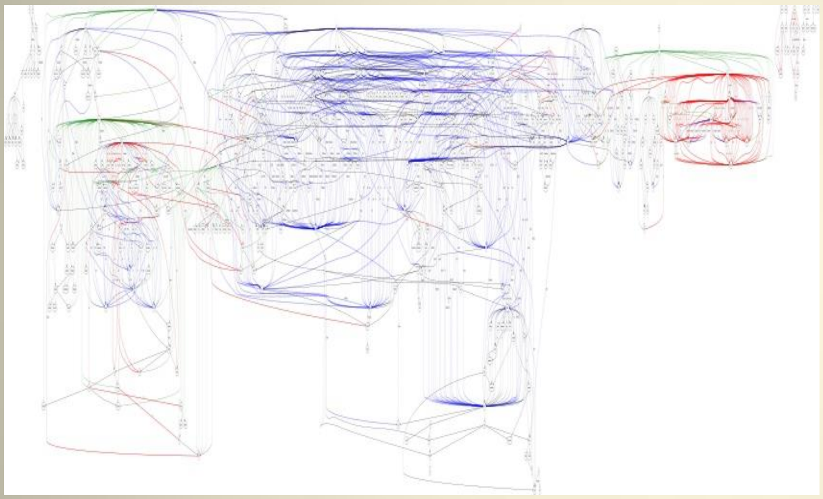
Category Theory and the Curry-Howard-Lambek Correspondence



Mark Hamilton, Yale University

Senior Seminar in Mathematics
Lecture Notes

2/22/2016

Contents

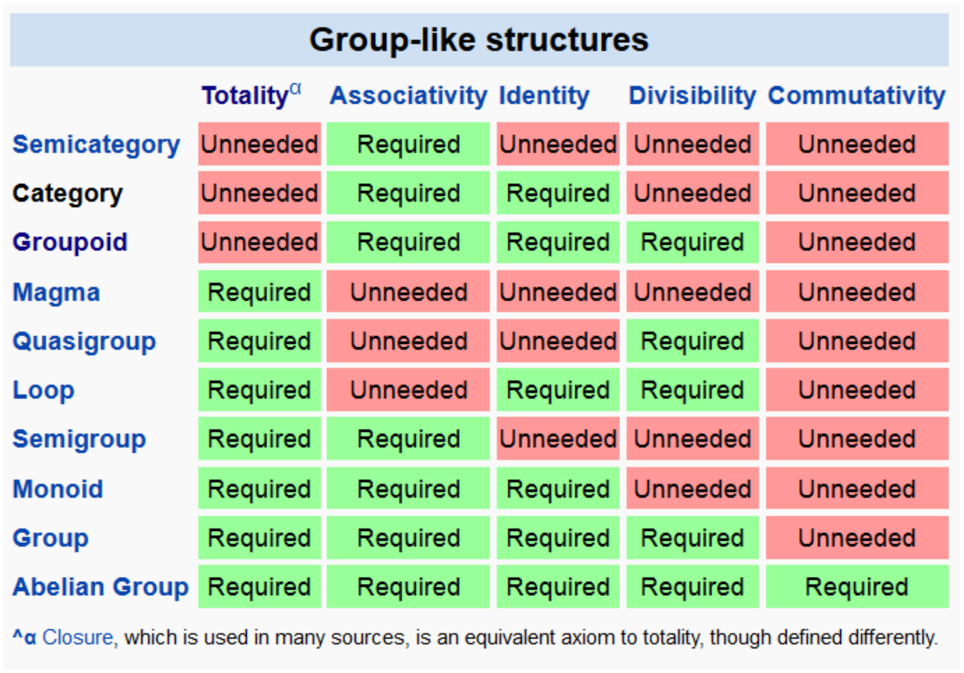
Section 1: Introduction to Categorical Notions

What is Category Theory?

Fundamentally, category theory is language or algebra of maps. This phenomenon of a language of symbols capturing an idea also underlies groups as the abstraction / language / algebra of symmetry. Groups can even be regarded as special types of categories, and as a property held by a category and proves to be very powerful in understanding other algebraic objects. Furthermore, Category theory is a powerful foundation of mathematics and allows one to unify many disparate mathematical notions elegantly. For example, category theory allows the unifications of:

- Proofs, programming languages, logic, and algebraic operations through the categorical notions of Products and exponentials
- Sub-objects, preimages, algebraic varieties through the categorical notions of Equalizers and pullbacks.
- And many more

To see how categories can be viewed as generalized groups, monoids, and groupoids, it is useful to see how their axioms compare as in figure 1.



