Category Theory and the λ-Calculus

Anna Williams

1 Introduction

Proof assistants are computer programs which can be used to prove mathematical theorems. They are particularly interesting because they link theory from mathematics and computer science together, namely that of category theory and type theory. One famous example of the use of proof assistants was the proof of the Four Colour Theorem by Georges Gonthier using the proof assistant Coq. The initial proof by Appel and Haken was quite controversial as it involved a large number of computer verified cases. These were difficult to verify and hard to trust due to the nature of programming being error prone (Gonthier, 2005). The new proof by Gonthier used similar methods to Appel and Haken, but instead used Coq for the proof; this proof was more widely accepted as only the core of the program had to be trusted in order to trust the whole proof.

In this project we will explore some of the theory behind proof assistants, in particular category theory and type theory through the lens of the simply typed λ-calculus, a model of computation. At the end of the paper we combine these and study the category of the simply type λ-calculus.

2 Key Ideas in this Project

In this section we look at sets along with set functions in order to illustrate some of key concepts of category theory.

2.1 Universal Property

One common idea in category theory is that of a universal property (Mac Lane, 1971). The idea of this is that we can show properties of 'objects' (things we want to study) based upon relations between them. For example, if we consider the Cartesian product in sets, the common set-theoretic definition is as given below.

Definition 2.1. Let A, B be sets, then the *Cartesian product*, denoted $A \times B$ is defined as

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

We let π_1 and π_2 denote the projections defined by

$$\begin{array}{ll} \pi_1 : A \times B \to A & \pi_2 : A \times B \to B \\ \pi_1(a, b) \mapsto a & \pi_2(a, b) \mapsto b \end{array}$$

However, we can instead define this property based upon relations between sets. This is done using the following definition.

Definition 2.2. Let $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$. Then for every set, C, for which there exists $f : C \to A$ and $g : C \to B$, there exists a unique $u : C \to A \times B$ such that $\pi_1 \circ u = f$ and $\pi_2 \circ u = g$.

Note. We can define this u for sets by $u(x) = (f(x), g(x))$.

2.2 Commutative Diagrams

One other important idea in Category Theory is that of a commutative diagrams.

Definition 2.3. A diagram *commutes* if for any path from set A to B composing the functions on the labels gives equal functions. For example, the following diagram is said to commute if $g \circ f = f' \circ g'$.

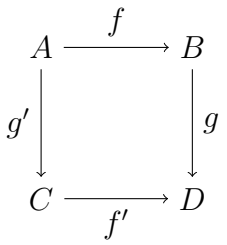


Figure 1: A commutative square

These are of particular interest because they give us a method of proof that is more clear than typical reasoning. For example, if we wanted to show that $g \circ f = f' \circ g'$, then proving that Figure 1 commutes exactly gives the proof.

3 Category Theory

3.1 Categories

Definition 3.1. A *category*, denoted $\mathbf{C}$, consists of

(i) a collection of *objects*, denoted $\operatorname{obj} \mathbf{C}$,

(ii) a collection of *morphisms*, denoted $\operatorname{mor} \mathbf{C}$, which have a domain and codomain in $\operatorname{obj} \mathbf{C}$. Additionally, if f has domain A and codomain B, then we write $f : A \to B$,

(iii) a composition operation, $\circ$, such that for morphisms $f, g \in \operatorname{mor} \mathbf{C}$ satisfying $\operatorname{dom} f = \operatorname{cod} g$, $f \circ g$ is a morphism from $\operatorname{dom} g$ to $\operatorname{cod} f$, and

(iv) an identity morphism for every object $A \in \operatorname{obj} \mathbf{C}$, denoted $1_A : A \to A$.

Where we require the following axioms hold:

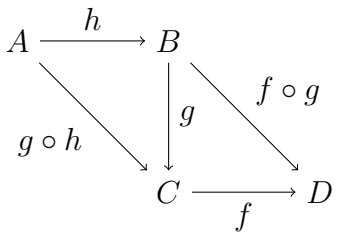
(Associativity) Let $A, B, C, D \in \operatorname{obj} \mathbf{C}$ and $f : C \to D, g : B \to C, h : A \to B \in \operatorname{mor} \mathbf{C}$. Then

$$f \circ (g \circ h) = (f \circ g) \circ h.$$

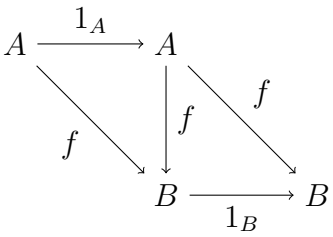
(Identity) Let $A, B \in \operatorname{obj} \mathbf{C}$ and $f : A \to B \in \operatorname{mor} \mathbf{C}$. Then

$$f \circ 1_A = f = 1_B \circ f.$$

Notation. We typically use upper-case letters $A, B, C, \ldots$ to denote objects and use lower-case letter $f, g, h, \ldots$ to denote morphisms.



