Types are Brunerie globular weak ω-groupoids

Younesse Kaddar

Summer Research Internship

Supervised by:

Thorsten Altenkirch

Paolo Capriotti

Nicolai Kraus

University of Nottingham

Functional Programming Laboratory

August 2017

Contents

1. Synthesis

General context

Martin Löf's Type theory (MLTT) is a framework which can serve as a foundation for mathematics and can be seen at the same time as a programming language. In this respect, special attention is paid to its computable properties and its connections with constructive mathematics. Homotopy Type Theory is a new flavour of MLTT which has burgeoned over the past few years, in which types are regarded as topological spaces, and identity proofs as (1-)paths between points of these spaces. In addition, identifications between proofs are seen as paths between paths (homotopies, or 2-paths), and identifications between 2-paths as 3-paths, and so on. This gives to types a structure of what are called *ω-groupoids*.

Numerous definitions of ω-groupoids have already been given by several researchers, among whom Alexander Grothendieck, Georges Maltsiniotis, Michael Batanin, or Tom Leinster, to name a few. But it is Guillaume Brunerie's definition that will be retained here, insofar as it can be formalized within type theory, by its very type theoretic nature.

Problem statement

The problem was to show that each type forms an globular weak ω-groupoid in the sense of Brunerie. Brunerie ω-groupoids are all-important, in that they enable us to express everything within type theory, which is a key point, since it may pave the way for getting rid of a non-constructive axiom of Homotopy Type Theory: the univalence axiom, posed by Vladimir Voevodsky.

To carry out the practical formalization, the dependently typed programming language Agda was resorted to, and in particular the "Type Theory in Type Theory" framework developed by Thorsten Altenkirch and Ambrus Kaposi, which features a useful mathematical structure used to model type theory: categories with families. This approach has not been adopted before because this framework is very recent: the features that are essential for the formalization of Brunerie ω-groupoids are still under development (by Thorsten Altenkirch and Ambrus Kaposi). In particular, the use and formalization of categories with families - down to the very definition of Brunerie weak ω-groupoids - is game-changer which looks very promising.

Contribution

The problem was solved, on paper, by resorting to categories with families (to model type theories) and seeing Brunerie ω-groupoids, associated with a given type, as certain kinds of morphisms that emerge as a composition of other morphisms, based on Paolo Capriotti's shrewd observation. A technical lemma in the the proof has been inspired by Peter LeFanu Lumsdaine's thesis[Lum10] (section 3.2).

When it comes to formalization, Brunerie's type theory has been defined in Agda, but the proof has not been completed yet.

Arguments in favour of what has been done

One can still question the very definition of Brunerie ω-groupoids as morphisms, but it is *natural*, in that it mimics the definition of a model of an algebra in Lawvere theories. It is also *intuitive*, to the extent that ω-groupoids are purposely patterned on Brunerie's typing rules. It should be noted that the result ("types are Brunerie globular weak ω-groupoids") holds for any type theory with *intensional* equality.

Results and prospects

This result is but a first step towards eliminating the univalence axiom, if it can successfully be formalized within type theory (it is in that spirit that it was addressed). If so, the next step is to construct the ω-groupoid of types seen as ω-groupoids, within which equivalences between types are to be constructively witnessed by actual morphisms (inhabitants of the equality types).

Acknowledgments

I heartily thank Prof. Altenkirch, Nicolai Kraus and Paolo Capriotti for giving me the opportunity to work on this summer internship project at the University of Nottingham. It was a fantastic foray into research!

I am also very grateful to Jean Goubault-Larrecq for making it possible and for his thoughtful advice and encouragement.

I must express my profound gratitude to Prof. Altenkirch for his generosity, kindness, nice support and guidance.

I cannot thank Paolo enough for his motivation, friendliness, and for everything he has taught me: he is and will always remain an inspiration for me.

I am very thankful to Ambrus Kaposi for his advice and assistance with Agda.

Finally, a special thanks goes to all at the Functional Programming Lab for the warm and friendly welcome and for all the fun we've had: I've had a wonderful time!

2. Introduction

The main aim of this internship is to show that

> Type Theory forms a globular weak ω-groupoid in the sense of Guillaume Brunerie

with a particular emphasis on formalization aspects in the dependently typed programming language *Agda*.

We shall now explain what all this means, and what is the underlying motivation behind such a problem.

Motivation

What is wrong with set theory

Modern mathematics is usually assumed to be based on **set theory**: all mathematical objects boil down to sets, and virtually all mathematical statements can be derived from the axioms of set theory, with resort to the rules of formal logic.

But as Bourbaki stated in [Bou07]:

> It may happen at some future date that mathematicians will agree to use models of reasoning which cannot be formalized in the current language described here [*that is, set theory*]: it would then be necessary, if not to change the language completely, at least to enlarge its rules of syntax.

Not that this "future date" has already come, but encoding rich mathematical objects set theoretically can prove to be very far-fetched and nettlesome to some extent, due to the poorness of the very notion of *set* (which is nothing more than a simple "bag" containing a certain amount of nested "bags", the most deeply nested being the empty set $\varnothing$); let alone reasoning about isomorphic or equivalent objects (which might be defined completely in a totally different manner, in terms of sets).

■ **Example 2.1 — Real numbers.** Just think of real numbers: does your mathematical intuition of what they really are match their rigorous definitions as equivalence classes of Cauchy sequences of rational numbers or as Dedekind cuts?
And is it even intuitive that these definitions are *equivalent*? Just for the record, the following sets both represent the *same* mathematical object, namely $\sqrt{2}$:

- The *Dedekind cut*:

$$\Big(\{x \in \mathbb{Q} : x^2 < 2 \text{ or } x < 0\}, \{y \in \mathbb{Q} : y^2 > 2 \text{ and } y > 0\}\Big) \stackrel{\text{def}}{=} \Big\{ \big\{\{x \in \mathbb{Q} : x^2 < 2 \text{ or } x < 0\}\big\}, \big\{\{x \in \mathbb{Q} : x^2 < 2 \text{ or } x < 0\}, \{y \in \mathbb{Q} : y^2 > 2 \text{ and } y > 0\}\big\} \Big\}$$

where

- $\mathbb{N}$ is defined as the smallest set containing $\varnothing$ and closed under the function $n \longmapsto n \cup \{n\}$
- $\sim_{\mathbb{Z}}$ is the equivalence relation on $\mathbb{N} \times \mathbb{N}$ defined by $(n_1, m_1) \sim_{\mathbb{Z}} (n_1, m_2) \iff n_1 + m_2 = m_1 + n_1$
- $\mathbb{Z} \stackrel{\text{def}}{=} (\mathbb{N} \times \mathbb{N})/\sim_{\mathbb{Z}} = \left\{\{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} n\} \mid n \in \mathbb{N} \times \mathbb{N}\right\}$
- the addition on $\mathbb{Z}$ is defined pointwise, and the multiplication as follows: $\{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (n_1, m_1)\} \times \{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (n_2, m_2)\} \stackrel{\text{def}}{=} \{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (n_1 n_2 + m_1 m_2, n_1 m_2 + n_2 m_1)\}$
- the ordering on $\mathbb{Z}$ is defined by: $\{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (n_1, m_1)\} < \{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (n_2, m_2)\} \iff n_1 + m_2 < n_2 + m_1$
- $\sim_{\mathbb{Q}}$ is the equivalence relation on $\mathbb{Z} \times \left(\mathbb{Z} \backslash \left\{\{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (\varnothing, \varnothing)\}\right\}\right)$ defined by $(n_1, m_1) \sim_{\mathbb{Q}} (n_1, m_2) \iff n_1 m_2 - m_1 n_1 = \{m \in \mathbb{N} \times \mathbb{N} \mid m \sim_{\mathbb{Z}} (\varnothing, \varnothing)\}$
- $\mathbb{Q} \stackrel{\text{def}}{=} (\mathbb{Z} \times (\mathbb{Z} \backslash \{0\}))/\sim_{\mathbb{Q}} = \left\{\{m \in \mathbb{Z} \times (\mathbb{Z} \backslash \{0\}) \mid m \sim_{\mathbb{Q}} n\} \mid n \in \mathbb{Z} \times (\mathbb{Z} \backslash \{0\})\right\}$
- the ordering on $\mathbb{Q}$ is given by: $\{m \in \mathbb{Z} \times (\mathbb{Z} \backslash \{0\}) \mid m \sim_{\mathbb{Q}} (n_1, m_1)\} < \{m \in \mathbb{Z} \times (\mathbb{Z} \backslash \{0\}) \mid m \sim_{\mathbb{Q}} (n_2, m_2)\} \iff n_1 m_2 < n_2 m_1$

• The following equivalence class of Cauchy sequence:

$$\left\{(b_n) \in \mathbb{Q}^{\mathbb{N}} \mid \forall \varepsilon > 0, \exists N, M \in \mathbb{N}, \left(\forall n_1, n_2 \geq N, |b_{n_1} - b_{n_2}| < \varepsilon\right) \wedge \left(\forall m \geq M |b_m - a_m| < \varepsilon\right)\right\}$$

where

- for every rational q, $|q| \stackrel{\text{def}}{=} \begin{cases} q, & \text{if } q \geq 0 \\ -q, & \text{if } q < 0. \end{cases}$
- $\mathbb{Q}^{\mathbb{N}} \stackrel{\text{def}}{=} \left\{\left\{b \mid b \subseteq \mathbb{N} \times Q \wedge \forall n \in \mathbb{N}, \exists! b_n \in Q, (n, b_n) \in b\right\} \mid Q \subseteq \mathbb{Q}\right\}$
- Cauchy sequences are elements (b_n) of $\mathbb{Q}^{\mathbb{N}}$ such that for each rational $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for all $n_1, n_2 > N, |b_{n_1} - b_{n_2}| < \varepsilon$.
- the addition and multiplication of Cauchy sequences is defined pointwise
- $(a_n) \in \mathbb{Q}^{\mathbb{N}}$ is the Cauchy sequence given by:

$$\begin{cases} a_0 = 2 \\ \forall n \in \mathbb{N}, a_{n+1} = \frac{1}{a_n} - \frac{a_n}{2} \end{cases}$$

Farfetchedness is not an overstatement! ∎

Another point where the shoe pinches has to do with **constructiveness**.

Indeed, advocates of constructivism (also called intuitionism) - among whom Kronecker and Brouwer - argue that proofs ought to be *constructive* in mathematics, that is that the existence of mathematical objects must be shown by explicitly constructing these objects (which are then said to be *computable*), rather than, for instance, merely demonstrating the impossibility of their non-existence. In this respect, proofs by contradiction and the axiom of choice should be rejected, contrary to what is usually done in classical set theory (on top of that, infinite sets tend to bring about non-computable methods and objects).

This is the context within which Vladimir Voevodsky's new foundational program lies, the main ideas of which were formulated in 2006/2009, after he discovered a mistake in a paper he had written several years earlier. Worried about the fact that this experience could be repeated, Voevodsky has since been championing computer proof assistants, by developing new constructive foundations for

mathematics, not based on set theory but on **Per Martin-Löf's dependent type theory** - called univalent foundations, also referred to as **homotopy type theory** (presented in [Uni13]). Besides, the functional programming language *Agda* (which can be used as a proof assistant) is also based on dependent type theory. According to Voevodsky, computer formalization should be necessary in classical mathematics (to verify theorems and constructions), insofar as many areas have become too complicated to be rigorously checked by humans.

Type Theory

Martin-Löf's dependent type theory, also called *intuitionistic/constructive type theory*, is underlaid by the fundamental notion of *dependent types*, which serve as a substitute for logical propositions *and* sets: the set-theoretical membership relation $x \in A$ is replaced by judgments of the form $x : A$ ("x is an object of type A").

Since types can depend on values, it generalizes *simple type theory* - to which the reader should be accustomed - used in functional programming languages (such as OCaml, Haskell, ...) which are themselves based on simply typed λ-calculus (that can be seen as an idealized version of a functional programming language).

Similarly to simple type theory, one can define - with resort to introduction and elimination rules - the boolean type $Bool$, the natural numbers type $\mathbb{N}$, and if A and B are two types:

- the product type $A \times B$ (an object of which corresponds to a pairing of an object of A **and** an object of B)
- the sum type $A + B$ (an object of which corresponds to an object of A **or** an object of B)
- the function type $A \longrightarrow B$ (whose objects are functions that send objects of A to objects of B)

But on top of the fact that it serves as a constructive foundation for mathematics, dependent type theory is also a **self-contained deductive system**. By *self-contained*, I mean that it can be regarded as a **bootstrapping system**, in which everything ultimately boils down to **types**: the mathematical objects, as well as the logical propositions!