Group-like structures

	Totality[α]	Associativity	Identity	Divisibility	Commutativity
Semicategory	Unneeded	Required	Unneeded	Unneeded	Unneeded
Category	Unneeded	Required	Required	Unneeded	Unneeded
Groupoid	Unneeded	Required	Required	Required	Unneeded
Magma	Required	Unneeded	Unneeded	Unneeded	Unneeded
Quasigroup	Required	Unneeded	Unneeded	Required	Unneeded
Loop	Required	Unneeded	Required	Required	Unneeded
Semigroup	Required	Required	Unneeded	Unneeded	Unneeded
Monoid	Required	Required	Required	Unneeded	Unneeded
Group	Required	Required	Required	Required	Unneeded
Abelian Group	Required	Required	Required	Required	Required

^α Closure, which is used in many sources, is an equivalent axiom to totality, though defined differently.

Figure 1: Table comparing the axioms of group-like mathematical objects

Def: Category

A category is a directed graph with a bit more structure. More formally, a category has:

- Objects:
  - X, Y, Z
- Arrows:
  - f, g, h
- Domains and Codomains (Ranges):
  - $f: X \to Y, \quad dom(f) = X, \quad cod(f) = y$
- Composites:
  - $if\ f: X \to Y,\ \ g: Y \to Z,\ then\ \exists\ g \circ f : X \to Z$
- Identities:
  - For each object X there is an arrow
  - $1_X: X \to X$

The Axioms of Category Theory:

- Associativity
  - $\forall\ f: A \to B,\ \ g: B \to C,\ \ h: C \to D$
  - $h \circ (g \circ f) = (h \circ g) \circ f$
- Units
  - $\forall\ f: A \to B$
  - $f\ \circ\ 1_A = 1_B \circ\ f$

Example: Category of Sets

This is the most familiar example of a category that one should often use for intuition.

- Objects: Sets
  - $A, B, C,\ \mathbb{R}, \mathbb{C}, \ldots.$
- Arrows: Maps/Functions
  - $f: \mathbb{R} \to \mathbb{C}, \quad f(x) = 3x + i$
  - $g: \mathbb{C} \to \mathbb{R}^2, \quad g(a) = (Re(a), Im(a))$
- Transitivity and Associativity: Composition of maps
  - $g \circ f: \mathbb{R} \to \mathbb{R}^2, \quad g \circ f(x) = (3x, 1)$
- Identities: Identity map
  - $1_{\mathbb{R}}: \mathbb{R} \to \mathbb{R}, \qquad 1_{\mathbb{R}}(x) = x$

Example: Category of Proofs

- Objects: Mathematical formulas, aka any statement of mathematics
  - $\varphi, \psi,\ \ldots$
    - $1 + 1 = 2$
    - $\forall H < G:\ \ \frac{|G|}{|H|}$

- Morphisms: Proofs of Implication
    - $proofs\ of\ "\varphi \Rightarrow \psi"$
    - $p\colon \varphi \Rightarrow \psi$
    - More formally, proofs are trees whose nodes are sentences of mathematics and whose connections arise from labeled proof rules as in the example in red below. In this example, “AI” stands for “And introduction” and “MP” stands for “Modus Ponens”. Both of these are standard logical deduction rules.

Ex:

$$\cfrac{\cfrac{p \qquad\qquad q}{p\ and\ q}\,AI \qquad\qquad (p\ and\ q)\ \Rightarrow r}{r}\,MP$$

- Composition comes out chaining proofs together and the transitivity of ⇒

Note that one can prove theorems of math in many ways, so there are many possible arrows from φ to ψ. The structure of this category is incredibly rich, and has many connections to other areas of math.

More Examples:

- **Group**s and group homomorphisms,
- **Vect**or spaces and linear mappings,
    - In physics they are Feynman Diagrams!
- **Graph**s and graph homomorphisms,
- Real numbers $\mathbb{R}$ and continuous functions $\mathbb{R} \to \mathbb{R}$,
- Open subsets $U \subseteq \mathbb{R}$ and continuous functions $f : U \to V \subseteq \mathbb{R}$ defined on them,
- **Top**ological spaces and continuous mappings,
- Differentiable manifolds and smooth mappings,
- the natural numbers $\mathbb{N}$ and all recursive functions $\mathbb{N} \to \mathbb{N}$,
- **Pos**ets and monotone functions
- (Small) **Cat**egories and functors

Commutative Diagrams

Commutative diagrams are pictorial representations of equalities that hold within a category. One reads a commutative diagram by equating paths that start and end at the same object. In the example diagram no matter which path you follow (either f then g or $f \circ g$) you will get the same answer. This diagram succinctly captures the semantics of composition and identity function axioms.

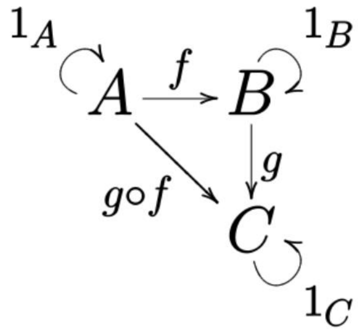


Figure 2 Commutative Diagram of composition and identities

Def: Functors

We have seen that most algebraic objects also come with related morphisms or structure preserving maps. The functors is the morphism or structure preserving map of categories.