(a) Associativity axiom



(b) Identity axiom

Figure 2: Commutative Diagrams for the Axioms of a Category

We can express the axioms using commutative diagrams (Asperti, 1991). As Figure 2a commutes, we can see that $(f \circ g) \circ h$ is equivalent to $f \circ (g \circ h)$. This is exactly the axiom of associativity. The axiom of identity is shown in Figure 2b similarly.

Examples.

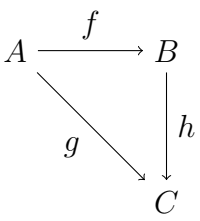
(i) One simple example would be the category on one object, A

(ii) A more concrete example is the category **Set**, for which all objects are sets and morphisms are functions between sets.

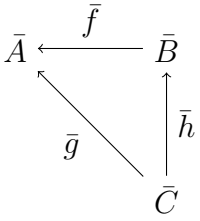
(iii) Another example is the category **Vec**, for which all objects are vector spaces and morphisms are linear transformations.

Definition 3.2. Let $\mathbf{C}$ be a category, then we define the *dual* of $\mathbf{C}$, denoted $\mathbf{C}^{\text{op}}$, by the operation of reversing all morphisms. More formally, for every morphism $f : A \to B$ in $\mathbf{C}$ there exists a corresponding dual morphism $\bar{f} : \bar{B} \to \bar{A}$ in $\mathbf{C}^{\text{op}}$, where $\bar{A}, \bar{B}$ in $\mathbf{C}^{\text{op}}$ correspond to the objects A, B.

Example. Define the category $\mathbf{C}$ with obj $\mathbf{C} = \{A, B, C\}$ and mor $\mathbf{C} = \{f, g, h\}$, where $f : A \to B$, $g : A \to C$ and $h : B \to C$. We can then form the dual as shown in Figure 3 (we omit the identity morphisms from the diagram for clarity).



(a) The unmodified category



(b) The opposite category

Figure 3: A pair of diagrams showing a category and its dual

Definition 3.3 (Crole, 1993)**.** Let $\mathbf{C}$ and $\mathbf{D}$ be categories, then a *functor*, $F : \mathbf{C} \to \mathbf{D}$ is a pair of maps

(i) $F_{\text{obj}} : \text{obj}\,\mathbf{C} \to \text{obj}\,\mathbf{D}$, and

(ii) $F_{\text{mor}} : \text{mor}\,\mathbf{C} \to \text{mor}\,\mathbf{D}$ (we write Ff to mean the morphism given by $F_{\text{mor}}\,(f)$, where $f \in \text{mor}\,\mathbf{C}$).

We denote by Ff, $F_{\text{mor}}\,f$ and by FA, $F_{\text{obj}}\,A$.
These maps are such that for any morphisms $f : B \to C, g : A \to B \in \text{mor}\,\mathbf{C}$ we have that

(a) Ff is a morphism from FB to FC, that is $Ff : FB \to FC$,

(b) $F(f \circ g) = (Ff) \circ (Fg)$, and

(c) $F(1_A) = 1_{FA}$.

Examples.

- One simple functor is the inclusion functor. This includes a smaller category into a larger one. For example we could have the category with one object, the set of natural numbers, and map it to the category **Set**.
- Now that we have the idea of a functor, we can define the category **Cat**, the category of categories, where objects are categories and morphisms are functors. We can use this category to define an interesting example of a functor, this being $F : \mathbf{Cat} \to \mathbf{Gph}$, where **Gph** denotes the category of digraphs. This functor takes a category to its underlying digraph, where vertices are mapped to objects and edges are mapped to morphisms between objects.
- Another example is the forgetful functor that 'forgets' the structure of objects contained within it. For example if we have the category **Grp** whose objects are groups and morphisms are group homomorphisms, then the forgetful functor maps each group to a set and each homomorphism to a mapping.

3.2 Morphisms

To introduce properties of morphisms, we first understand some properties of functions on sets. Which will help in the understanding of their categorical generalisations.