Whereas in set theory, for instance, you'd better make a clear distinction between mathematical objects (sets, numbers, groups, spaces, etc…) and the meta-theory you use to talk about these objects: as a matter of facts, they're two clearly *distinct layers*.

As a matter of facts, mathematical statements in type theory are *themselves* types, and proving one these statements is tantamount to providing an element of its corresponding type. Thus, the type $A \longrightarrow B$ is to be thought as the proposition "A implies B" if A and B are logical propositions. Through the lens of this analogy between propositions and types, a proposition may be identified with the type of its proofs: this is known as the *Curry–Howard isomorphism*.

Type-theoretical concept	Logic counterpart
Type discipline	Logical system
Types	Formulas
λ-terms	Proofs

*The **Curry-Howard correspondence**, also known as the proof-as-programs/propositions-as-types interpretation*

NB According to the Curry–Howard correspondence, expressing any mathematical property is akin to constructing a type. Proof assistants based on dependently typed programming languages advantageously make use of it: for a compiler to check a proof of a given mathematical property, the user is to provide a constructive proof that the corresponding type has an object. As such, this approach can be powerfully applied to formal program verification.

A major difference between set theory and type theory consists in how equality is treated. In type theory, equality between two objects x and y is well formed only provided that they are inhabitants of a same type A. For instance, it does not even make sense - from a syntactical point of view - to ask whether $0 = \textbf{false}$, since $0 : \mathbb{N}$ and $\textbf{false} : Bool$; whereas in set theory, it makes perfect sense (it is even true if **false** is encoded as the empty set!). Therefore, this typed restriction of objects aims at classifying them, to avoid trivially nonsensical assertions. On the contrary, set theory does not prevent us from asking completely mathematically irrelevant questions, such as "is the imaginary number i a vector space?".

On top of that, equality between two objects $x : A$ and $y : A$, noted $x \simeq_A y$, is itself a new type, whose members correspond to proofs of the equality of x and y.

NB this is a particular case of the Curry-Howard correspondence mentioned before, since "$x = y$" is a logical assertion, and proving this assertion is tantamount to providing a inhabitant of the corresponding equality type.

Whereas in set theory, equality is a proposition which is classically either true of false.

Homotopy Type Theory

Homotopy theory studies homotopy groups, which - broadly speaking - witness the "shape" of a given topological space, with resort to homotopy classes (two mappings are said to be homotopic/homotopy equivalent if one can be "continuously deformed" into the other).

The link between type theory and homotopy theory stems from the fact that a type A can be *abstractly* seen as a topological space, whose points are its objects, and whose features stem from those of A. And if $x, y : A$ (an abbreviation for $x : A$ and $y : A$), the equality type $x \simeq_A y$ represents the paths from x (start point) to y (end point).

Given two parallel paths $p, q : x \simeq_A y$, a path $r : p \simeq_{x \simeq_A y} q$ corresponds to a homotopy (a continuous maps between paths), referred to as a *2-path*. In a similar fashion, $r \simeq_{p \simeq_{x \simeq_A y} q} s$ the type of 3-paths, and so on: *higher* homotopy theory comes in!

The tower comprised of these paths, paths of paths, paths of paths of paths, ... arising from a given type A encode what we call an ω-groupoid (or ∞-groupoid). It is in this sense that *types are said to form ∞-groupoids*.

But that is not the end of the story: the key point of Voevodsky's univalent program is the Univalence Axiom, which basically states that equality of types is (weakly) equivalent to weak equivalence, and

this implies functional extensionality (which means that two functions are equal if their values are equal at every argument). As such, this axiom does not fit into the pattern of introduction and elimination rules of type theory, and whether it is possible to provide a constructive justification of it is still an open issue, which turns out to be very pressing and problematic, from a constructive standpoint.

Eliminating univalence is thus a bit of a holy grail, and it appears that it might be done by formalizing a weak ω-groupoid model of Type Theory inside Type Theory. Indeed, by constructing an ω-groupoid of types seen as ω-groupoids, equivalences between types would be witnessed by actual morphisms of this ambient ω-groupoid, which would themselves turn out to be inhabitants of the equality of these types.

As a first step towards this major challenge ahead, we shall show that each type forms a particular kind of ω-groupoid, posed by Guillaume Brunerie, whose type-theoretically defined syntax is well suited for a formalization within Type Theory in the dependently typed programming language Agda.

3. Category Theory

Category theory is a theory of "structure-preserving functions" that provides a unifying framework to study abstractly other mathematical structures and will be used throughout this report to define the semantics of type theory. Numerous mathematical structures[1] can be successfully formalized as categories[AT10].

The Basics

Definition

Definition 3.0.1 — a category $\mathscr{C}$ is given by:

- **Structure**:
  - *Objects:* a class of **objects**, denoted $|\mathscr{C}|$
  - *Arrows:* for each pair of objects $X, Y : |\mathscr{C}|$, a class[a] of **arrows** (or **morphisms**) from X to Y denoted by $\mathrm{Hom}_{\mathscr{C}}(X, Y)$, or $\mathscr{C}(X, Y)$ (called *hom-class*).
    For each $f \in \mathscr{C}(X, Y)$,
    - $f \in \mathscr{C}(X, Y)$ is noted $f : X \longrightarrow Y$ or $X \xrightarrow{f} Y$
    - X is the domain of f, Y its codomain
  - *Composition:* for each pair of morphisms $f : X \longrightarrow Y$ and $g : Y \longrightarrow Z$, a composite arrow $g \circ f : A \longrightarrow C$ (and $f ; g : A \longrightarrow C$)
  - *Identity:* for each object $A : |\mathscr{C}|$, an identity morphism $\mathrm{id}_X : A \longrightarrow A$
- **Laws**:
  - *Associativity of composition:* for each $f : X \longrightarrow Y, g : X \longrightarrow Z, h : Z \longrightarrow T$,

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

  - *Unit law:* for each $f : X \longrightarrow Y$,

$$\mathrm{id}_Y \circ f = f \circ \mathrm{id}_X$$

[a] The fact that the collections of objects and arrows are taken to be *classes* is a foundational issue, to avoid Russel's paradox. The category is said to be *locally small* if the classes of arrows are sets, and *small* if the classes of arrows *and* objects are sets (the former are then called *hom-sets*).

■ **Example 3.1 — Examples of categories (objects and arrows).**

- the category Set of sets and arbitrary mappings
- **Mon**: monoids and monoid homomorphisms (multiplication-preserving morphisms)

[1] such as sets, groups, rings, vector spaces, topological spaces, and, we are just about to see: type theory!

- **Grp**: groups and group homomorphisms
- **Vec**$_{\mathbb{K}}$: vector spaces over a field $\mathbb{K}$ and linear maps
- **Graph**: graphs and graph homomorphisms
- **Top**: topological spaces and continuous mappings
- **Pos**: posets and monotone functions
- the category $\mathbb{1}$ is comprised of 1 object, and the identity arrow

■

Vocabulary 3.1 — a subcategory of a category $\mathscr{C}$ is a category whose objects are objects in $\mathscr{C}$ and whose arrows are arrows in $\mathscr{C}$, with the same identities and composition of arrows.

Let $\mathscr{C}$ be a category.

Definition 3.0.2 — A morphism $f : X \longrightarrow Y$ in $\mathscr{C}$ is an isomorphism if there exists an arrow $f^{-1} : Y \longrightarrow X$ (the inverse of f) such that:

$$f^{-1} \circ f = \mathrm{id}_X \text{ and } f \circ f^{-1} = \mathrm{id}_Y$$

Principle of duality

Definition 3.0.3 — The dual category $\mathscr{C}^{op}$ of a category $\mathscr{C}$ has the same objects as $\mathscr{C}$, and all of its arrows are "turned around":

$$f : X \longrightarrow Y \text{ in } \mathscr{C} \iff f^* : Y \longrightarrow X \text{ in } \mathscr{C}^{op}$$

For all $f : X \longrightarrow Y, g : Y \longrightarrow Z$ in $\mathscr{C}$, the compostion in $\mathscr{C}^{op}$ is defined as follows:

$$f^* \circ g^* \stackrel{\text{def}}{=} (g \circ f)^*$$

Vocabulary 3.2 — Dual proposition Given a proposition P, one forms the dual proposition P^* by turning around all morphism and replacing each composition $g \circ f$ by $f \circ g$.

Proposition 3.0.1 — Duality principle. If a proposition P that expresses the existence of some objects/morphisms OR an equality of compositions of arrows is true in any category, *then* P^* is also true in any category.

Proof. Proving P^* in any category $\mathscr{C}$ amounts to proving P in any category $\mathscr{C}^{op}$. But, by assumption, P stands in any category. ■

Commutative diagrams

Definition 3.0.4 — A finite diagram D in a category $\mathscr{C}$ is a finite oriented multigraph (that is, an finite (finite number of vertices and edges) oriented graph with possibly multiple edges between two vertices) whose vertices are objects of $\mathscr{C}$ and edges between two vertices are arrows between the two corresponding objects in $\mathscr{C}$.

Vocabulary 3.3 — A finite diagram D commutes if any two paths in it with common source and target such that at least one these paths has length (strictly) greater than 1, are equal.

NB From now on, the adjective "finite" will be omitted when the meaning is clear.

■ **Example 3.2**

Saying that the following diagram commutes...	
$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle h} \\ Z & \xrightarrow{i} & T \end{array}$	... means that $h \circ f = i \circ g$
$E \xrightarrow{e} X \underset{g}{\overset{f}{\rightrightarrows}} Y$	... means that $f \circ e = g \circ e$ but *not* $f = g$

■

Functors and Natural Transformations

Functors

Let $\mathscr{C}, \mathscr{D}$ be two categories.

Definition 3.0.5 — a (covariant) functor $F : \mathscr{C} \longrightarrow \mathscr{D}$ from $\mathscr{C}$ to $\mathscr{D}$ is a mapping such that:

- for each object $X : |\mathscr{C}|$ in $\mathscr{C}$, $F(X)$ (also written FC) is an object in $\mathscr{D}$
- for each morphism $f : X \longrightarrow Y$ in $\mathscr{C}$, $F(f) : F(X) \longrightarrow F(Y)$ (also written Ff) is a morphism from $F(X)$ to $F(Y)$ in $\mathscr{D}$
- *Preservation of composition:* $F(g \circ f) = F(g) \circ F(f)$ for all morphisms $f : X \longrightarrow Y$, $g : Y \longrightarrow Z$ in $\mathscr{C}$.
- *Preservation of identity:* $F(\mathrm{id}_X) = \mathrm{id}_{F(X)}$ for each object $X : |\mathscr{C}|$ in $\mathscr{C}$

NB The small categories form a category Cat, whose objects are small categories and arrows are functors between them.

Vocabulary 3.4 — a contravariant functor from $\mathscr{C}$ to $\mathscr{D}$ is a convariant functor from $\mathscr{C}^{op}$ to $\mathscr{D}$: in other words, with respect to $\mathscr{C}$, it "turn arrows around" and "reverse composition".
Vocabulary 3.5 — a presheaf on a category $\mathscr{C}$ is a functor from $\mathscr{C}^{op}$ to Set

■ **Example 3.3 — Hom-functors.** Let $\mathscr{C}$ be a locally small category (i.e. all hom-classes are sets). For all objects $X, Y : |C|$, the convariant and contravariant hom-functors are defined as follows:

- *Covariant hom-functor* $\mathrm{Hom}(X,\_) : \mathscr{C} \longrightarrow \mathrm{Set}$*:*

$$\mathrm{Hom}(X,\_) \stackrel{\text{def}}{=} \begin{cases} \mathscr{C} & \longrightarrow \mathrm{Set} \\ A : |C| & \mapsto \mathrm{Hom}(X,A) \\ f : A \to B & \mapsto Hom(X,f) \stackrel{\text{def}}{=} \begin{cases} \mathrm{Hom}(X,A) & \longrightarrow \mathrm{Hom}(X,B) \\ g : X \to A & \mapsto f \circ g : X \to B \end{cases} \end{cases}$$

- *Contravariant hom-functor* $\mathrm{Hom}(\_,Y) : \mathscr{C}^{op} \longrightarrow \mathrm{Set}$*:*

$$\mathrm{Hom}(\_,Y) \stackrel{\text{def}}{=} \begin{cases} \mathscr{C}^{op} & \longrightarrow \mathrm{Set} \\ A : |C| & \mapsto \mathrm{Hom}(A,Y) \\ f : A \to B & \mapsto Hom(f,Y) \stackrel{\text{def}}{=} \begin{cases} \mathrm{Hom}(B,Y) & \longrightarrow \mathrm{Hom}(A,Y) \\ g : B \to Y & \mapsto g \circ f : A \to Y \end{cases} \end{cases}$$

■

Natural Transformations

Let $F, G : \mathscr{C} \longrightarrow \mathscr{D}$ be two functors.