- Given two categories C, D a functor $F\colon C \to D$ is a mapping of objects to objects and arrows to arrows such that:
  - $F(f\colon X \to Y) = F(f)\colon F(X) \to F(Y)$
  - $F(1_X) = 1_{F(X)}$
  - $F(g \circ f) = F(g) \circ F(f)$

One can imagine a functor as an image of one category in another as shown in figure 3 below.

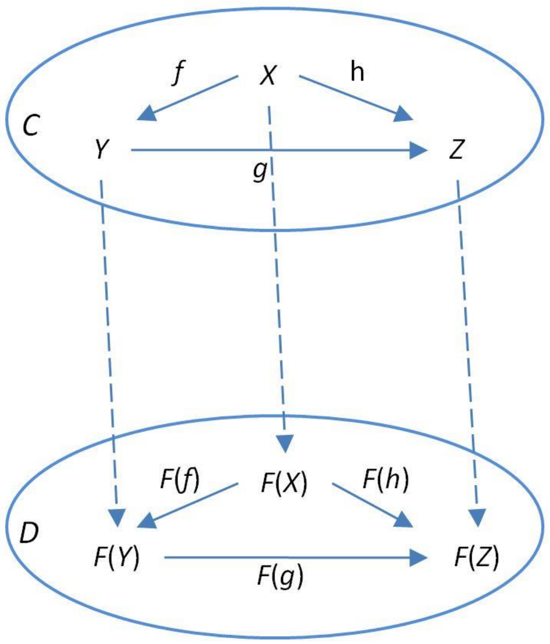


Figure 3: Pictoral representation of a functor

Examples: Functors

Functors appear almost everywhere in modern mathematics. They are the links that help move information around the web of mathematical theories.

- Taking a group action is functor from **Group** to **Set**
- Representation theory is the study of functors from **Group** to **Vect$_k$**
- Associating a Fundamental group to a topological space is a functor from **Top*** to **Group**
- (Roughly) Galois Theory is a functor from an automorphism group to Q extensions (Can be further generalized to a "Galois connection")

Note: Top* is the category of pointed topological spaces or topological spaces with a distinguished point. (The starting point of the homotopy loop)

Remark: Duality

Let us take the time now to note that every category has a corresponding opposite category formed by reversing the arrows. This is called the "duality principle" of category theory and arises because the axioms and symbols of category theory are self-dual. Here, transforming a categorical statement to its dual involves flipping the direction of the arrows, and switching out domains for co-domains. This duality means is that every theorem and construction of category theory counts double!

Section 2: Direct Products

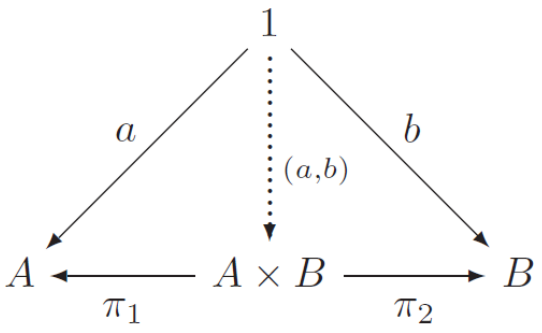
Now that we have seen some of the basic categorical language, let us think about possible constructions and objects within the language of category theory. The first structure we will consider is that of a direct product. This astonishingly general construction arises in most categories of interest. It generalizes many other notions of products in algebraic theories, as we shall see shortly.

Def: Direct Products in Set

To give some helpful intuition let us consider the familiar case of direct products in sets.

- Consider the direct product of sets
    - $A \times B = \{(a, b) | a \in A, b \in B\}$
- We can "project" onto the coordinates
    - $\pi_1 : A \times B \to A, \quad \pi_1(a, b) = a$
    - $\pi_2 : A \times B \to B, \quad \pi_b(a, b) = b$
    - $c = \left(\pi_1(c), \pi_2(c)\right) \;\; if \;\; c \in A \times B$

Note that if any set has maps to the parts of the product, there is a unique map into the product making the diagram "commute". The existence of this map ensures that A x B is large enough to contain all of the proper elements. The uniqueness of this map prevents the product from having excess clutter. For example if A x B was replaced with A x B x C in Figure 4 below, there could be |C| potential maps into A x B x C by mapping to an arbitrary element of C in the 3rd coordinate. Also, note that Figure 4 below shows this mapping for the set with one element, but this property is true of any set that maps into both A and B.



$$\begin{tikzcd} & 1 \arrow[dl, "a"'] \arrow[d, dotted, "{(a,b)}"] \arrow[dr, "b"] & \\ A & A \times B \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & B \end{tikzcd}$$

Figure 4: A diagram in Sets showing how the set with one element maps into a direct product of sets, as well as the sets themselves.