3.2.1 Properties of Functions

Proposition 3.4. *Let A, B be sets and f be a function $f : A \to B$. Then f is injective if and only if for all $g, h : C \to A$*

$$f \circ g = f \circ h \implies g = h.$$

Proof.
($\implies$) Let f be injective, we have for all $c \in C$ that

$$f \circ g(c) = f \circ h(c) \implies f(g(c)) = f(h(c)).$$

Then by definition of injectivity for f,

$$g(c) = h(c)$$

for all $c \in C$. Therefore $g = h$ as required.
($\impliedby$) Let $f \circ g = f \circ h \implies g = h$ for all $g, h : C \to A$. Choose g, h surjective with $f \circ g = f \circ h$. Then for all $c \in C$,

$$f(g(c)) = f(h(c)) \implies f \circ g(c) = f \circ h(c) \implies g(c) = h(c).$$

As g, h surjective, $g(C) = A$ and thus we have that f is injective as $f(a) = f(b) \implies a = b$ for all $a, b \in A$. □

Proposition 3.5. *Let A, B be sets and $f : A \to B$ be a function. Then f is surjective if and only if*

$$g \circ f = h \circ f \implies g = h.$$

Proof.
($\implies$) Let f be surjective, then given that $g \circ f = h \circ f$

$$g \circ f(a) = h \circ f(a)$$

for all $a \in A$. As f is surjective $f(A) = B$, so we have that

$$g(b) = h(b)$$

for all $b \in B$. Therefore $g = h$ as required.
($\impliedby$) Let $g \circ f = h \circ f \implies g = h$, then define g and h by the following

$$g(x) = \begin{cases} 1 \text{ if } x \in \text{im}\,(f) \\ 0 \text{ otherwise} \end{cases}$$

$$h(x) = 1$$

We then have that $g \circ f = h \circ f \implies g = h$, that is that $g(x) = 1 = h(x)$. Thus $\text{im}\,(f) = B$, so f is surjective as required. □

In this definition, we bring about the idea of monomorphisms and epimorphisms, which correspond to injective and surjective functions respectively.

Definition 3.6. Let $\mathbf{C}$ be a category and $f : A \to B$ be a morphism in $\text{mor}\,\mathbf{C}$. Then we say that

- f is a *monomorphism* if and only if

  $$f \circ g = f \circ h \implies g = h$$

  for all morphisms $g, h : C \to A$, and
- f is an *epimorphism* if and only if

  $$g \circ f = h \circ f \implies g = h$$

  for all morphisms $g, h : B \to C$.

Definition 3.7. We call $f : A \to B$ an *isomorphism* if and only if there exists some morphism $g : B \to A$ such that

$$g \circ f = 1_A \text{ and } f \circ g = 1_B$$

Proposition 3.8. *An isomorphism is both a monomorphism and an epimorphism.*

Proof. Let $f : A \to B$ be an isomorphism, where $g : B \to A$ is such that $g \circ f = 1_A$ and $f \circ g = 1_B$. Then notice that for $h, h' : C \to A$ such that $f \circ h = f \circ h'$, we have that

$$\begin{aligned} f \circ h &= f \circ h' \\ g \circ f \circ h &= g \circ f \circ h' \\ 1_A \circ h &= 1_A \circ h' \\ h &= h' \end{aligned}$$

that is, f is a monomorphism. Similarly for $m, m' : B \to C$, if $m \circ f = m' \circ f$ then

$$\begin{aligned} m \circ f &= m' \circ f \\ m \circ f \circ g &= m' \circ f \circ g \\ m \circ 1_B &= m' \circ 1_B \\ m &= m' \end{aligned}$$

that is, f is an epimorphism. Therefore f is both a monomorphism and an epimorphism as required. □

Remark. The reverse implication is true in **Set** (this is because monomorphisms are exactly injective functions and epimorphisms are surjective functions (Awodey, 2010)), but this is not true in general.

3.3 Initial and Terminal Objects

In sets, the empty set and singleton sets have unique properties. The empty set is defined by the property that there is only one function from the empty set to every other set. The singleton set is defined by the property that there is one function from every set to the singleton set. It is these properties that give us universal properties for initial and terminal objects.

Proposition 3.9. *The property that there is a single function from the empty set defines the empty set uniquely.*

Proof. The proof, with ideas from Walters (1991) is as follows. Assume that there is some other set with this property Z. Then there exists two functions from Z, $1_Z : Z \to Z$ and $\alpha : Z \to \varnothing$ and two functions from the empty set $1_\varnothing : \varnothing \to \varnothing$ and $\beta : \varnothing \to Z$. We thus have

$$\alpha \circ \beta : Z \to Z \text{ and } \beta \circ \alpha : \varnothing \to \varnothing.$$

As the functions 1_Z and $1_\varnothing$ are unique, we must have that

$$\alpha \circ \beta = 1_Z \text{ and } \beta \circ \alpha = 1_\varnothing.$$

Therefore $Z = \varnothing$ as required. □

Note. The proof for the defining property of the singleton set is similar, though note that each set with the property is not equal, but isomorphic.