Definition 3.0.6 — a natural transformation ϕ is a family of arrows in $\mathscr{D}$

$$\big(\phi_X : F(X) \longrightarrow G(X)\big)_{X:|C|}$$

such that, for any $f : X \longrightarrow Y$ in $\mathscr{C}$, the following diagram commutes:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(f)} & F(Y) \\ \downarrow{\scriptstyle \phi_X} & & \downarrow{\scriptstyle \phi_Y} \\ G(X) & \xrightarrow{G(f)} & G(Y) \end{array}$$

Vocabulary 3.6 — Natural isomorphism.
If ϕ_X is an isomorphism (in $\mathscr{D}$) for each object $X : |\mathscr{C}|$, then ϕ is called a *natural isomorphism*. Two functors F and G are said to be *naturally isomorphic* if there exists a natural isomorphism from F to G.

NB The *functor category* $\mathscr{D}^{\mathscr{C}}$ is the category whose objects are functors from $\mathscr{C}$ to $\mathscr{D}$ and arrows are natural transformations between them.

Definition 3.0.7 — Representable functors A covariant (resp. contravariant) functor $F : \mathscr{C} \to$ Set is said to be *representable* if there exists a natural isomorphism ϕ from F to the covariant (resp. contravariant) hom-functor $\mathrm{Hom}(X, \_)$ (resp. $\mathrm{Hom}(\_, X)$) for some object $X : |\mathscr{C}|$.
Vocabulary 3.7 If so, the pair (X, ϕ) is a *representation* of F. Representations of functors are unique up to isomorphism.

Adjointness

Let $\mathscr{C}$ and $\mathscr{D}$ be two categories, and two functors $F : \mathscr{C} \longrightarrow \mathscr{D}$ and $G : \mathscr{D} \longrightarrow \mathscr{C}$.

Definition 3.0.8 — Adjoints.
F is said to be a left adjoint to G (noted $F \dashv G$) and G a right adjoint to F, if for each $X : |\mathscr{C}|$, $Y : |\mathscr{D}|$, there is a bijection

$$\Phi_{X,Y} : \mathrm{Hom}_{\mathscr{D}}(F(X), Y) \longrightarrow \mathrm{Hom}_{\mathscr{C}}(X, G(Y))$$

that is *natural* in X and Y, which means that for each for all morphisms $f : X' \longrightarrow X$ in $\mathscr{C}$ and $g : Y \to Y'$ in $\mathscr{D}$, the following diagram commutes:

$$\begin{array}{ccc} \mathrm{Hom}_{\mathscr{D}}(F(X), Y) & \xrightarrow{\mathrm{Hom}_{\mathscr{D}}(F(f),g)} & \mathrm{Hom}_{\mathscr{D}}(F(X'), Y') \\ \downarrow{\scriptstyle \Phi_{X,Y}} & & \downarrow{\scriptstyle \Phi_{X',Y'}} \\ \mathrm{Hom}_{\mathscr{C}}(X, G(Y)) & \xrightarrow{\mathrm{Hom}_{\mathscr{C}}(f,G(g))} & \mathrm{Hom}_{\mathscr{C}}(X', G(Y')) \end{array}$$

Universal Properties

Intuitively, the notion of universal properties corresponds to constructions such that particular hypotheses ensure the existence and unicity of a morphism.

Initial and terminal objects

Definition 3.0.9 — An object $\mathbb{0}$ in a category $\mathscr{C}$ is initial if for every object $X : |\mathscr{C}|$, there exists a unique arrow from $\mathbb{0}$ to X.

Definition 3.0.10 — An object $\mathbb{1}$ in a category $\mathscr{C}$ is terminal if for every object $X : |\mathscr{C}|$, there exists a unique arrow from X to $\mathbb{1}$.

Vocabulary 3.8 — Weakly initial/terminal if the morphisms are not required to be unique, $\mathbb{0}$ (respectively $\mathbb{1}$) is said to be *weakly* initial (respectively terminal).

NB Initial and terminal objects are dual notions: $\mathbb{0}$ is initial in $\mathscr{C}$ if and only if $\mathbb{1}$ is terminal in $\mathscr{C}^{op}$

Theorem 3.0.2 Initial (respectively terminal) objects are unique up to isomorphism.

Proof. If $\mathbb{0}$ and $\mathbb{0}'$ are two initial objects in a category $\mathscr{C}$: there exist two morphisms $\iota_{\mathbb{0}} : \mathbb{0} \longrightarrow \mathbb{0}'$ and $\iota_{\mathbb{0}'} : \mathbb{0}' \longrightarrow \mathbb{0}$ by initiality of $\mathbb{0}$ and $\mathbb{0}'$. By uniqueness, $\iota_{\mathbb{0}'} \circ \iota_{\mathbb{0}} = \mathrm{id}_{\mathbb{0}}$; and similarly: $\iota_{\mathbb{0}} \circ \iota_{\mathbb{0}'} = \mathrm{id}_{\mathbb{0}'}$.

For terminal objects, the result is due to the duality principle. ■

Theorem 3.0.3 — Left adjoint and initial object.
Let $\mathscr{D}$ be a category, $\mathbb{1}$ the category with a single object $\bullet$, and $U : \mathscr{D} \longrightarrow \mathbb{1}$ the unique functor from $\mathscr{D}$ to $\mathbb{1}$.

If U has a left adjoint $F \dashv U$, then $F(\bullet)$ is an initial object in $\mathscr{C}$.

Proof. For each $Y : |\mathscr{D}|$, there is a bijection

$$\mathrm{Hom}_{\mathscr{D}}(F(\bullet), Y) \simeq \underbrace{\mathrm{Hom}_{\mathbb{1}}(\bullet, G(Y))}_{=\{\mathrm{id}_\bullet\}}$$

So for each $Y : |\mathscr{D}|$, there exists a unique morphism in $\mathrm{Hom}_{\mathscr{D}}(F(\bullet), Y)$. ■

Products and pullbacks

Products

Let X, Y be objects in a category $\mathscr{C}$.
Vocabulary 3.9 — An X,Y-pairing is a triple (P, p_X, p_Y) where $P : |\mathscr{C}|$, $p_X : P \longrightarrow X$, and $p_Y : P \longrightarrow Y$.

Definition 3.0.11 — The category $\mathbf{Pair}(X,Y)$ of X,Y-pairings is the category whose

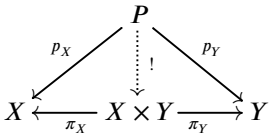
1. *objects* are the X, Y-pairings
2. *morphisms* are defined as follows: for all pairings (P, p_X, p_Y) and (Q, q_X, q_Y),

$$f \in \mathbf{Pair}(X,Y)\big((P, p_X, p_Y), (Q, q_X, q_Y)\big) \iff f \in \mathscr{C}(P,Q) \text{ and } \begin{array}{ccccc} & & P & & \\ & \swarrow^{p_X} & \downarrow f & \searrow^{p_Y} & \\ X & \xleftarrow[q_X]{} & Q & \xrightarrow[q_Y]{} & Y \end{array} \text{ commutes}$$

Definition 3.0.12 — $(X \times Y, \pi_X, \pi_Y)$ is a product of X and Y if it is terminal in $\mathbf{Pair}(X,Y)$ (if so, π_X and π_Y are called *projections*).

NB

1. Unfolding the definition gives that for each X, Y-pairing (P, p_X, p_Y), there exists a unique morphism from P to $X \times Y$ such that the following diagram commutes:



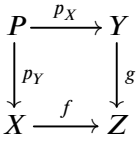
2. Since the product is a terminal object, it results from 3.0.2 that it is unique up to isomorphism.

Vocabulary 3.10 — $\mathscr{C}$ is said to have products if each pair of objects has a product in $\mathscr{C}$.

Pullbacks

Let $X \xrightarrow{f} Z \xleftarrow{g} Y$ be a pair of morphisms in a category $\mathscr{C}$.

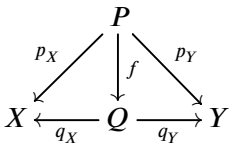
Vocabulary 3.11 — A f, g-cone is a triple (P, p_X, p_Y) where $P : |\mathscr{C}|$, $p_X : P \longrightarrow X$, and $p_Y : P \longrightarrow Y$ such that the following diagram commutes:



Definition 3.0.13 — The category Cone(f, g) of f, g-cones is the category whose

1. *objects* are the f, g-cones
2. *morphisms* are defined as follows: for all cones (P, p_X, p_Y) and (Q, q_X, q_Y),

$$f \in \mathbf{Pair}(X, Y)\left((P, p_X, p_Y), (Q, q_X, q_Y)\right) \iff f \in \mathscr{C}(P, Q) \text{ and }$$

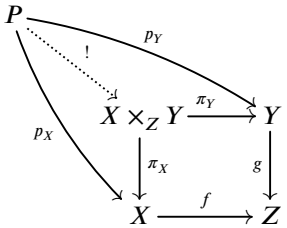


commutes

Definition 3.0.14 — $(X \times_Z Y, \pi_X, \pi_Y)$ is a pullback of f along g if it is terminal in **Cone**(f, g).

NB

1. Unfolding the definition gives that for each f, g-cone (P, p_X, p_Y), there exists a unique morphism from P to $X \times_Z Y$ such that the following diagram commutes:



2. Again, the pullback is unique up to isomorphism (since it is a terminal object).

Vocabulary 3.12 — $\mathscr{C}$ is said to have pullbacks if, for each pair of morphisms in $\mathscr{C}$, there exists a f, g-cone which is the pullback of f along g.

Limits

The notion of limit encompasses universal constructions such as products and pullbacks.

Let I and $\mathscr{C}$ be two categories.

Definition 3.0.15 — A diagram of shape I in $\mathscr{C}$ is a functor from I to $\mathscr{C}$ (which is thought of as indexing a collection of objects and morphisms in $\mathscr{C}$ with the shape of I).
Vocabulary 3.13 I is called the *index category* of the diagram.

Definition 3.0.16 — The diagonal functor $\Delta : \mathscr{C} \longrightarrow \mathscr{C}^I$
In the functor category $\mathscr{C}^I$, for each object $N : |\mathscr{C}|$, there is a constant functor fixing the object N:

$$\Delta(N) : |\mathscr{C}^I|$$

which sends every object to N and every morphism to id_N.

The diagonal functor $\Delta : \mathscr{C} \longrightarrow \mathscr{C}^I$ sends

- each object $N : |\mathscr{C}|$ to $\Delta(N)$
- each morphism $f : N \longrightarrow M$ to the natural transformation defined by $\phi_i \stackrel{\text{def}}{=} f$ for each $i : |I|$

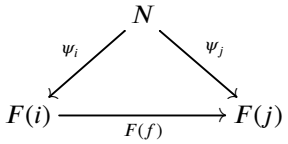
Definition 3.0.17 — a cone from $N : |\mathscr{C}|$ to a diagram $F : I \longrightarrow \mathscr{C}$ is a natural transformation from $\Delta(N)$ to F.
Vocabulary 3.14 — $\mathbf{Cone}(N, F)$ is the set of cones from N to F.

$$\mathbf{Cone}(\_, F) \stackrel{\text{def}}{=} \mathscr{C}^I(\Delta(\_), F) : \mathscr{C} \longrightarrow \mathrm{Set}$$

Definition 3.0.18 — A limit of the diagram $F : I \longrightarrow \mathscr{C}$ is a pair $(L, \phi_L(\mathrm{id}_L))$ where

- $L : |\mathscr{C}|$
- $\phi : \mathrm{Hom}(\_, L) \to \mathbf{Cone}(\_, F)$ is a natural isomorphism
- (L, ϕ) is a *representation* of the functor $\mathbf{Cone}(\_, F) : \mathscr{C} \longrightarrow \mathrm{Set}$.