Def: Direct Products in any Category

Armed with the intuition from considering the category of sets, we can generalize this notion to an arbitrary category.

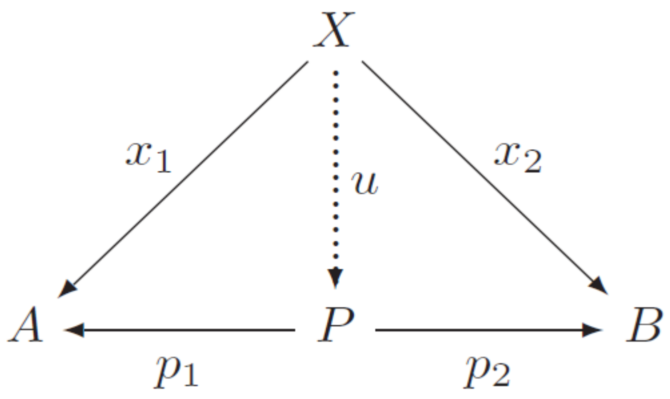
- In any category **C**, a *product diagram* for the objects *A* and *B* consists of an object *P* and arrows

$$A \overset{p_1}{\leftarrow} P \overset{p_2}{\rightarrow} B$$

- Such that given another object X with arrows

$$A \overset{x_1}{\leftarrow} X \overset{x_2}{\rightarrow} B$$

- There exists a unique arrow $u: X \to P$ making the following diagram commute



$$\begin{tikzcd} & X \arrow[dl, "x_1"'] \arrow[d, dotted, "u"] \arrow[dr, "x_2"] & \\ A & P \arrow[l, "p_1"] \arrow[r, "p_2"'] & B \end{tikzcd}$$

Please note that products do not necessarily exist in every category. The existence of all direct products in a category is a very special property and tells you something nontrivial about the structure of the category. We will now show that direct products of the same objects are isomorphic. First, let us define an isomorphism in a category.

Def: Isomorphism in a Category

In any category **C**, an arrow $f: A \to B$ is called an *isomorphism*, if there is an arrow $g: B \to A$ in **C** such that

$$g \circ f = 1_A \; and \; f \circ g = 1_B$$

Prop: Products are Isomorphic

To prove that all products are isomorphic, consider any two direct products P and Q with projections labelled by p and q. We would like to show that these products have isomorphisms (aka invertible morphisms) between them as the definition for isomorphism calls for.

$$\begin{tikzcd} A & P \arrow[l, "p_1"'] \arrow[r, "p_2"] & B \end{tikzcd}$$

$$\begin{tikzcd} A & Q \arrow[l, "q_1"'] \arrow[r, "q_2"] & B \end{tikzcd}$$

Through their universal properties, one can form the following commutative square on the right. From the diagram, we can begin to read off equations by following separate paths A few key equations are shown on the left. j and i will be shown to be isomorphisms.

$$p_1 \circ j \circ i = p_1$$
$$p_1 \circ 1_p = p_1$$
$$p_2 \circ j \circ i = p_2$$
$$p_2 \circ 1_p = p_2$$

$$\begin{tikzcd} & P \arrow[dl, "p_1"'] \arrow[dr, "p_2"] \arrow[d, dotted, "i"] & \\ A & Q \arrow[l, "q_1"] \arrow[r, "q_2"'] \arrow[d, dotted, "j"] & B \\ & P \arrow[ul, "p_1"] \arrow[ur, "p_2"'] & \end{tikzcd}$$

Now, by the uniqueness of the projections into the direct products we can conclude $j \circ i = 1_p$. Likewise $i \circ j = 1_q$ which satisfies the definition of an isomorphism. ■

Section 3: Category Theory as a Foundation

One of the large drivers of category theory initially was the fact that it can serve as a foundation of mathematics like Set theory. Foundational theories serve as a standard way of talking about all known mathematics in a unified language. As an example of this, one can construct the natural numbers just as one can in set theory:

- 0 = (category with no objects)
- 1 = *
- $2 = * \to \#$
- 3 = $$\begin{tikzcd} * \arrow[r] \arrow[dr] & \# \arrow[d] \\ & \& \end{tikzcd}$$
- ……..

As a philosophical note, because each object has an identity arrow (not shown) we can theoretically ignore the objects! Then mathematics becomes completely dependent on the morphisms. In category theory there are no objects, everything is relative.