Definition 3.10. Let $\mathbf{C}$ be a category. Then

- an *initial* object, denoted 0, in $\mathbf{C}$ is the object for which there is a unique morphism $f : 0 \to A$ to every object $A \in \text{obj}\,\mathbf{C}$ and
- a *terminal* object, denoted 1, in $\mathbf{C}$ is the object for which every object, A, has a unique morphism, denoted $! : A \to 1$.

Note. Terminal and initial objects need not exist in categories.

Examples.

- One example, as we have seen, is that of the empty set and the singleton set for **Set**.
- In the category **Grp** of groups, the trivial group is both an initial and a terminal object. It is initial because the single element in the trivial group is the identity, so it must therefore map to the identity in any other group, making this morphism unique for each group. It is terminal because it has one element, so there is only one map to it.

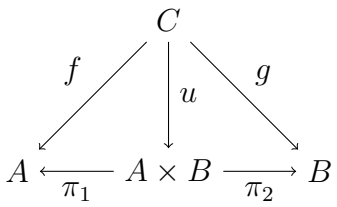
4 Cartesian Closed Categories

In this section we look at how the ideas of Cartesian product and functions on sets present themselves in a categorical form. We use this to define the product of objects and the exponential of objects respectively.

4.1 Products

In sets we have the idea of a Cartesian product and as explained in the introduction (Definition 2.2), there is a universal property which models this. It is this property which is used to expand the idea into category theory.

Definition 4.1. The *categorical product of objects* A, B is the object $A \times B$ with morphisms $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ such that for any object C with morphisms $f : C \to A$ and $g : C \to B$ there exists a unique morphism $u : C \to A \times B$ such that $\pi_1 \circ u = f$ and $\pi_2 \circ u = g$. That is, the following diagram commutes:



Definition 4.2 (Asperti, 1991)**.** A Cartesian Category (CC) is a category which has

(i) a terminal object, and

(ii) a categorical product for each pair of objects.

4.2 Exponentials

In set theory, we might want to look at encoding functions into sets. Following Awodey (2010) consider the value of the function $f : A \times B \to C$ at (x, y)

$$f(x, y) : C.$$

Let us fix some value $a \in A$ and let $y \in B$ vary then

$$f(a, y) : B \to C.$$

Introducing the notation C^B to be the set of functions $B \to C$, we have that $f(a, y) \in C^B$. Letting $a \in A$ vary, we can then define some function $\tilde{f} : A \to C^B$, where $a \mapsto f(a, y)$. We call the function that gives such a corresponding function in general Λ. That is, $\Lambda(f) = \tilde{f}$. If we want to obtain f from $\Lambda(f)$, we can do the following.

$$f(a, b) = \Lambda(f)(a)(b)$$

We can therefore see the correspondence between functions on $A \times B \to C$ and $A \to C^B$. This property is that of *currying*. Using this and an evaluation function eval, we can define the universal property of exponentials for sets.

Definition 4.3. For any set A and functions $\Lambda : (C \times A \to B) \to (C \to B^A)$, $f : A \times B \to C$, and $\text{eval} : B^A \times A \to B$, we have that

$$\text{eval} \circ (\Lambda(f) \times 1_B) = f$$

or equivalently

$$\text{eval}(\Lambda(f)(a), b) = f(a, b).$$

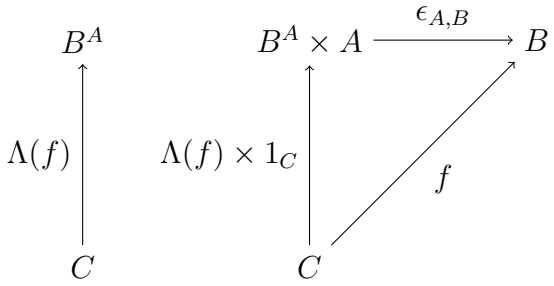
Using this property we can describe the idea of an exponential, which encodes morphisms of a category in objects as follows.

Definition 4.4 (Asperti, 1991)**.** The *exponent* of objects A and B is the object B^A along with maps $\epsilon : B^A \times A \to B$, called the evaluation map, and $\Lambda : (A \times B \to C) \to (A \to C^B)$. Such that

i) $\epsilon(\Lambda(f) \times 1_B) = f$

ii) $\Lambda(\epsilon(h \times 1_A)) = h$

We can express these properties using the following commutative diagram.