NB Cones to $F : I \longrightarrow \mathscr{C}$ may also be defined as pairs (N, ψ) - where $N : |C|$ and ψ is a family of morphisms $\left(\psi_i : N \to F(i)\right)_{i:|I|}$ - such that for every morphism $f : i \longrightarrow j$ in I, the following diagram commutes:



A limit of the diagram $F : J \longrightarrow C$ is then a *universal cone*, in the sense that it is a terminal object in the category of cones to F; i.e. a cone (L, φ) to F such that for any other cone (N, ψ) to F, there exists a unique morphism $f : N \to L$ such that $\varphi_i \circ u = \psi_i$ for each $i : |I|$.

Vocabulary 3.15 — A complete category is a category that has all small limits (*i.e.* all limits of shape I for every small category I).

Proposition 3.0.4 The categories Cat and Set are complete.

Vocabulary 3.16 — The functor G preserves the limits of F if $(G(L), G(\varphi))$ is a limit of $G(F)$ whenever (L, φ) is a limit of F.

Theorem 3.0.5 — Left adjoint functor theorem.
Let $\mathscr{C}$ and $\mathscr{D}$ be two locally small categories such that $\mathscr{C}$ is complete, and $G : \mathscr{C} \longrightarrow \mathscr{D}$ a functor.

If

- G preserves limits
- *Solution set condition*: for all $Y : |\mathscr{D}|$, there exists a **set** of objects $S \subseteq |\mathscr{C}|$ such that: for all $X : |\mathscr{C}|$ and $u : Y \longrightarrow G(X)$, there exists an object $X' : |S|$ and a morphism $v : X' \longrightarrow X$ such that u factors through $G(v)$, i.e. $u \overset{\text{def}}{=} G(v) \circ u'$, for a certain $u' : Y \longrightarrow G(X')$.

then G has a left adjoint.

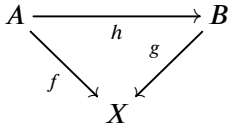
Other categorical constructions

Slice and coslice categories

Let $\mathscr{C}$ be a category and $X : |\mathscr{C}|$ an object of $\mathscr{C}$.

Definition 3.0.19 — The slice category $\mathscr{C}/X$ is described as follows:

- An object of $\mathscr{C}/X$ is a pair (A, f) where $A : |\mathscr{C}|$ and $f : A \longrightarrow X$ in $\mathscr{C}$.
- An arrow of $\mathscr{C}/X$ from (A, f) to (B, g) is an arrow $h : A \longrightarrow B$ such that the following diagram commutes:



Vocabulary 3.17 — The coslice category $X\backslash\mathscr{C}$ is the dual category of $\mathscr{C}/X$.

Category of elements

Let $F : \mathscr{C} \longrightarrow \mathrm{Set}$ be a functor.

Definition 3.0.20 — The category of elements $\int^{\mathscr{C}} F$ is described as follows:

- An object of $\int^{\mathscr{C}} F$ is a pair (A, a) where $A : |\mathscr{C}|$ and $a \in F(A)$.
- An arrow of $\int^{\mathscr{C}} F$ from (A, a) to (B, b) is an arrow $h : A \longrightarrow B$ such that $Ff(a) = b$

If F is contravariant, the condition on the arrows becomes $Ff(b) = a$

Groupoids

Definition 3.0.21 — A groupoid is a small category in which every morphism is an isomorphism.

Vocabulary 3.18 — Higher category theory is the generalization of category theory where there are not only morphisms between objects, but generally $n+1$-morphisms between n-morphisms, for all $n \geq 1$.

4. Syntax of Dependent Type Theory

Basics

NB Readers are assumed to have a certain working familiarity with a functional programming language, such as Haskell or OCaml.

Martin Löf's Dependent Type Theory, denoted by **MLTT**, can be seen as a functional programming language. It is more expressive than simply-typed λ-calculus (the idealized version of simply-typed functional programming languages such as OCaml), in that most mathematical concepts can be formalized thereby, owing to the fact that types can depend on values of another type.

Definition 4.0.1 — The universe type $\mathcal{U}$
All types are regarded as objects of a particular type $\mathcal{U}$ (the *universe*).

To avoid Russel's paradox, $\mathcal{U}$ cannot be an object of itself so there is an infinite hierarchy of universes $\mathcal{U}_0, \mathcal{U}_1, \ldots$ such that $\mathcal{U}_n$ is of type $\mathcal{U}_{n+1}$ for all n.

By abuse of notation, though, the subscripts are omitted and we still write $\mathcal{U}$.

■ **Example 4.1** An example of a dependent type is the type of n-tuples: $A^n : \mathcal{U}$, whose elements are of the form $(a_1, \ldots, a_n) : A^n$, where $a_i : A$.

A function into $\mathcal{U}$ can be used to represent such a dependent type:

$$\text{Vec} : \mathcal{U} \longrightarrow \mathbb{N} \longrightarrow \mathcal{U}$$
$$\text{Vec}\, A\, n :\equiv A^n$$

■

Via the *proposition-as-types* correspondence, dependent types can be viewed as logical predicates or relations:

■ **Example 4.2**

- Even $: \mathbb{N} \longrightarrow \mathcal{U}$ assigns to any natural number n a proof that n is even. In this regard, Even 4 is inhabited, but Even 3 is empty.
- $>: \mathbb{N} \longrightarrow \mathbb{N} \longrightarrow \mathcal{U}$ is such that $m > n$ is the type of evidence that m is greater than n

■

Type theory is comprised of

- contexts (denoted by the letters $\Gamma, \Delta, \Theta, \ldots$
- types (denoted by $A, B, C, \ldots$)

- terms (represented by $a, b, c, t, u, v, ...$)
- substitutions (represented by $\gamma, \theta, \delta, ...$).

and a collection of *inference rules*.

On top of that, we may assume that we are given a stock of variables[1] (represented by $x, y, ...$).

Contexts

The notion of context arise from the need to express what it is that dependent types can depend on.

Definition 4.0.2 — A context Γ is a list $x_1 : A_1, \ldots, x_n : A_n$ of variables, such that each type A_i can depend[a] on variables of earlier types $A_1, \ldots, A_{i-1}$.

NB We may point out that dependence by writing $x_1 : A_1, x_2 : A_2(x_1), \ldots, x_n : A_n(x_1, \ldots, x_{n-1})$

[a] since types are *dependent*, they may depend on terms

Intuitively, contexts are lists of assumptions of the form $x : A$ (a variable name and the type of this variable). In mathematics, theorems are indeed always demonstrated relatively to a fixed context.

■ **Example 4.3** For instance, when you say

> let n be a natural number, p be a prime number, $\mathbb{K}$ a field of order p^n, and V a $\mathbb{K}$-vector space

you are placing yourself in a context Γ containing the variables n, p, $\mathbb{K}$, and V, having the desired types.
■

Types and Terms

Judgements

Types and terms can be formed according to **inference rules**, by means of which are derived **judgments** of the form:

[1] but we may do without these variables, by resorting to *De Bruijn* indices

Judgement	Interpreted as
$\Gamma \vdash A$ type	A is a well-formed type in the context Γ *If A is a type representing a **proposition**:* then a is called a **witness** (or **evidence**) of the truth of A
$\Gamma \vdash a : A$	a is a term of type A in the context Γ
$\Gamma \vdash A \equiv B$	A and B are equal types in the context Γ
$\Gamma \vdash a \equiv b : A$	a and b are **definitionally**/judgmentally (that is, **by construction**) **equal** objects of type A in the context Γ

NB The variables of Γ may appear free in the judgment together with their respective types.

If $x : A$ is a typing declaration in Γ, then we have $\Gamma \vdash x : A$.

Introducing brand new types and terms

Inference rules introducing new types and terms usually follow this recurrent pattern:

an operator or

For example, application is an eliminator for function types; projections are eliminators for record types; conditional expression is an eliminator for Booleans.

The opposite of an eliminator is a

Computation happens whenever you feed a constructor to the corresponding eliminator.

1. **Formation rules:** how to **construct this type**
2. **Constructors / Introduction rules:** how to **construct elements** of that type (e.g. lambda-abstraction for functions, creating tuples for records, constants 'true' and 'false' for booleans, ...).
3. **Eliminators / Elimination rules:** how to **use elements** of that type (e.g. application for function types, projections for record types, conditional expression for booleans, ...).

   NB The use of eliminators is also called **induction**.
   Vocabulary 4.1 *Recursors* (of which use is called **recursion**) are special cases of eliminators (the difference will be made precise later).
4. **Computation rules:** how **eliminators** are to *act on* **constructors** (how to feed constructors to the corresponding eliminators).
5. ***[Optional]* Uniqueness principle:**

- *either* every function **into** that type is uniquely determined: i.e. how **constructors act on eliminators** (every element is uniquely determined by what you get after having applied eliminators, and can then be reconstructed by applying constructors)
- *or* every function **from** that type is uniquely determined

■ **Example 4.4 — for function types (i.e. of the form $A \longrightarrow B$).**

1. **Formation** rule:
   If A and B are types in Γ, then we can derive $\Gamma \vdash A \longrightarrow B$ type
2. **Constructor / Introduction** rule: λ-abstraction
   If A, X are two types and $u : A$ in Γ, then we can derive $\Gamma \vdash \lambda x.u : X \longrightarrow A$
3. **Eliminators / Elimination** rules: function application
   If A, X are two types, $u : A$ and $y : X$ in Γ, then we can derive $\Gamma \vdash (\lambda x.u)(y) : A$
4. **Computation** rule: β-reduction
   Under the same hypotheses: $\Gamma \vdash (\lambda x.u)(y) \equiv u[x := y] : A$
5. **Uniqueness** principle: η-expansion (each function is uniquely determined by its values) is expressed by $\Gamma \vdash \lambda x.u \equiv \lambda y.(\lambda x.u)y : X \longrightarrow A$

■

Π-Types / Dependent function types

Π-types generalize function types to model *dependent* functions: that is, functions whose result type depends on the argument (e.g. functions $f : (x : A) \longrightarrow B(x)$, such that $f(x) : B(x)$ for each $x : A$).

NB The non-dependent function type is a special case of Π-type.

From a logical point of view (via the Curry-Howard correspondence), dependent products are types expressing universal quantification, such as $\forall x \in A, B(x)$.

■ **Example 4.5** A proof of $\forall x, y \in \mathbb{N}, x+y = y+x$ is a dependent function of type $f : \prod_{x:\mathbb{N}} \prod_{y:\mathbb{N}} x+y = y+x$, such that, for instance, $f(1,2)$ (also denoted by $f\,1\,2$) is evidence that $1+2 = 2+1$. ■

Inference rules

- *Formation*:
$$\frac{\Gamma \vdash A \qquad \Gamma, x:A \vdash B}{\Gamma \vdash \prod_{x:A} B}$$
- *Introduction*:
$$\frac{\Gamma, x:A \vdash b:B}{\Gamma \vdash \lambda x.b: \prod_{x:A} B}$$
- *Elimination*:
$$\frac{\Gamma \vdash f: \prod_{x:A} B \qquad \Gamma \vdash a:A}{\Gamma \vdash f(a): B[x := a]}$$

NB where $B[x := a]$ (resp. $b[x := a]$) is the type (resp. term) obtained by substituting the term a for each free (avoiding variable capture) occurrence of the variable x in B (resp. b).

- *Computation*:
$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x.b)\, a \equiv b[x := a] : B[x := a]}$$
- *Uniqueness*:
$$\frac{\Gamma \vdash f : \prod_{x:A} B}{\Gamma \vdash \lambda x.f(x) = f : \prod_{x:A} B}$$
- *Congruence rule*:
$$\frac{\Gamma \vdash A \equiv A' \qquad \Gamma, x : A \vdash B \equiv B'}{\Gamma \vdash \prod_{x:A} B \equiv \prod_{x:A'} B'}$$

Similarly, Σ-types are used to represent *existential quantification*, as an indexed pair, where the type of the second component depends on the first. That is, if $(a, b) : \Sigma_{(x:A)} B(x)$, then $a : A$ and $b : B(a)$. The first component is seen as the instance for which the logical property is supposed to hold and the second component as a proof that it actually holds for this instance.

■ **Example 4.6** A proof of $\exists n \in \mathbb{N}, \mathrm{Even}\, n$ corresponds to $\sum_{n:\mathbb{N}} \mathrm{Even}\, n$: there exists an even natural number *if and only if* there exists a natural number n and a proof of type $\mathrm{Even}\, n$. For instance, $(2, p)$, where $p : \mathrm{Even}\, 2$ is such a proof ■

Intensional Equality

Whenever $a, b : A$ can be derived, we have a type $a \simeq_A b$ called *propositional equality*.
Vocabulary 4.2 — Propositional equality : a and b are said to be **propositionally equal** whenever $a \simeq_A b$ is inhabited by a witness.

