 (Natural Transformation)**.** Let F and G be functors both from the category $\mathcal{C}$ to $\mathcal{K}$. Then a *natural transformation* $\eta : F \to G$ associates to every object $X \in \mathcal{C}$ a morphism $\eta_X : F(X) \to G(X) \in \mathcal{K}$, called the *component* of η at X. The component morphism η_X is subject to the condition of *naturality*, that is, the diagram commutes for every morphism $\varphi : X \to Y \in \mathcal{C}$; or simply, $\eta_Y \circ F(\varphi) = G(\varphi) \circ \eta_X$.

We draw the commutative diagram in the category $\mathcal{K}$ know as the *naturality square*:

$$\begin{array}{ccc} F(X) & \xrightarrow{\eta_X} & G(X) \\ \downarrow{\scriptstyle F(\varphi)} & & \downarrow{\scriptstyle G(\varphi)} \\ F(Y) & \xrightarrow{\eta_Y} & G(Y) \end{array}$$

Remark. **N.B.** The naturality square is the key concept to understanding natural transformations and there is one of them for every morphism in $\mathcal{C}$ as stated above. Note this well.

Constructions are often "naturally related" - a vague notion, at first sight. This leads to the clarifying concept of natural transformation, a way to map one functor to another.

Remark. Various important constructions in mathematics can be studied in this context. Naturality is a principle, like that of general covariance in physics, that cuts deeper than is initially apparent.

8.8 Monads

Monads derived from Greek, *μοναϛ* (monas), "unit". The computer science interpretation is that of a unit of computation. Monads are like algebraic theories and that an algebra for a monad is a model of that theory. A monad is simply a functor with some extra 'stuff' that tells us what is going on in the associated theory. The notion of *algebras for a monad* generalises classical notions from universal algebra, and in this sense, monads can be thought of as *theories*. More formally, a monad or *(Kleisli triple)*, is an endofunctor coupled with two natural transformations, η and μ that satisfy so-called *coherence conditions*.

Definition 8.8.1 (Monad)**.** Given a category $\mathcal{K}$, a *monad* on $\mathcal{K}$ is the endofunctor $T : \mathcal{K} \to \mathcal{K}$ coupled with two natural transformations, $\eta : id_{\mathcal{C}} \rightrightarrows T$ and $\mu : T^2 \rightrightarrows T$ that satisfy the coherence conditions:

- $\mu \circ T\mu = \mu \circ \mu T$, and
- $\mu \circ T\eta = \mu \circ \eta T = id_T$.

Remark. N.B. η called *the unit* since it is a natural mapping from idenity and μ called *the multiplication* also noting the natural mapping from $T^2 = T \circ T$.

Recall the definition of a natural transformation of η and μ, that for every $X \in \mathrm{Ob}(\mathcal{K})$ there is a *component* morphism of η and μ at X, denoted by, η_X and μ_X respectively.
So, we have the following:

- $\eta_X : X \to T(X) \in \mathcal{K}$, read, "*the component of η at X is a morphism in $\mathcal{K}$ from X to T of X*", and,
- $\mu_X : T^2(X) \to T(X) \in \mathcal{K}$, read, "*the component of μ at X is a morphism in $\mathcal{K}$ from T^2 of X to T of X*".

Hence, the coherence conditions are shown the by commutative diagrams for every component $X \in \mathrm{Ob}(\mathcal{K})$

For the existence idenity by *the left and right unit triangles*:

$$\begin{array}{ccc} T(X) & \xrightarrow{\eta_{T(X)}} & T^2(X) \\ {\scriptstyle T(\eta_X)}\downarrow & \searrow & \downarrow{\scriptstyle \mu_X} \\ T^2(X) & \xrightarrow[\mu_X]{} & T(X) \end{array}$$

and for that of *the associativity square* by:

$$\begin{array}{ccc} T^3(X) & \xrightarrow{T(\mu_X)} & T^2(X) \\ {\scriptstyle \mu_{T(X)}}\downarrow & & \downarrow{\scriptstyle \mu_X} \\ T^2(X) & \xrightarrow[\mu_X]{} & T(X) \end{array}$$

Since these are natural transformations, we can write these without X and by writing $\eta_{T(X)}$ as ηT and $T(\eta_X)$ as $T\eta$, for example. Since the commutative diagrams in the category $\mathcal{K}$ are component wise, that is, in terms of X.

Remark (Comonad). A *C*omonad, or just *cotriple*, is the categorical dual of a Monad. That is, A comonad for a category $\mathcal{K}$ is a monad for the opposite category $\mathcal{K}^{op}$.

Example 8.8.2 (Monad for monoids)**.** From the definition of a monoid, take the category $\mathcal{K} =$ **Set** and a functor $T : X \to X^*$ where X is the "set of words" in X.

Remark. A word is just a list of objects in X e.g., (x_1, x_2, x_3) or simply the empty list $()$.

Therefore, we define the natural transformations component wise, as: $\eta_X : X \to X^*$ and $\mu_X : X^{**} \to X^*$.

Remark. To understand $\mu_X : X^{**} \to X^*$ consider the example: $(a,(b,c)) \to (a,b,c)$

Finish me..... TODO.

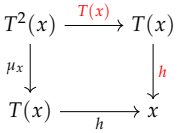
Note. Haskell Note:

- η is the Monadic equivalent of the "return" function and,
- μ is the Monadic equivalent of the "join" operator.

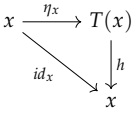
8.8.1 Algebras for Monads

Given the monad (T,η,μ) on a category $\mathcal{K}$. A T-algebra denoted by (x,h), where $x \in \mathrm{Ob}(\mathcal{K})$ coupled with $h \in \mathrm{Hom}(\mathcal{K})$ with $h : T(x) \to x \in \mathcal{K}$. The morphism h is called the structure map of the algebra T provided the usual diagrams commute (i.e., the coherence conditions are satisfied).

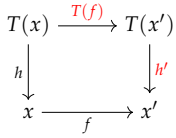
The *the associativity square* by:



and the *right unit triangle* (idenity) by:



A morphism $f : (x,h) \to (x',h')$ of *T-algebras* is a morphism $f : x \to x' \in \mathcal{K}$ given that the following diagram commutes:



8.9 Groupoids

The extension from *group* to *groupoid* is motivated with the desire to describe reversible mappings which may traverse a number of states. Intuitively, in *group theory* we study mappings of the form $\tau : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ that are total functions. Whereas in *groupoid theory* one considers a set of compositable and invertible mappings of the form $\phi \circ \psi : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ where ϕ and ψ need only be partial functions on $\mathcal{G}$.

Definition 8.9.1 (Groupoid)**.** A **groupoid** is a small category in which every morphism is an isomorphism.

Thus, as a result one may view groups as a groupoid with one object. More precisely, a groupoid X with one object x is uniquely determined by its automorphism group $\mathcal{G} = \mathrm{Aut}(x)$. Consequently, any group may be regarded as a groupoid with one object.

Notice that can naturally view morphisms between *groupoids* are *functors* between categories.

Part III

Higher Computational Theory

..

Appendix

Index of symbols

From the Introduction:

We present *Homotopy type theory* as the corner stone to understanding both mathematics and programming.

Get a free copy of the book at http://victoredwardocallaghan.github.io/book/.