Definition 4.5 (Asperti, 1991)**.** A category **C** is a Cartesian Closed Category (CCC) if

(i) **C** is Cartesian, and

(ii) every pair of objects has an exponent.

5 The Simply Typed λ-Calculus

The simply typed λ-calculus is a language (set of strings) used to model computation. It consists of function definitions, function applications, variables and pairs. In this section we describe the language which represents the simply typed λ-calculus and explore the corresponding category. The highpoint we reach is that the category of the simply typed λ-calculus is in fact a Cartesian closed category.

5.1 Describing the Language

We give the language recursively by showing the different possible ways that a term can be written, as well as its intended computational meaning which we make precise later. An M represents a term that is in the language.

- $\lambda x.M$ - an abstraction, this defines a function with input x and that is taken into M,
- $M\ M$ - an application, the right term applied to the left,
- A variable from the set of infinite variables,
- $\langle M, M \rangle$ - a pair of terms,
- $\texttt{fst}\ M$ - represents the projection of the first element from a pair of terms,
- $\texttt{snd}\ M$ - represents the projection of the second element from a pair of terms.

Examples. We first look at some example term derivations to show how we can build up term.

- If we look at the term $\lambda x.\texttt{fst}\ x$, we derive it in the following way:

Current Term	Step from previous term
M	The start term
$\lambda x.M$	Expanding M to be a lambda abstraction
$\lambda x.\texttt{fst}\ M$	Expanding M to be $\texttt{fst}\ M$
$\lambda x.\texttt{fst}\ x$	Expanding M to be a variable, x

- If we look at the term $\langle y, z \rangle$, we derive it in the following way:

Current Term	Step from previous term
M	The start term
$\langle M, M \rangle$	Expanding M to be a pair of terms
$\langle y, M \rangle$	Expanding the first M to be a variable y
$\langle y, z \rangle$	Expanding the second M to be the variable z

5.1.1 Types

When talking about computation, it is often useful to reason about the *type* of a given function. A type is a collection of items which typically fit a certain set of properties. For example, the natural numbers can be added together or rational numbers can be divided, it is useful to distinguish these because division is not always defined on the natural numbers.

To give the collection of types of terms, we introduce a set of ground types, which are the types we wish to study computation on. Then for any two types S, T, we introduce the pair of types, $S \times T$ which corresponds to the pair of terms of types S and T and the function type $S \to T$, which corresponds to an abstraction term, where x is of type S and M is of type T.

5.2 Semantics

Now that we have defined the language, we need some way to reason precisely about the meanings of terms. To do this we look at free variables, variable substitution, and term equivalences.

5.2.1 Free Variables

When talking about terms, we would like to know which variables are defined within the term, and those which aren't. The variables which are defined outside of the scope of a term are known as free variables (FV). We can derive the free variables of a term as follows.

$$\begin{aligned} FV(x) &= \{x\} \\ FV(\langle N, M\rangle) &= FV(N) \cup FV(M) \\ FV(\texttt{fst}\ M) &= FV(M) \\ FV(\texttt{snd}\ M) &= FV(M) \\ FV(\lambda x.M) &= FV(M) \setminus \{x\} \\ FV(MN) &= FV(M) \cup FV(N) \end{aligned}$$

Note. Usually it is clear from context what the free variables are. For example the term $\lambda x.yx$ has y as a free variable. However both for computational reasons and for clarity it is worth defining how free variables are derived. See that,

$$\begin{aligned} FV(\lambda x.yx) &= FV(yx) \setminus \{x\} \\ &= FV(y) \cup FV(x) \setminus \{x\} \\ &= \{y\} \cup \{x\} \setminus \{x\} \\ &= \{y, x\} \setminus \{x\} \\ &= \{y\}. \end{aligned}$$

5.2.2 Computationally Equivalent Terms

When looking at terms of computation it seems natural to define what one step in this computation would be. We do this via a relation called "beta reduction", the symbol $\rightarrow_\beta$ denoting one step of beta reduction. The beta reductions are defined below

- $(\lambda x.M)N \rightarrow_\beta M$ with all occurrences of x replaced with the term N - this is the application rule, allowing us to apply an argument to a lambda term,
- $\texttt{fst}\ \langle M, N\rangle \rightarrow_\beta M$ - this projects the first term in a pair, and
- $\texttt{snd}\ \langle M, N\rangle \rightarrow_\beta N$ - this projects the second term in a pair.

We then give an equivalence based on whether two terms beta reduce, or compute, to the same value, the first three equivalences given by their beta reductions and the last two known as η-reductions.