Recall that in homotopy type theory, a type is regarded as a topological space, in which elements of the type are interpreted as points, and a propositional equality proofs as paths between the corresponding points.

NB When forming a new type such that the uniqueness principle does not result in a judgmental equality, propositional equality can often be demonstrated instead.

- *Formation*:
$$\frac{\Gamma \vdash A \quad \Gamma \vdash a : A \quad \Gamma \vdash a' : A}{\Gamma \vdash a \simeq_A a'}$$
- *Introduction*:
$$\frac{\Gamma \vdash A \quad \Gamma \vdash a : A}{\Gamma \vdash \mathrm{refl} : a \simeq_A a}$$
- *Elimination*:
$$\frac{\Gamma \vdash x : A \quad \Gamma, y : A, p : x \simeq_A y \vdash P(y, p) \quad \Gamma \vdash d : P(x, \mathrm{refl}_x)}{\Gamma, y : A, p : x \simeq_A y \vdash \mathrm{J}(y, p, d) : P(y, p)}$$

NB Here, the J eliminator says that, to prove a proposition $P(y, p)$ about an arbitrary path p from a fixed element x to any endpoint y, it is enough to consider the case where the endpoint is also x itself, and the path is the identity path from x to x (the reflexivity refl_x).

- *Computation*:

$$\mathrm{J}(x, \mathrm{refl}_x, d) \equiv d : P(x, \mathrm{refl}_x)$$

In Martin Löf's type theory, equality is said to be *intensional*, as opposed to *extensional*:

- *extensional* equality involves a "reflection rule", which states that any propositional equality can be converted to definitional equality
- *intensional* equality has no such rule: instead, it is replaced by the J eliminator and its computation rule, as presented above

Eliminators and recursors

To form inductive types, an eliminator (resp. recursor) for a type T can be delivered in the form of an operator, given some constructors. The type of this operator expresses an **induction principle** (resp. **recursion principle**) and follows this overall general pattern:

- for **recursors**:

$$\prod_{C:\mathcal{U}} \boxed{?} \longrightarrow \Big(T \longrightarrow C\Big)$$

- for **eliminators**:

$$\prod_{C:T\longrightarrow\mathcal{U}} \boxed{?} \longrightarrow \prod_{x:T} C(x)$$

where $\boxed{?}$ is defined on a case-by-case basis, depending on the type.

NB Recursors are special cases of eliminators where the family C is constant.

A more detailed account of the different types formers in dependent type theory is set out in appendix.

Substitutions

Substitutions (also called *context morphisms*) operate on types as well as terms, insofar as types can contain terms. If we view contexts as lists of assumptions, substitutions are instantiations of a given list of assumptions in terms of other assumptions (given by another context).

Definition 4.0.3 — A substitution $\sigma : \Delta \longrightarrow \Gamma$ between two contexts Δ and $\Gamma \stackrel{\text{def}}{=} x_1 : A_1,\ x_2 : A_2(x_1),\ \ldots,\ x_n : A_n(x_1, \ldots, x_{n-1}))$ is a list of terms

$$\sigma \stackrel{\text{def}}{=} (t_1, \ldots, t_n)$$

such that

$$\begin{aligned}
&\Delta \vdash t_1 : A_1 \\
&\Delta \vdash t_2 : A_1(t_2) \\
&\vdots \\
&\Delta \vdash t_n : A_n(t_1, t_2, \ldots, t_{n-1})
\end{aligned}$$

NB σ can be regarded as an interpretation of the variables of Γ within Δ, by substituting terms available in Δ for variables needed in Γ.

The substitution $\sigma : \Delta \longrightarrow \Gamma$ in a term t (resp. a type A) defined in a context Γ will be written $t[\sigma]$ (resp. $A[\sigma]$): the variables of t (resp. A) are substituted by their corresponding term in σ, so that $t[\sigma]$ (resp. $A[\sigma]$) is a term (resp. type) in Δ.

■ **Example 4.7** If $\Gamma \stackrel{\text{def}}{=} (n : \mathbb{N},\ p : \text{Prime},\ \mathbb{K} : \text{Field}\, p^n,\ V : \mathbb{K} - \text{VectorSpace})$ is the aforementioned context and $\Delta \stackrel{\text{def}}{=} (q : \text{PalindromicPrime}, \mathbb{L} : \text{TopologicalField}\, q^q,\ A : \mathbb{L} - \text{Algebra})$ is a context in which we have a palindromic prime number q, a topological field of order q^q, and an $\mathbb{L}$-algebra A.

The following would define a substitution from Δ to Γ:

$$\begin{aligned}
&\Delta \vdash q : \mathbb{N} \\
&\Delta \vdash q : \text{Prime} \\
&\Delta \vdash \mathbb{L} : \text{Field}\, q^q \\
&\Delta \vdash A : \mathbb{L} - \text{Algebra}
\end{aligned}$$

■

5. Categorical semantics of Type Theory

Several categorical framework can be used to define semantics of type theory. We are just about to give an algebraic notion of **model of type theory** *via* a particular kind of categories, known as **Categories with Families**[Cap17] (which will be abbreviated **CwF**).

Categories with Families

The idea behind Categories with Families is to regard contexts as forming a category whose arrows are substitutions/context morphisms. Types and terms are then seen as families over a context.

Definition 5.0.1 — a Category with Families (abbreviated CwF) is given by

- a **category** $\mathscr{C}$, with a terminal object $\bullet$ (the *empty context*)
- a **presheaf**

$$\mathrm{Ty} : \mathscr{C}^{\mathrm{op}} \longrightarrow \mathrm{Set}$$

- a **presheaf** from the category of elements of Ty:

$$\mathrm{Tm} : \left(\int_{\mathscr{C}} \mathrm{Ty} \right)^{\mathrm{op}} \longrightarrow \mathrm{Set}$$

- *Universal property of context extension*: for each $\Gamma : \mathscr{C}$, $A : \mathrm{Ty}(\Gamma)$, the following presheaf from the slice category $\mathscr{C}/\Gamma$

$$\begin{cases} (\mathscr{C}/\Gamma)^{\mathrm{op}} & \longrightarrow \mathrm{Set} \\ (\Delta, \sigma) & \longmapsto \mathrm{Tm}\left(\Delta, \mathrm{Ty}(\sigma)(A)\right) \end{cases}$$

  is **represented** by an object $(\Gamma.A, \pi_A)$

  NB $\Gamma.A$ is the *context extension* of Γ by the type A, and π_A is the *display map* of A (a projection from $\Gamma.A$ to Γ, which simply "forgets" about A).

For **type theory**, the interpretation will be the following:

CwF / Type theory correspondence

CwF	Type theory
objects $\Gamma, \Delta, \Theta, \dots$	**contexts** $\Delta \overset{\text{def}}{=} (x_1 : A_1, x_2 : A_2(x_1), x_3 : A_3(x_1, x_2), \dots)$ $\overset{\text{def}}{=} \bullet, A_1, A_2(x_1), A_3(x_1, x_2) \dots$ $\Gamma \overset{\text{def}}{=} (y_1 : B_1, y_2 : B_2(y_1), y_3 : B_3(y_1, y_2), \dots)$
morphisms $\sigma : \Delta \longrightarrow \Gamma,\ \theta : \Gamma \longrightarrow \Theta, \dots$	**context morphisms / substitutions** $\sigma \overset{\text{def}}{=} (t_1 : B_1, t_2 : B_1(t_1), t_3 : B_3(t_1, t_2), \dots)$
$\lvert \mathrm{Ty}(\mathscr{C}) \rvert$ $\mathrm{Ty}(\Delta), \mathrm{Ty}(\Gamma), \dots$	**types** over a given context $\{ \text{ types over } \Delta \} \overset{\text{def}}{=} \{A \mid \Delta \vdash A\},$ $\{ \text{ types over } \Gamma \} \overset{\text{def}}{=} \{B \mid \Gamma \vdash B\}, \dots$
$\mathrm{Hom}_{\mathrm{Ty}}$ $\mathrm{Ty}(\underbrace{\sigma}_{\in \mathrm{Hom}_{\mathscr{C}}(\Delta,\Gamma)}) : \mathrm{Ty}(\Gamma) \longrightarrow \mathrm{Ty}(\Delta), \dots$	**type substitutions** $\begin{cases} \{B \mid \Gamma \vdash B\} & \longrightarrow \{A \mid \Delta \vdash A\} \\ B & \longmapsto B[\sigma] \end{cases}$
$\left\lvert \mathrm{Tm}\left(\int_{\mathscr{C}} \mathrm{Ty}^{\mathrm{op}} \right) \right\rvert$ $\mathrm{Tm}(\Delta, A), \mathrm{Tm}(\Gamma, B), \dots$	**terms** of a given type over a given context $\{ \text{ terms of type } \Delta \text{ over } \Delta \} \overset{\text{def}}{=} \{t \mid \Delta \vdash t : A\},$ $\{ \text{ terms of type } \Gamma \text{ over } \Gamma \} \overset{\text{def}}{=} \{t \mid \Gamma \vdash t : B\}, \dots$
$\mathrm{Hom}_{\mathrm{Tm}}$ $\mathrm{Tm}(\underbrace{\sigma}_{\in \mathrm{Hom}_{\mathscr{C}}(\Delta,\Gamma) \text{ and } \mathrm{Ty}(\sigma)(B)=A}) : \mathrm{Tm}(\Gamma, B) \longrightarrow \mathrm{Tm}(\Delta, A), \dots$	**term substitutions** $\begin{cases} \{t \mid \Gamma \vdash t : B\} & \longrightarrow \{t \mid \Delta \vdash t : A\} \\ t & \longmapsto t[\sigma] \end{cases}$

Vocabulary 5.1 For all $\Delta, \Gamma : |\mathscr{C}|$, $A : \mathrm{Ty}(\Gamma)$, $a : \mathrm{Tm}(\Gamma, A)$, $\sigma : \Delta \longrightarrow \Gamma$, we will denote

- $\mathrm{Tm}(\Gamma, A)$ by $\mathrm{Tm}_\Gamma(A)$
- $\mathrm{Ty}(\sigma)(A)$ by $A[\sigma]$
- $\mathrm{Tm}_(\sigma, a)$ by $a[\sigma]$

Universal property of context extension

Given $\Gamma : |\mathscr{C}|$ and $A : \mathrm{Ty}(\Gamma)$, if we denote by ϕ the natural isomorphism between the two functors, the representation condition implies that the following diagram commutes:

$$\begin{array}{ccc} \mathrm{Tm}(\Delta, \mathrm{Ty}(\sigma)(A)) & \xleftarrow{\mathrm{Tm}(f)} & \mathrm{Tm}(\Delta, \mathrm{Ty}(\sigma)(A)) \\ \downarrow{\scriptstyle \phi_{\Delta,\sigma}} & & \downarrow{\scriptstyle \phi_{\Delta',\sigma'}} \\ \mathrm{Hom}_{\mathscr{C}/\Gamma}((\Delta,\sigma),(\Gamma.A,\pi_A)) & \xleftarrow[\_\circ f]{} & \mathrm{Hom}_{\mathscr{C}/\Gamma}((\Delta',\sigma'),(\Gamma.A,\pi_A)) \end{array}$$

for all

$$\begin{array}{ccc} \Delta & \xrightarrow{f} & \Delta' \\ & {\scriptstyle\sigma}\searrow \quad \swarrow{\scriptstyle\sigma'} & \\ & \Gamma & \end{array}$$

in $\mathscr{C}/\Gamma$.

For each context morphism $\sigma : \Delta \longrightarrow \Gamma$ and type $A : \mathrm{Ty}(\Gamma)$, there is a natural (in the variable (Δ, σ)) isomorphism:

$$\mathrm{Hom}_{\mathscr{C}/\Gamma}((\Delta,\sigma),(\Gamma.A,\pi_A)) \quad \simeq \quad \mathrm{Tm}(\Delta, \underbrace{\mathrm{Ty}(\sigma)(A)}_{A[\sigma]}) \tag{5.1}$$

$$\begin{array}{ccc} \Delta & \xrightarrow{f} & \Gamma.A \\ & {\scriptstyle\sigma}\searrow \quad \swarrow{\scriptstyle\pi_A} & \\ & \Gamma & \end{array}$$

That is, each term $a : \mathrm{Tm}_\Delta(A[\sigma])$ gives a corresponding morphism $C(\Delta, \Gamma.A)$ that we will denote by $\langle \sigma, a \rangle$.