Remark about Foundational Math

The modern study of foundations has shown that there are a HUGE number of possible foundational languages. However, most math is "Foundationally Independent" which means that it does not depend on the exact choice of foundation. For example, one can define the natural numbers in either set theory or category theory, it doesn't really matter. When considering a good language of mathematics, a candidate should:

- be useful on many levels of generality
- unify many distinct concepts cleanly
- describe many concepts simply
- have good computational properties
    - set theory does not, category theory has Haskell!
- be used by many
    - "Those who don't believe it are on their way out anyway" – Rodger Howe

Note that choosing a foundational language is incredibly similar to choosing a programming language. You would laugh if someone refused to learn java because c already was Turing complete. One needs to choose the language for the task and be flexible.

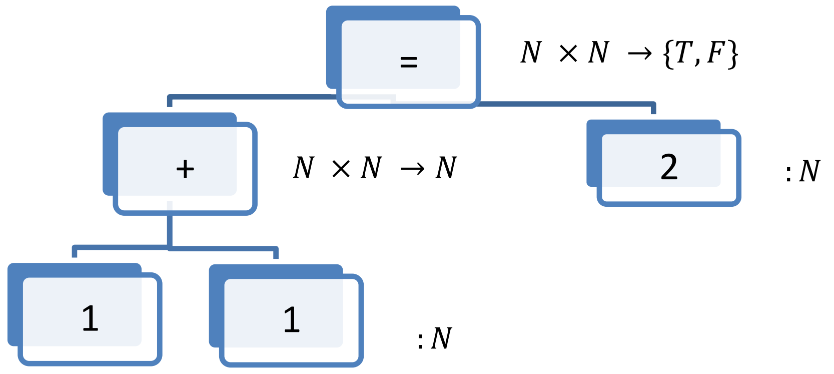
Section 4: Unifying Math and Computation

Other than using category theory as a foundation of mathematics, category theory can give us insight into how to construct mathematical and computational languages. It gives elegant and "universal" characterizations of logic itself. To see this let us examine some mathematical formulas from distinct areas in greater depth. We will see that most mathematical equations can be seen as syntax trees denoting the composition of maps on base terms, the same language used to describe functional programming languages. Let us first consider the familiar equation in arithmetic:

$$1 + 1 = 2$$

-Contains the symbols: {1, 2,+,=}

-After parsing:

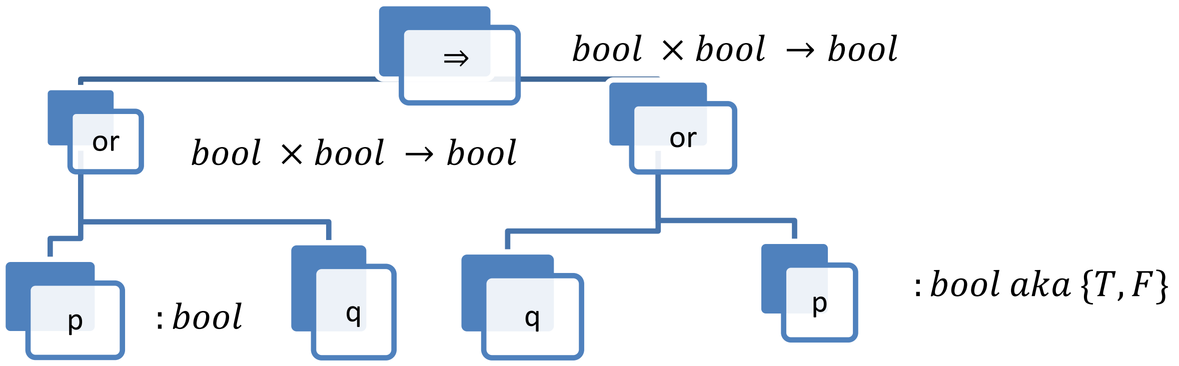


Where ":" denotes the type of the node, "N" being the type of natural numbers. Now consider a theorem of propositional logic:

$$p \; or \; q \Rightarrow q \; or \; p$$

-Symbols: {p, q, or, ⇒}

-After parsing:


⇒
$bool \times bool \rightarrow bool$
or
$bool \times bool \rightarrow bool$
or
p
$: bool$
q
q
p
$: bool \; aka \; \{T, F\}$

Note that these structures look incredibly similar despite being from very different types of mathematics. This points to a language of mappings to understand equations.