- $(\lambda x.M)N = M$ with all occurrences of x replaced with the term N - derived from the β reduction
- $\texttt{fst}\ \langle M, N\rangle = M$ - derived from the β reduction
- $\texttt{snd}\ \langle M, N\rangle = N$ - derived from the β reduction.
- $\langle \texttt{fst}\ M, \texttt{snd}\ M\rangle = M$ - This rule allows us to simplify pairs
- $\lambda x.Mx = M$ where $x \notin FV(M)$ - This rule anticipates application

6 Categorical Semantics

6.1 Definition of the Category

Definition 6.1. We define the category of the simply typed λ-calculus to have objects be types and morphisms be lambda terms. We then define the identity morphism to be the lambda term $\lambda x.x$ and the composition operation $f \circ g$ to be the lambda term $\lambda x.f(gx))$ (Awodey, 2010).

We now prove that this definition satisfies the axioms for a category.

(Associativity) Let A, B, C, D be types and $f : C \to D, g : B \to C, h : A \to B$ be lambda terms. Then we need to show that $f \circ (g \circ h) = (f \circ g) \circ h) f \circ (g \circ h) = (f \circ g) \circ h)$.

$$\begin{aligned} f \circ (g \circ h) &= f \circ (\lambda x.g(hx)) \\ &= \lambda y.f((\lambda x.g(hx))y) \\ &= \lambda y.f(g(hy)) \\ &= \lambda y.(\lambda x.f(gx))(hy) \\ &= (\lambda x.f(gx)) \circ h \\ &= (f \circ g) \circ h \end{aligned}$$

(Identity) Let A, B be types and $f : A \to B$ be a lambda term, then we need to show that $f \circ 1_A = f = 1_B \circ f$. Firstly, we show that $f \circ 1_A = f$.

$$\begin{aligned} f \circ 1_A &= \lambda x.f(1_A x) \\ &= \lambda x.f((\lambda y.y)x) \\ &= \lambda x.fx \\ &= f \end{aligned}$$

Then we show that $f = 1_B \circ f$.

$$\begin{aligned} f &= \lambda x.(fx) \\ &= \lambda x.(\lambda y.y)(fx) \\ &= \lambda x.1_B(fx) \\ &= 1_B \circ f \end{aligned}$$

The morphisms and objects satisfy the axioms, therefore the category we have defined is valid.

6.2 Cartesian Property

The Cartesian property of a category encapsulates the idea of pairs being objects. We have a pair type for every two types and the pair of any two terms as a term, therefore it feels natural to say that the simply typed λ-calculus is in fact a Cartesian category. We prove this in the following proposition.

Proposition 6.2. *The category of the simply typed λ-calculus is a Cartesian category.*

Proof. To show that a category is Cartesian, we have to show that every pair of a objects has a product and that the category has a terminal object.

For an object C with morphisms $f : C \to A$ and $g : C \to B$, we define the unique morphism $c : C \to A \times B$ to be the lambda term $c = \lambda x.\langle fx, gx\rangle$ and projections $\pi_1 = \lambda x.\texttt{fst}\ x$ and

$\pi_2 = \lambda x.\texttt{snd}\, x$. It now suffices to show that $\pi_1 \circ c = f$ and $\pi_2 \circ c = g$ and prove uniqueness of this term.

We first show that $\pi_1 \circ c = f$. By definition $c \circ \pi_1$ is $\lambda x.\pi_1\, c\, x$. We now show that this is equivalent to f.

$$\begin{aligned}
\lambda x.\pi_1\, c\, x &= \lambda x.\pi_1\, (\lambda y.\langle fy, gy\rangle\, x \\
&= \lambda x.\pi_1(\langle fx, gx\rangle) \\
&= \lambda x.(\lambda z.\texttt{fst}\, z)(\langle fx, gx\rangle \\
&= \lambda x.\texttt{fst}\, \langle fx, gx\rangle \\
&= \lambda x.fx \\
&= f
\end{aligned}$$

as required. We now show that $\pi_2 \circ c = g$. By definition $c \circ \pi_2$ is $\lambda x.\pi_2 cx$. We now show that this is equivalent to g.

$$\begin{aligned}
\lambda x.\pi_2 cx &= \lambda x.\pi_2(\lambda y.\langle fy, gy\rangle x \\
&= \lambda x.\pi_2\langle fx, gx\rangle \\
&= \lambda x.(\lambda z.\texttt{snd}\, z)\langle fx, gx\rangle \\
&= \lambda x.\texttt{snd}\, \langle fx, gx\rangle \\
&= \lambda x.gx \\
&= g
\end{aligned}$$