Corollary 5.0.1 *Reciprocally*, if $f : \Delta \longrightarrow \Gamma.A$ is such that:

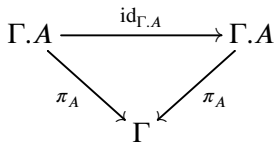
$$\begin{array}{ccc} \Delta & \xrightarrow{f} & \Gamma.A \\ & {\scriptstyle\sigma}\searrow \quad \swarrow{\scriptstyle\pi_A} & \\ & \Gamma & \end{array}$$

commutes and $a : \mathrm{Tm}_\Delta(A[\sigma])$ is the term associated with f via the isomorphism 5.1, then $f = \langle \pi_A, a \rangle$

Moreover, by taking $\Delta \overset{\text{def}}{=} \Gamma$ and $\sigma \overset{\text{def}}{=} \mathrm{id}$ in (5.1), it follows that $\mathrm{Tm}(\Gamma, A) \simeq \mathrm{Hom}_{\mathscr{C}/\Gamma}((\Gamma, \mathrm{id}), (\Gamma.A, \pi_A))$, which means that:

Proposition 5.0.2 Terms of type A over a context Γ can be seen as *sections* (right inverses) of the display map $\pi_A : \Gamma.A \longrightarrow \Gamma$, and reciprocally.

Definition 5.0.2 — In $\Gamma.A$, *the variable of type* A: $v_A : \mathrm{Tm}_{\Gamma.A}(A[\pi_A])$ is the term obtained by applying (5.1) to the identity morphism $\mathrm{id}_{\Gamma.A}$ in $\mathscr{C}/\Gamma$:



NB The genitive case "of type A" stems from the fact that the substitution along the display map $[\pi_A]$ can be left implicit, so that v_A can be seen as a term of type A in $\mathrm{Tm}_{\Gamma.A}(A)$.

Corollary 5.0.3 For all $\Gamma : |\mathscr{C}|$ and $A : \mathrm{Ty}(\Gamma)$:

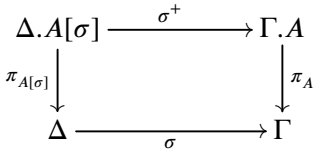
$$\langle \pi_A, v_A \rangle = \mathrm{id}_{\Gamma.A}$$

Proof. As $\mathrm{id}_{\Gamma.A}$ makes the triangle in 5.0.2 commute and $v_A : \mathrm{Tm}_{\Gamma.A}(A[\pi_A])$ is the term associated with $\mathrm{id}_{\Gamma.A}$ via the isomorphism 5.1, the result is obtained by the corollary 5.0.1. ■

Upon context extension $\Gamma.A$, another name (for example a) can be given to the variable $v_A : \mathrm{Tm}_{\Gamma.A}(A)$; if so, it will be explicitly denoted in this way: $\Gamma.(a : A)$.

Lastly, we need to express the ability to transport terms of type $A : \mathrm{Ty}(\Gamma)$ to terms of type $A[\sigma] : \mathrm{Ty}(\Delta)$ provided that $\sigma : \Delta \longrightarrow \Gamma$, via *morphism extension*.

Proposition 5.0.4 — Extended morphisms.
If $\sigma : \Delta \longrightarrow \Gamma$ and $A : \mathrm{Ty}(\Gamma)$, there exists a morphism $\sigma^+ : \Delta.A[\sigma] \longrightarrow \Gamma.A$ (referred to as σ *extended with* A) such that the following diagram:



is a pullback

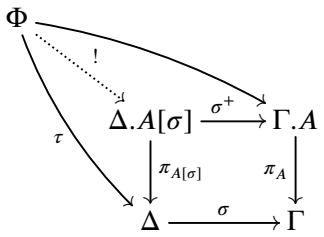
Proof.

- *Existence*: By (5.1), the existence of such a morphism σ^+ is equivalent to the set $\mathrm{Tm}_{\Delta.A[\sigma]}(A[\sigma \circ \pi_{A[\sigma]}]) = \mathrm{Tm}_{\Delta.A[\sigma]}(A[\sigma][\pi_{A[\sigma]}])$ (the functor Ty preserves composition) containing a term, which is clear since the variable of type $A[\sigma]$ is an element thereof.
- *Pullback*: the square is pullback if and only if for all contexts Φ and morphisms $\tau : \Phi \longrightarrow \Delta$, the following sets are isomorphic:

$$\left\{ f : \Phi \longrightarrow \Gamma.A \text{ such that } \begin{array}{ccc} \Phi & \xrightarrow{f} & \Gamma.A \\ {\scriptstyle\tau}\downarrow & & \downarrow{\scriptstyle\pi_A} \\ \Delta & \xrightarrow{\sigma} & \Gamma \end{array} \text{ commutes} \right\}$$

$$\simeq$$

$$\left\{ u : \Phi \longrightarrow \Delta.A[\sigma] \text{ such that } \begin{array}{ccc} \Phi & \xrightarrow{u} & \Delta.A[\sigma] \\ & {\scriptstyle\tau}\searrow \quad \swarrow{\scriptstyle\pi_{A[\sigma]}} & \\ & \Delta & \end{array} \text{ commutes} \right\}$$



It is the case, since the latter is $\mathscr{C}/\Delta\left((\Phi,\tau),(\Delta.A[\sigma],\pi_{A[\sigma]})\right)$, and the former is isomorphic to $\mathscr{C}/\Gamma\left((\Phi,\sigma\circ\tau),(\Gamma.A,\pi_A)\right)$, both of which are naturally isomorphic $\mathrm{Tm}_\Phi(A[\sigma\circ\tau]) = \mathrm{Tm}_\Phi(A[\sigma][\tau])$ by (5.1).

■

Thanks to this property, context extension constitutes a functor from $\int^{\mathscr{C}} \mathrm{Ty}$ to $\mathscr{C}$.

■ **Example 5.1** — Set **is a CwF.**
The category of sets Set is an important example of CwF that will be used later.

- *Contexts* are sets
- *Morphisms* are functions
- *Types* over Γ are families of sets indexed by Γ:

$$\mathrm{Ty}(\Gamma) \stackrel{\text{def}}{=} \Gamma \longrightarrow \mathrm{Set}$$

NB Intuitively, if Γ is the context "let n be an odd natural number" and $A : \mathrm{Ty}(\Gamma)$ the type Prime n ("n is a prime number"), then in the CwF Set, we will have

$$\Gamma \stackrel{\text{def}}{=} 2\mathbb{N}+1$$

and

$$A \stackrel{\text{def}}{=} \Gamma \ni n \mapsto \begin{cases} \{*\} & \text{if } n \text{ is prime} \\ \varnothing & \text{else} \end{cases}$$

- *Terms* of type A over Γ are defined as follows:

$$\begin{aligned}\mathrm{Tm}_\Gamma(A) &\overset{\text{def}}{\equiv} \left\{(a_\gamma)_{\gamma\in\Gamma} \mid \forall \gamma,\ a_\gamma \in A(\gamma)\right\} \\ &:\equiv \prod_{\gamma:\Gamma} A(\gamma) \qquad \text{(in Type Theory)}\end{aligned}$$

■

CwF-morphisms

Let $\mathscr{C}, \mathscr{D}$ be two CwFs.

Definition 5.0.3 — A (weak) CwF-morphism $\mathscr{C} \longrightarrow \mathscr{D}$ is given by:

- a functor $F : \mathscr{C} \longrightarrow \mathscr{D}$
- a natural transformation $\phi : \mathrm{Ty} \longrightarrow \mathrm{Ty} \circ F$
- a family of functions $\theta_{\Gamma,A} : \mathrm{Tm}(\Gamma, A) \longrightarrow \mathrm{Tm}(F(\Gamma), \phi_\Gamma(A))$ such that the following diagram commutes:

$$\begin{array}{ccc} \mathrm{Tm}(\Gamma, A) & \xrightarrow{\theta_{\Gamma,A}} & \mathrm{Tm}(F(\Gamma), \phi_\Gamma(A)) \\ \Big\downarrow{\scriptstyle \mathrm{Tm}(f)} & & \Big\downarrow{\scriptstyle \mathrm{Tm}(F(f))} \\ \mathrm{Tm}(\Delta, A[f]) & \xrightarrow[\theta_{\Delta,A[f]}]{} & \mathrm{Tm}(F(\Delta), \phi_\Gamma(A[f])) \end{array}$$

such that the CwF structure is preserved "up to isomorphism":

- $F(\bullet)$ is a terminal object in $\mathscr{D}$
- for all $\Gamma : |\mathscr{C}|,\ A : \mathrm{Ty}(\Gamma)$: the morphism $\vartheta : F(\Gamma.A) \longrightarrow F(\Gamma).\phi_\Gamma(A)$ obtained from $\theta_{\Gamma,A}(v_A)$ via the universal property of context extension is an isomorphism:

$$\begin{array}{ccc} F(\Gamma.A) & \xrightarrow{\vartheta} & F(\Gamma).\phi_\Gamma(A) \\ & {\scriptstyle F(\pi_A)}\searrow \quad \swarrow{\scriptstyle \pi_{\phi_\Gamma(A)}} & \\ & F(\Gamma) & \end{array}$$

Vocabulary 5.2 — A strict CwF-morphism $\mathscr{C} \longrightarrow \mathscr{D}$ is a weak CwF-morphism which *strictly* preserves the CwF structure: terminal objects and context extension are preserved "on the nose", and the map $\vartheta \overset{\text{def}}{\equiv} \mathrm{id}_{F(\Gamma.A)}$ for all $\Gamma : |\mathscr{C}|,\ A : \mathrm{Ty}(\Gamma)$.

Therefore, categories with families form a category CwF, whose objects are CwFs and whose morphisms are weak CwF-morphisms.

The category comprised of CwFs and strict CwF-morphisms is a *subcategory* of CwF.

6. Globular ω-groupoids

We are beginning to talk about the crux of the matter: here are ω-groupoids!

Comparison of structures and laws of different algebraic structures

	Monoids	Categories	Higher Categories	ω-Groupoids
Carrier set/Underlying set	Sets	Multigraphs (also called *quivers*)	Globular Sets	Globular Sets
Structure	Binary operation, Neutral element	Composition, Identity morphisms	Compositions, Identity morphisms	Compositions, Identity morphisms, Inverse elements
Laws	Associativity, Unit law	Associativity, Unit law	Associativity, Unit law, Interchange law	Associativity, Unit law, Interchange law, Inverse law

Globular sets

As shown in the above table, the underlying *carrier* of an ω-groupoid is what we call a **globular set**.

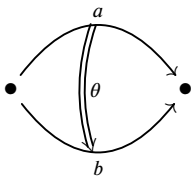
Definition 6.0.1 — A globular set is a family of sets $(G_n)_{n\in\mathbb{N}}$ and functions $s_n, t_n : G_n \longrightarrow G_{n-1}$ (which stand for *source* and *target* respectively) such that, for all $n \in \mathbb{N}$:

$$\begin{cases} s_{n-1} \circ s_n = s_{n-1} \circ t_n \\ t_{n-1} \circ s_n = t_{n-1} \circ t_n \end{cases} \tag{6.1}$$

NB To avoid clutter, the subscripts in s_n, t_n will be omitted.

What is the intuition behind (6.1)?

Rather simple, we want the sets to form kinds of *cells* (that's what is meant by the adjective *globular*) in this way:



That is, for instance, in the above example, where

- the points • are members of G_0
- $a, b \in G_1$
- $\theta \in G_2$
- $s(\theta) \stackrel{\text{def}}{=} a$
- $t(\theta) \stackrel{\text{def}}{=} b$

we have to make sure that:

$$\begin{cases} s(a) = s(b) \\ t(a) = t(b) \end{cases}$$

i.e.:

$$\begin{cases} s(s(\theta)) = s(t(\theta)) \\ t(s(\theta)) = t(t(\theta)) \end{cases}$$

As it happens, $s(\theta) = a$ and $t(\theta) = b$ are said to be **parallel**.

Thus, **??** enable us to ensure that all sources and their corresponding targets are parallel.