The Curry-Howard-Lambek Correspondence

This correspondence establishes that proving, coding, and category theory are essentially the same thing! The backbone of the correspondence is the idea that proofs are algorithms/functions/mappings. To see this consider a few examples shown in the table below:

Proofs of theorem:	**Algorithms/Mappings/Functions**
$p \Rightarrow q$	A machine/mapping that takes in proofs of p, returns proofs of q
not p	A machine that takes in proofs of p, returns contradictions
$\forall x\, P(x)$	A machine that takes in objects/number/set/whatever and spits out a proof of the formula P with the object subbed in

One might note that these definitions are recursive; the definitions require some starting proofs to work with. These starting proofs are called axioms, the foundational building blocks of all mathematical theories. It is worth noting that axioms are like the standard library in coding. One can compose them and fit them together to build bigger and more complicated theorems (resp. pieces of software). This correspondence does not stop at the above table, almost every mathematical notion has a corresponding notion in programming as shown in the table below:

Logic	**Programming**
universal quantification : ∀	generalized product type (Π type)
existential quantification: ∃	generalized sum type (Σ type)
implication: ⇒	function type: add = function(a, b) {a + b}
Conjunction: and	product type: (Int, Int)
Disjunction: or	sum type: if x=1: return True else: return 5
true formula: T	unit type: unit
false formula: F	bottom type: void
modus ponens: p ⇒q , p thus q	Function application
…..	……..

Algebraic Data Types (F-Algebras)

To understand how this correspondence works formally, we must understand how to think about computation algebraically. The notation will hopefully be familiar from other areas of mathematics where data is represented as "$Term$: $Type$" with an example being $25\colon\ Integer$. Algebraic data types are a rigorous algebraic formulation of how computers interpret data and functions, and are used in functional programming languages such as Haskell. One starts with a collection of datatypes and can then combine them with "tuples" and "options".

Tuples are a way of taking pairs of objects, or collections of objects of different types. Some examples include:

- $(T,\ F)\colon\ \ Bool\ \times\ Bool$
- $(0,\ 25)\colon\ \ Int \times Int$
- $(\text{"}hello\text{"},\ \text{"}world\text{"})\colon\ \ String \times String$

Options are a way of specifying a datatype of mixed type. Where in certain situations it might behave like one datatype and in other situations it will behave like another. Some examples include

- $if\ x = 1\colon\ return\, T\ else\colon\ return\, 5$
    - The above has type $Option(Bool,\ Int)$ or $Bool\ +\ Int$

Combinatorial Interpretation of Operations

Surprisingly, these type theoretical operators have combinatorial interpretations. These operators tell one exactly how many terms a composite type might have. Under this interpretation, tuples are like products:

- Bool type has 2 terms: $\{T,\ F\}$
- Bool x Bool has $2 \times 2 = 4$ terms:
    - $\{(T,F),(F,T),(T,T),(F,F)\}$

and options are like sums:

- $Option(Bool,\ Int)$ can either be a boolean with 2 terms or an integer 2^{16} terms
- So $Option(Bool,\ Int)$ has $2 + 2^{16}$ terms

Algebraic Data Types: Generating Data Types

To generate the set of familiar datatypes used in computation, it is sufficient to start with one type called the "unit" type which has one "state" or "term" : 1. Next one can form the Boolean and integer datatypes with an options and tuples:

- $Option(unit, unit) = 1 + 1 = 2 = Bool$
- $Int = 2^{16} =\ 2 \times 2 \times 2 \ldots \times 2$

One should note that this tuple of Booleans can be interpreted as the binary number interpretation of any integer. Furthermore, mappings between types arise out of "exponentiation" of datatypes formed by successive products:

- $Int^2 = (Int, Int)$

Int^2 can be thought of as counting the number of functions from $\{T, F\}$ to $\{0,1,2 \ldots 2^{16} - 1\}$. Just like $\mathbb{R} = \mathbb{R}^1$, $\mathbb{R}^2, \mathbb{R}^{\mathbb{N}}$ etc. Now that we have a notion of products, sums, and exponentials, one can define recursive datatypes (such as a linked list) which lead to Taylor series, derivatives, division, and subtraction. This continues to blossom into the intriguing topic of "Combinatorial Species"

Definition: Lambda Calculus

Lambda calculus can be thought of as the language that describes computation. Typed lambda calculus has two levels corresponding to types and terms. The type level is governed by algebraic datatypes, and the term level is governed by specific term combination and formation functions in the definition of the lambda calculus. In the lambda calculus, terms are represented by strings of symbols that are manipulated by formation and substitution rules. This is designed to model how mathematicians manipulate symbols from one line of reasoning to the next. Computation can be thought of as the result when terms of compatible types are composed. For example the term "2":N, can be fed into the term "λx.x+1" : N→ N to yield "3":N. More formally, Lambda calculus has the following objects and symbols:

- Types:
  - Generated from base types using tuples, options, and functions….
    - $A,\ B,\ A \times B\ ,\ A \to B$
- Terms:
  - Variables $x{:}\,A$
  - Constants $a{:}\,A$
  - Tuples $\langle a, b\rangle{:}\,A \times B \quad if\ a{:}\,A,\ b{:}\,B$
  - Projection terms $fst(c){:}\,A \quad if\ c{:}\,A \times B$
  - $snd(c){:}\,\mathrm{B} \quad \text{if c}{:}\,\mathrm{A} \times B$
  - Application $ca{:}\,B \quad if\ c{:}\,A \to B\ ,\ a{:}\,A$
  - Function Def $\lambda x.b{:}\,A \to B \quad if\ x{:}\,A\ ,\ b{:}\,B$