To show that c is unique, assume there is some other function d with this property i.e. $\pi_1 \circ d = f$ and $\pi_2 \circ d = g$. Then

$$\begin{aligned}
c &= \lambda x.\langle fx, gx\rangle \\
&= \lambda x.\langle(\pi_1 \circ d)x, (\pi_2 \circ d)x\rangle \\
&= \lambda x.\langle(\lambda y.\pi_1(dy))x, (\lambda z.\pi_2(dz))x\rangle \\
&= \lambda x.\langle(\lambda y.(\lambda a.\texttt{fst}\, a)(dy))x, (\lambda z.(\lambda b.\texttt{snd}\, b)(dz))x\rangle \\
&= \lambda x.\langle(\lambda y.\texttt{fst}\, (dy))x, (\lambda z.\texttt{snd}\, (dy))x\rangle \\
&= \lambda x.\langle\texttt{fst}\, (dx), \texttt{snd}\, (dx)\rangle \\
&= \lambda x.dx \\
&= d
\end{aligned}$$

We have shown that c is unique and satisfies the required properties therefore a product must exist between each pair of objects. The terminal object, denoted 1, is the type in the base types which has one element, therefore any morphism from an object A is necessarily unique as it maps all elements of A to the single element of 1.

□

6.3 Cartesian Closed Property

The key idea of Cartesian closed categories is capturing not just pairs as objects, but also functions as objects. By definition for any two types we have the type of a function between them, therefore it seems reasonable that the category of the simply typed λ-calculus is in fact Cartesian closed. We prove this with the following proposition.

Proposition 6.3. *The category of the simply typed λ-calculus is a Cartesian closed category.*

Proof. Following Awodey (2010), we define eval to be

$$\epsilon := \lambda x.(\texttt{fst}\ x)(\texttt{snd}\ x)$$

and Λ to be

$$\Lambda := \lambda a.\lambda z.\lambda x.a\langle z, x\rangle.$$

It then suffices to show that $\epsilon \circ (\Lambda f \times 1_A) = f$ and also that $\Lambda(\epsilon \circ (h \times 1_A)) = h$. We first prove that $\epsilon \circ (\Lambda f \times 1_A) = f$.

$$\begin{aligned}
\epsilon \circ (\Lambda(f) \times 1_A) &= \epsilon \circ [((\lambda a.\lambda z.\lambda x.a\langle z, x\rangle)f) \times (\lambda y.y)] \\
&= \epsilon \circ [(\lambda z.\lambda x.f\langle z, x\rangle) \times (\lambda y.y)] \\
&= \epsilon \circ [\lambda w.\langle \lambda z.\lambda x.f\langle z, x\rangle \texttt{fst}\ w, (\lambda y.y)\texttt{snd}\ w\rangle] \\
&= \lambda v.\epsilon([\lambda w.\langle \lambda z.\lambda x.f\langle z, x\rangle \texttt{fst}\ w, \texttt{snd}\ w\rangle]v) \\
&= \lambda v.\epsilon(\langle \lambda z.\lambda x.f\langle z, x\rangle \texttt{fst}\ v, \texttt{snd}\ v\rangle) \\
&= \lambda v.\epsilon(\langle \lambda x.f\langle \texttt{fst}\ v, x\rangle, \texttt{snd}\ v\rangle) \\
&= \lambda v.(\lambda x.(\texttt{fst}\ x)(\texttt{fst}\ x))\langle \lambda x.f\langle \texttt{fst}\ v, x\rangle, \texttt{snd}\ v\rangle \\
&= \lambda v.(\texttt{fst}\ \langle \lambda x.f\langle \texttt{fst}\ v, x\rangle, \texttt{snd}\ v\rangle, \texttt{snd}\ \langle \lambda x.f\langle \texttt{fst}\ v, x\rangle, \texttt{snd}\ v\rangle) \\
&= \lambda v.(\lambda x.f\langle \texttt{fst}\ v, x\rangle)(\texttt{snd}\ v) \\
&= \lambda v.f\langle \texttt{fst}\ v, \texttt{snd}\ v\rangle \\
&= \lambda v.fv \\
&= f.
\end{aligned}$$