Besides, for all $n \in \mathbb{N}^*$, $a, b : G_{n-1}$, let us define:

$$G_n(a, b) \stackrel{\text{def}}{=} \{f \in G_n \mid s(f) = a \text{ and } t(f) = b\}$$

NB A globular set $(G_n)_{n\in\mathbb{N}}$ may also be defined inductively as follows:

- G_0 is a set
- for each $n \in \mathbb{N}$, for all $x, y \in G_n$, $G_{n+1}(x, y)$ is a globular set

Laws and Structure of ω-groupoids

The bottom line is that for usual mathematical objects such as monoids, the enriched layers over the carrier set are twofold:

1. one the one hand, you have everything that pertains to the **structure**:
   - for *monoids*: that encompasses the binary operation $\circ$ of the monoid
2. on the other hand, there are the **laws** (which are stated as logical formulas):
   - for *monoids*: they're comprised of
   - the *associativity law*:

$$\forall x, y, z.\ (x \circ y) \circ z = x \circ (y \circ z)$$

- the *unit law*:

$$\exists e, \forall x.\ x \circ e = e \circ x = x$$

For **strict** ω-groupoids, the pattern will remain the same: there are structures (compositions, identity elements, inverse elements) and laws (associativity, interchange law, identity law, inverse law).

But when it comes to **weak** ω-groupoids, *there is nothing but structure*. To be more precise, in **weak** ω-groupoids, the laws and structures with regard to elements of G_n, such as:

- [Structure] *Composition*:

$$x \xrightarrow{f} y \xrightarrow{g} z \quad \overset{\_\circ\_}{\Longrightarrow} \quad x \xrightarrow{g \circ f} z$$

- [Law] *Associativity*:

$$(h \circ g) \circ f \overset{\alpha}{\Longrightarrow} h \circ (g \circ f)$$

will now become elements of G_{n+1} (and thus pertain to the structure).

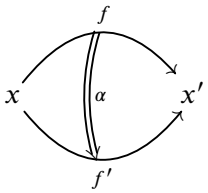
That will lead to a combinatorial explosion; if only for associativity: the number of ways of associating n applications of $\circ$ is the Catalan number $C_n \overset{\text{def}}{=} \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$

Strict ω-groupoids

Let us take a look at what the G_n ($n \in \mathbb{N}$) look like in Type Theory, if $G \overset{\text{def}}{=} \bigsqcup_{n \geq 0} G_n$ is a **strict** ω-groupoid.

Let $n \in \mathbb{N}^*$. The elements of G_n will be called ***n*-arrows**.

Definition 6.0.2 — Dimension skips: If $i > j > k \in \mathbb{N}$, $x, x' : G_k$, $f, f' : G_j(x, x')$ and $\alpha : G_i(f, f')$



then α **can be seen as** k**-arrow of** $G_k(x, x')$.

NB Indeed, since

$$\begin{cases} s \circ s = s \circ t \\ t \circ s = t \circ t \end{cases}$$

it follows that for all $n \in \mathbb{N}^*$ and $f_1, \ldots, f_n \in \{s, t\}$:

$$\begin{cases} s\ f_1\ \ldots\ f_n = s \\ t\ f_1\ \ldots\ f_n = t \end{cases}$$

which means that only the leftmost composition matters.

G_n: the structure

In G_n, we are given identity elements, inverse elements, and composition maps.

- *Identity*:

$$\mathrm{id}^n(\_) : \prod_{a : G_{n-1}} G_n(a, a)$$

- *Inverse*:

$$\mathrm{inv}^n(\_,\_) : \prod_{a,b : G_{n-1}} G_n(a, b) \longrightarrow G_n(b, a)$$

- *Vertical Composition* (going along the n-arrows):

$$;^n_n(\_,\_,\_) : \prod_{a,b,c : G_{n-1}} G_n(a, b) \times G_n(b, c) \longrightarrow G_n(a, c)$$

However, we cannot just settle for this. The only composition that was presented was the "obvious" one, but the thing is that there are n different types of compositions in G_n: $;^n_1, \ldots, ;^n_n$

Basically, if $1 \leq i \leq n$, the intuition is that the i-th composition "goes along the i-arrows".

Let $i \in \mathbb{N}$ such that $1 \leq i \leq n$. Remember that an n-arrow can be seen as an i-arrow, since $n \geq i$.

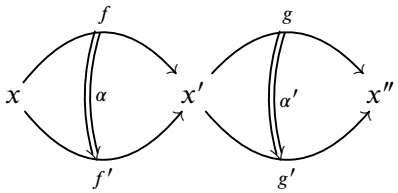
NB In the following diagrams, it will be assumed that
- simple arrows are members of G_i
- double arrows are members of G_n

Here we go:

- *Horizontal Composition* (it goes along the i-arrows):

$$;^n_i : \prod_{x,x',x'' : G_0} \prod_{\substack{f,f' : G_i(x,x') \\ g,g' : G_i(x',x'')}} G_n(f, f') \times G_n(g, g') \longrightarrow G_n(f \;;^i_i\; g,\; f' \;;^i_i\; g')$$

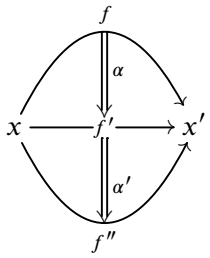
$$\frac{\alpha : G_i(x, x') \qquad \alpha' : G_i(x', x'')}{\alpha \;;^n_i\; \alpha' : G_i(x, x'')}$$



- *Vertical Composition* (it goes along the n-arrows):

$$;^n_n : \prod_{x,x' : G_0} \prod_{f,f',f'' : G_i(x,x')} G_n(f,f') \times G_n(f',f'') \longrightarrow G_n(f,f'')$$

$$\frac{\alpha : G_n(f,f') \qquad \alpha' : G_n(f',f'')}{\alpha ;^n_n \alpha' : G_n(f,f'')}$$

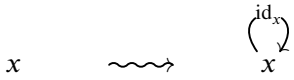


G_n: the laws

Now, things are becoming trickier, but fortunately G_2 *actually illustrates the general case*, as long as you keep in mind the following general patterns, in which we will perform *dimension skips*.

NB in the following, the diagrams formed by the squiggly arrows $\rightsquigarrow$ are to commute.

- **Identity** (i): If $x : G_i$, then we have

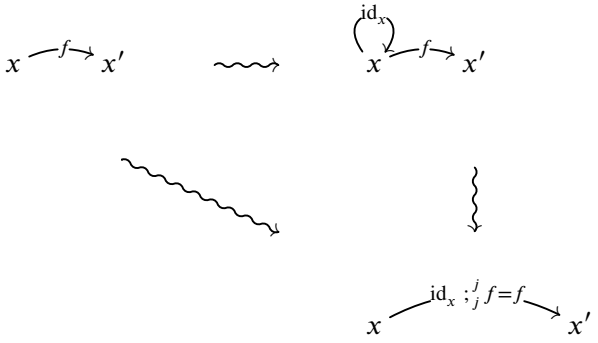


- **Composition** ($i < j$): Given that
  - the points $x, x', x'' : G_i$
  - the simple arrows $f, g : G_j$

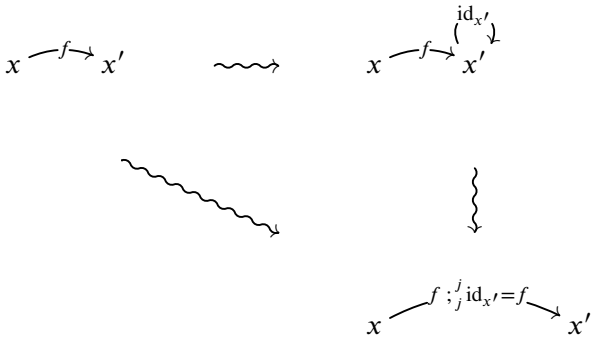
  then

- **Identity Laws** ($i < j$): Given that
  - the points $x, x' : G_i$
  - the simple arrow $f : G_j$

  then
  - *Left*:



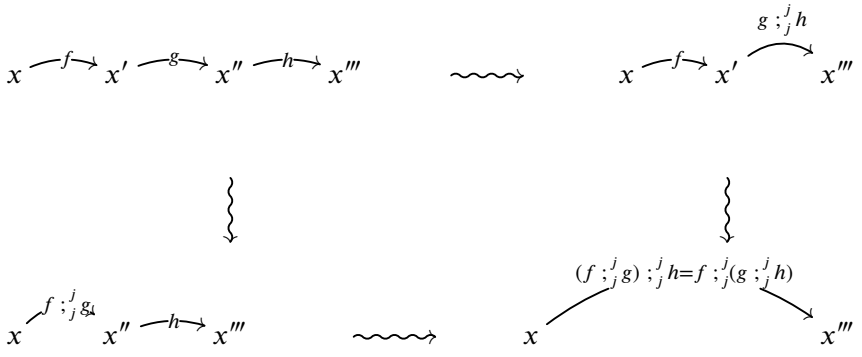
– *Right*:



- **Associativity law** ($i < j$): Given that

  – the points $x, x', x'', x''' : G_i$
  – the simple arrows $f, g, h : G_j$

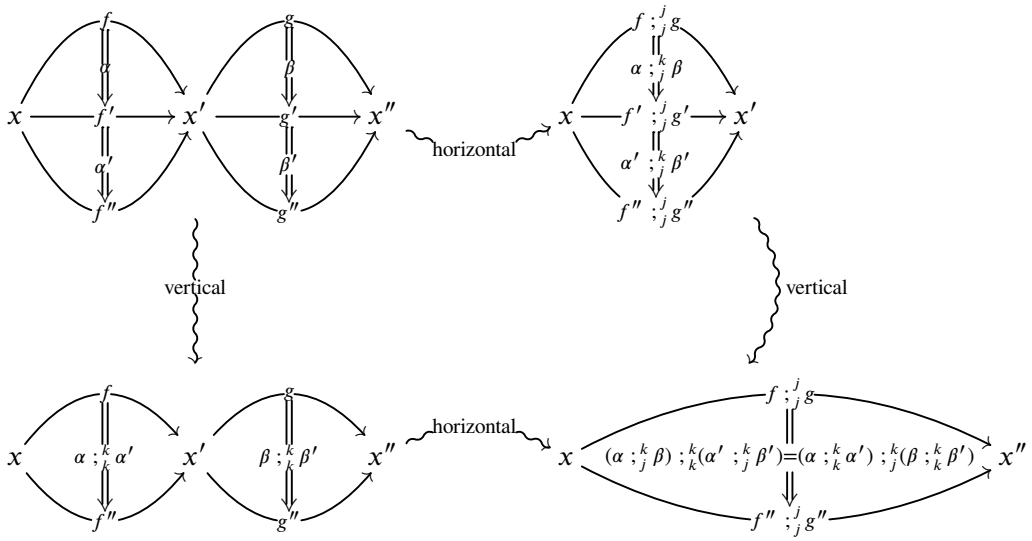
  then



- **Interchange law** ($i < j < k$): Given that

  – the points $x, x', x'' : G_i$
  – the simple arrows $f, f', f'', g, g', g'' : G_j$
  – the double arrows $\alpha, \alpha', \beta, \beta' : G_k$

  then



Weak ω-groupoids: handmade partial construction

Let us take a look at what G_1 and G_2 look like in Type Theory, if $G \stackrel{\text{def}}{=} \bigsqcup_{n\geq 0} G_n$ is a **weak** ω-groupoid: the laws of G_n now become elements of G_{n+1}, which themselves induce new *coherence laws* at level G_{n+2}, which themselves… and so on.