And its semantics or meaning is defined through the axioms

- $fst(a,\ b) = a$
- $snd(a,\ b) = b$
- $\langle fst(c),\ snd(c)\rangle = c$
  - Note that we use angled brackets to distinguish from products in the category of lambda calculus (referenced in the next section)
- $(\lambda x.b)a = b[a/x]$ Function Application

    - Ex: $b = 1 + x$, $(\lambda x. b)a = (\lambda x. (1 + x))a = 1 + a$
    - one writes $\lambda y. \lambda x. x + 2y$ for the function x, $y \longmapsto x + 2y$
- $(\lambda x. c)x = c\ (no\ x\ in\ c)$ Function application 2
    - Ex: $c = 1 + y$, $(\lambda x. c)x = 1 + y = c$

Furthermore, two terms are "$\beta\eta - equivalent$" if you can "rename" the bound variables to make them equal:

- $\lambda x. b = \lambda y. b[y/x]$ $(no\ y\ in\ b)$

$\beta\eta - equivalence$ Formalizes the idea that renaming the variables in a piece of code does not change the meaning of the algorithm you coded.

Prop: Lambda Calculus is a Category

Given this definition of lambda calculus, one can show that it is an example of a category with:

- Objects: The Types
    - $A, B, A \times B$ ....
- Arrows: (Closed) Function Terms
    - $c: A \to B$
    - Terms that are $\beta\eta - equivalent$ are identified
- Identities:
    - $1_A = \lambda x. x : A \to A$
- Composition:
    - $c \circ b = \lambda x. c(bx): A \to C$
- Associativity: Easily proven from axioms

Note that in a closed term all variables are bound by a lambda. Ex $\lambda y. x + y$ is open and $\lambda y. \lambda x. x + y$ is closed.

Theorem: The category of λ-calculus has products

Given types A and B, the tuple type $A \times B$ is the product type in the category of lambda calculus with projections:

- $p_1 = \lambda z. fst(z): A \times B \to A$
- $p_2 = \lambda z. snd(z): A \times B \to B$

If there is some other type X with function terms a and b let $(a, b) = \lambda x. \langle ax, bx \rangle$

$$\begin{tikzcd}
 & X \arrow[dl, "a"'] \arrow[d, dotted, "{(a,b)}"] \arrow[dr, "b"] & \\
A & A \times B \arrow[l, "p_1"] \arrow[r, "p_2"'] & B
\end{tikzcd}$$

Proof Plan:

In order to prove the theorem we need to consult the definition of direct products in a category and show that:

- $p_1 \circ (a,\ b)\ = a$
  - Existence part 1
- $p_2 \circ (a,\ b) =\ b$
  - Existence part 2
- $if\ c : X \to A \times B\ and\ if\ p_1 \circ c\ = a$ and $p_2 \circ c\ = b\ then\ c = (a, b)$
  - The uniqueness part

Existence: $p_1 \circ (a,\ b)\ = a$

In all proofs to follow reasoning for the step will be given as a sub-bullet.

- $p_1 \circ (a,\ b) = \lambda x(p_1((\lambda y. \langle ay,\ by\rangle)x))$
  - From expanding def of $\circ$ and $(a,\ b)$
- $= \lambda x(p_1\langle ax,\ bx\rangle)$
  - From application of inner $\lambda y. \langle ay,\ by\rangle)x$
- $= \lambda x(ax)$
  - From projection p_1
- = a
  - a does not contain x

Note that the proof is identical for p_2

Uniqueness: $c = (a, b)$

- $(a,\ b)\ =\ \lambda x. \langle ax,\ bx\rangle$
  - Definition of (a,b)
- $=\ \lambda x. \langle (p_1 \circ c)x, (p_2 \circ c)x\rangle$
  - $p_1 \circ c\ = a$ in the assumption
- $=\ \lambda x. \left\langle \left(\lambda y(p_1(cy))\right) x, (\lambda y(p_2(cy)))x \right\rangle$
  - Definition of $\circ$
- $=\ \lambda x. \left\langle \left(\lambda y((\lambda z. fst(z))(cy))\right) x, (\lambda y\left(((\lambda z. snd(z))(cy))\right))x \right\rangle$
  - Definition of p_1
- $=\ \lambda x. \langle \lambda y(fst(cy))x, \lambda y(snd(cy))x\rangle$
  - Inner application of λz
- $=\ \lambda x. \langle fst(cx), snd(cx)\rangle$
  - Application of λy
- $=\ \lambda x. cx$

    - Axiom characterizing ordered pair
- $= c$
    - No x in c

Theorem: The category of Lambda Calculus is Isomorphic to the Category of Proofs

One might have noticed that every term formation rule of lambda calculus also doubles as a rule of logical inference in mathematics. Where product formation corresponds to “and introduction” and function application is “modus ponens” to name a few. More generally, the category of proofs defined earlier, is isomorphic to the category of lambda calculus as categories. Where an isomorphism of categories is an isomorphism in the category of finite categories. Equivalently we can find a pair of mutually invertible functors that can take you from the category of proofs to the category of lambda calculus.