And now show that $\Lambda(\epsilon \circ (h \times 1_A)) = h$

$$\begin{aligned}
\Lambda(\epsilon \circ (h \times 1_A)) &= \Lambda(\epsilon \circ [\lambda x.\langle h(\texttt{fst}\ x), 1_A(\texttt{snd}\ x)\rangle]) \\
&= \Lambda(\epsilon \circ [\lambda x.\langle h(\texttt{fst}\ x), (\lambda y.y)(\texttt{snd}\ x)\rangle]) \\
&= \Lambda(\epsilon \circ [\lambda x.\langle h(\texttt{fst}\ x), \texttt{snd}\ x\rangle]) \\
&= \Lambda(\lambda w.\epsilon((\lambda x.\langle h(\texttt{fst}\ x), \texttt{snd}\ x\rangle)w)) \\
&= \Lambda(\lambda w.\epsilon(\langle h(\texttt{fst}\ w), (\texttt{snd}\ w)\rangle)) \\
&= \Lambda(\lambda w.(\lambda x.(\texttt{fst}\ x)(\texttt{snd}\ x))(\langle h(\texttt{fst}\ w), (\texttt{snd}\ w)\rangle)) \\
&= \Lambda(\lambda w.(\texttt{fst}\ \langle h(\texttt{fst}\ w), (\texttt{snd}\ w)\rangle)(\texttt{snd}\ \langle h(\texttt{fst}\ w), (\texttt{snd}\ w)\rangle)) \\
&= \Lambda(\lambda w.h(\texttt{fst}\ w)(\texttt{snd}\ w)) \\
&= \lambda a.\lambda z.\lambda x.a\langle z, x\rangle)(\lambda w.h(\texttt{fst}\ w)(\texttt{snd}\ w)) \\
&= \lambda z.\lambda x.(\lambda w.h(\texttt{fst}\ w)(\texttt{snd}\ w))\langle z, x\rangle \\
&= \lambda z.\lambda x.h(\texttt{fst}\ \langle z, x\rangle)(\texttt{snd}\ \langle z, x\rangle) \\
&= \lambda z.\lambda x.hzx \\
&= \lambda z.hz \\
&= h
\end{aligned}$$

□

7 Further Work

In this project we have seen a lot of the theory behind proof assistants, but how does this work in practice? The key idea is given by that of the Curry-Howard correspondence. This correspondence links type theory — our lambda calculus — and propositions. Therefore it is

important that we provide some way for a computer to check the type and validity of a term. We can check term validity through a concept known as well typed terms. The type of a term is checked through a process known as type checking; based on the way a term is structured we can check whether a given type makes sense. For example, if we have the lambda term $\lambda x.\texttt{fst}\, x$ and we say it has type $\mathbb{N} \times \mathbb{Q} \to \mathbb{N}$, then we can check this type is correct. Indeed x must be a pair, because we apply `fst` to it and since we project the first element of the pair, we must return the first type in the pair. Therefore our given type is considered valid, because we take in a pair type and return the first type in the pair. Through the Curry-Howard correspondence, we give meaning to types in the following way:

- A pair of types corresponds to logical *and* of the two propositions.
- A function from type T to S, is the statement that T implies S.

The type of the term is thought to give the statement of a proposition and the body of the term to give the proof of the proposition. We can introduce more types and terms to make our type system more rich, allowing us to describe more propositions. This includes concepts such as existential quantification and the logical *or* of two propositions.

The last step is to extend the Curry-Howard correspondence to the Curry-Howard-Lambek correspondence (Lambek, 1986) which adds links from both type theory and propositions to Cartesian closed categories. The link between type theory and category theory is shown by the fact that the category of the simply typed λ-calculus is in fact a Cartesian closed category.

References

Asperti, Andrea (1991). *Categories, types, and structures : an introduction to category theory for the working computer scientist / Andrea Asperti, Giuseppe Longo.* eng. Foundations of computing series. Cambridge, Mass. ; London: MIT. ISBN: 0262011255.

Awodey, Steve (2010). *Category theory / Steve Awodey.* eng. ISBN: 9780191612558.

Crole, Roy L (1993). *Categories for types / Roy L. Crole.* eng. Cambridge mathematical textbooks. Cambridge: Cambridge University Press. ISBN: 0521450926.

Gonthier, Georges (2005). "A computer-checked proof of the Four Colour Theorem". In: URL: https://api.semanticscholar.org/CorpusID:18529792.

Lambek, J (1986). *Introduction to higher order categorical logic / J. Lambek, P.J. Scott.* eng. Cambridge studies in advanced mathematics ; 7. Cambridge: Cambridge University Press. ISBN: 0521246652.

Mac Lane, Saunders (1971). *Categories for the working mathematician.* eng. Graduate texts in mathematics 5. New York: Springer-Verlag. ISBN: 0387900365.

Walters, Richard F (1991). *Categories and computer science / R. F. C. Walters.* eng. Cambridge computer science texts ; 28. Cambridge: Cambridge University Press. ISBN: 0521419972.