Proof sketch: Create a pair of functors which:

- Map term formation rules to rules of logical inference
- Map base terms to axioms
- Do the diagram chase to show isomorphism holds

Under this isomorphism, we can think of types representing theorems, and terms representing their proof. An example of this is shown in figure 5. Furthermore, we can show “and” is a direct product in the category of proofs using this isomorphism!

$$\dfrac{\dfrac{\dfrac{[x : A] \qquad [y : B]}{\langle x, y\rangle : A \times B}}{\lambda y.\langle x, y\rangle : B \to (A \times B)}}{\lambda x \lambda y.\langle x, y\rangle : A \to (B \to (A \times B))} \qquad \dfrac{\dfrac{\dfrac{A \qquad\qquad B}{A \text{ and } B}}{B \Longrightarrow (A \text{ and } B)}}{A \Longrightarrow (B \Longrightarrow (A \text{ and } B))}$$

Figure 5: An example of the correspondence between lambda calculus (left) and proofs in mathematics (right)

What about “Or”?

Under this interpretation, “Or” corresponds to adding an option type to lambda calculus. In category theory, this corresponds to the dual notion of the direct product, called the direct sum. The direct sum is just the direct product with the arrows reversed. By reversing the arrows, the corresponding logical formation rules are flipped, which yields the logical rules that characterize “or”. This is shown in figures 6 and 7 below.

$$\frac{A \text{ and } B}{A}, \quad \frac{A \text{ and } B}{B}$$

$$\begin{tikzcd}
 & Z \arrow[dl, "z_1"'] \arrow[d, dotted, "u"] \arrow[dr, "z_2"] & \\
A & P \arrow[l, "p_1"] \arrow[r, "p_2"'] & B
\end{tikzcd}$$

Figure 7: Logical rules describing and (top) and categorical diagram describing and (bottom)

$$\frac{B}{A \text{ or } B}, \quad \frac{A}{A \text{ or } B}$$

$$\begin{tikzcd}
 & Z & \\
A \arrow[ur, "z_1"] \arrow[r, "q_1"'] & Q \arrow[u, dotted, "u"'] & B \arrow[ul, "z_2"'] \arrow[l, "q_2"]
\end{tikzcd}$$

Figure 6: Logical rules describing or (top) and categorical diagram describing or (bottom)

Thinking About this More Generally

More generally, the Curry-Howard-Lambek isomorphism is a functor from The category of lambda calculus to the category of proofs. This is called "Categorical Semantics" and it occurs in **every** language of math and computation, making it a valuable tool to understand new mathematical theories and new programming languages. Furthermore, it is incredibly useful when building new languages, or finding structure in languages. More specifically, one can use category theory to perform "theory engineering". This roughly refers to the idea that mathematical theories can be built up from simpler mathematical theories. Just as numbers can be factored into primes, theories themselves can be "refactored" into modular components. One familiar example of this is that a ring can be seen as a combination of a abelian group (+) and a semi-group (*) with axioms which mediate between the two.

Why does it matter?

The study of this kind of mathematics matters because the CHL correspondence yields the proper data-structure for storing proofs. This is very important for large formal verification efforts like the Feit-Thompson, Robbins Conjecture, Four color theorem, and Kepler conjecture where many mathematicians need to share and check proofs automatically. Computerizing mathematics allows machines to process and analyze more math than any one person could. Furthermore, categorical constructions are different directions to pursue in the effort to create a richer mathematical theory. Their incredible level of generality helps to guide mathematics and direct mathematicians to “universal structure” in the objects they consider.

References

- Awodey, Steve. *Category theory*. Oxford University Press, 2010.
- Mac Lane, Saunders. *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media, 1978.
- Kohlhase, Michael, Till Mossakowski, and Florian Rabe. "Latin: Logic atlas and integrator." *URL: http://latin. omdoc. org (v isited on 09/15/2010)* (2012).
- Rabe, Florian. “Integrated Lecture Notes on Logic” (2104)
- Bergeron, François, Gilbert Labelle, and Pierre Leroux. *Combinatorial species and tree-like structures*. Vol. 67. Cambridge University Press, 1998.
- Jay, Barry. "Algebraic data types." *Pattern Calculus*. Springer Berlin Heidelberg, 2009. 149-160.