G_1

- *Identity*:

$$\mathrm{id}^1(\_) : \prod_{a:G_0} G_1(a,a)$$

- *Inverse*:

$$\mathrm{inv}^1(\_,\_) : \prod_{a,b:G_0} G_1(a,b) \longrightarrow G_1(b,a)$$

- *Composition*:

$$;^1_1(\_,\_,\_) : \prod_{a,b,c:G_0} G_1(a,b) \times G_1(b,c) \longrightarrow G_1(a,c)$$

G_2

- *Identity*:

$$\mathrm{id}^2(\_) : \prod_{f:G_1} G_2(f,f)$$

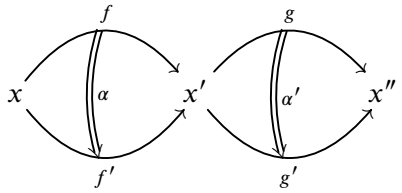
- *Inverse*:

$$\mathrm{inv}^2(\_,\_) : \prod_{f,g:G_1} G_2(f,g) \longrightarrow G_2(g,f)$$

- *Horizontal Composition* (along the 1-arrows):

$$;^2_1 : \prod_{x,x',x'':G_0\ f,f':G_1(x,x'),g,g':G_1(x',x'')} G_2(f,f') \times G_2(g,g') \longrightarrow G_2(f\ ;^1_1 g,\ f'\ ;^1_1 g')$$

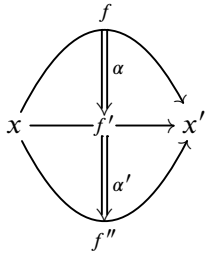
$$\frac{\alpha : G_1(x,x') \qquad \alpha' : G_1(x',x'')}{\alpha\ ;^2_1 \alpha' : G_1(x,x'')}$$



- *Vertical Composition* (along the 2-arrows):

$$;^2_2 : \prod_{x,x':G_0\ f,f',f'':G_1(x,x')} G_2(f,f') \times G_2(f',f'') \longrightarrow G_2(f,f'')$$

$$\frac{\alpha : G_2(f,f') \qquad \alpha' : G_2(f',f'')}{\alpha\ ;^2_2 \alpha' : G_2(f,f'')}$$



- *Associativity along the* 1*-arrows*:

$$\alpha^1_1(\_,\_,\_) : \prod_{x,x',x'',x''':G_0} \prod_{\substack{f:G_1(x,x') \\ g:G_1(x',x'') \\ h:G_1(x'',x''')}} G_2((f\ ;^1_1 g)\ ;^1_1 h,\ f\ ;^1_1 (g\ ;^1_1 h))$$

- *Left identity "law"*[1]:

$$\lambda^1(\_) : \prod_{f:G_1} G_2((\mathrm{id}^1_{sf}\ ;^1_1 f),\ f)$$

- *Right identity "law"*:

$$\rho^1(\_) : \prod_{f:G_1} G_2((f\ ;^1_1 \mathrm{id}^1_{tf}),\ f)$$

- *Left inverse "law"*:

$$l^1(\_) : \prod_{f:G_1} G_2((\mathrm{inv}^1_{sf,tf}(f)\ ;^1_1 f),\ \mathrm{id}^1_{tf})$$

- *Right inverse "law"*:

$$r^1(\_) : \prod_{f:G_1} G_2((f\ ;^1_1 \mathrm{inv}^1_{sf,tf}(f)),\ \mathrm{id}^1_{sf})$$

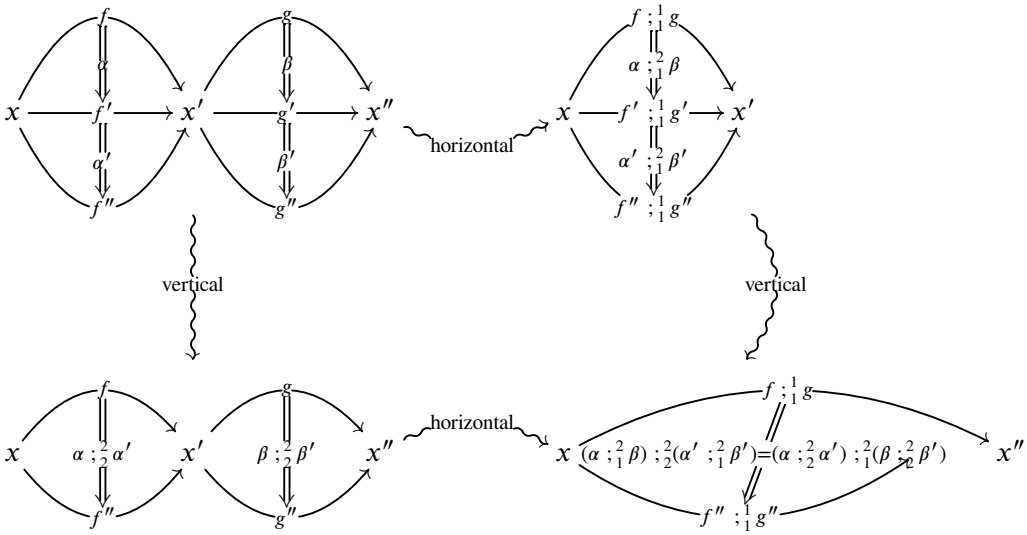
[1] this "law" has become structure

G_3, and so on

At level 3, in addition to that, we have to take into account

- the *interchange law* between the 2-arrows and the 1-arrows:

$$\alpha_2^1 : \prod_{x,x':G_0} \prod_{f,f',f'',f''':G_1(x,x')} \prod_{\substack{\alpha:G_2(f,f')\\ \beta:G_2(f',f'')\\ \gamma:G_2(f'',f''')}} G_3((\alpha \,;_2^2 \beta) \,;_2^2 \gamma,\ \alpha \,;_2^2 (\beta \,;_2^2 \gamma))$$



- the different compositions $;_1^3$, $;_2^3$, $;_3^3$
- the *associativity along the* 2*-arrows*:

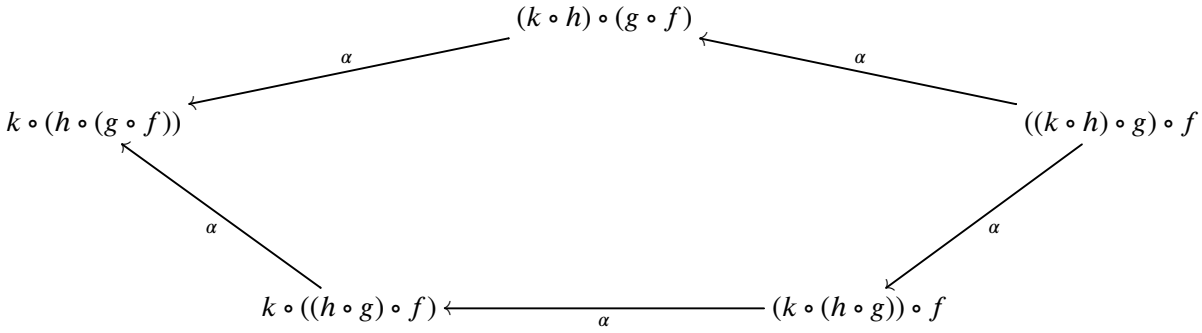
$$\alpha_2^1 : \prod_{x,x':G_0} \prod_{f,f',f'',f''':G_1(x,x')} \prod_{\substack{\alpha:G_2(f,f')\\ \beta:G_2(f',f'')\\ \gamma:G_2(f'',f''')}} G_3((\alpha \,;_2^2 \beta) \,;_2^2 \gamma,\ \alpha \,;_2^2 (\beta \,;_2^2 \gamma))$$

- etc...

Coherence laws in G_3:

Equations between elements of G_1 have been replaced by new 2-arrows. But we're not done yet, when it comes to these new 2-arrows: one also require that they satisfy some new equations (represented by new 3-arrows) of their own, called *coherence laws*!

■ **Example 6.1** For instance, there are 5 ways to parenthesize the composition of 4 1-arrows in G_1, which are related as follows:



We impose a coherence law - the *pentagon identity* - (which becomes a new 3-arrow), stating that this diagram commutes.

Analogously, another coherence law says that the diagram obtained by composing with the left identity then with the right identity and composing with the right identity then with the left one are the same thing commutes. ∎

Strict ω-groupoids: a categorical definition

Strict ω-groupoids can be rigorously with strict ∞-categories, about which a brief presentation *via* 2**-categories** and **globular sets** will be given.

It all boils down to 2-categories

Everything is said in title! But what are 2-categories anyway?

2-Categories

Definition 6.0.3 — a 2-category $\mathscr{C}$ is a structure comprised of:

- a class of objects:

$$|\mathscr{C}| : Set$$

- a function which associates a **category** - whose objects (resp. arrows) are called 1-*morphisms*/1-*cells* (resp. 2-*morphisms*/2-*cells*) - to each pair of objects (still noted $\mathscr{C}$ by abuse of notation):

$$\mathscr{C} : |\mathscr{C}| \longrightarrow |\mathscr{C}| \longrightarrow Cat$$

- a composition **functor**:

$$\circ' : \mathscr{C}(y,z) \times \mathscr{C}(x,y) \longrightarrow \mathscr{C}(x,z)$$

- an identity 1-morphism $\mathrm{id}_x : |\mathscr{C}(x,x)|$, for each object x:

$$\mathrm{id} : \prod_{x:|\mathscr{C}|} |\mathscr{C}(x,x)|$$

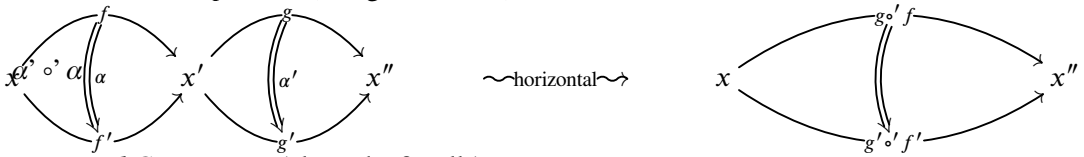
such that:

- for each object x, id_x, $\mathrm{id}_{\mathrm{id}_x}$ are identities of $\circ'$
- the following diagram commutes:

$$\begin{array}{ccc} \mathscr{C}(z,w)\times\mathscr{C}(y,z)\times\mathscr{C}(x,y) & \longrightarrow & \mathscr{C}(y,w)\times\mathscr{C}(x,y) \\ \downarrow & & \downarrow \\ \mathscr{C}(z,w)\times\mathscr{C}(x,z) & \longrightarrow & \mathscr{C}(x,w) \end{array}$$

As for ω-groupoids, we have a horizontal and a vertical compositions:

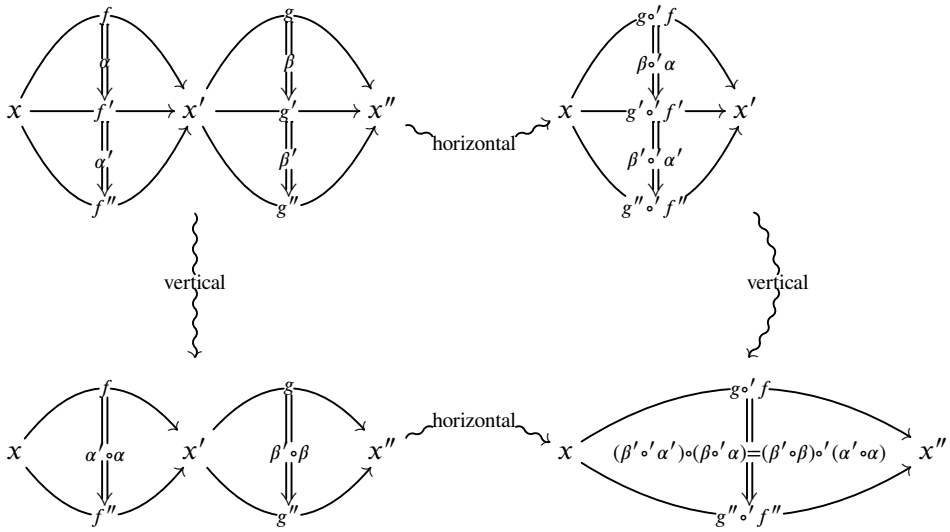
- *Horizontal Composition* (along the 1-cells):



- *Vertical Composition* (along the 2-cells):

An interchange law is given by the fact that $\circ'$ is a **functor**:

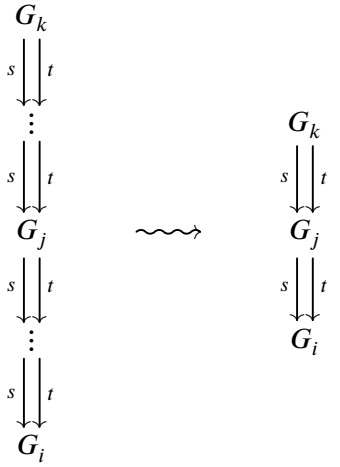
$$\begin{cases} \alpha : \mathscr{C}(x,x')(f,f') \\ \alpha' : \mathscr{C}(x,x')(f',f'') \\ \beta : \mathscr{C}(x',x'')(g,g') \\ \beta' : \mathscr{C}(x',x'')(g',g'') \end{cases} \Longrightarrow (\beta' \circ' \alpha') \circ (\beta \circ' \alpha) = (\beta' \circ \beta) \circ' (\alpha' \circ \alpha)$$



Strict ω-groupoids

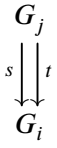
For a globular set G, recall that j-arrows could be seen as i-arrows, provided that $j > i$. So if $i < j < k$,

- G_k can be seen as a set of arrows whose sources and targets are in G_j
- G_j can be seen as a set of arrows whose sources and targets are in G_i



Definition 6.0.4 — a strict ∞-category $\mathscr{C}$ is a structure comprised of:

- a globular set $G \overset{\text{def}}{=} \bigsqcup_{n \geq 0} G_n$
- For all i, j with $i < j$, a structure of category on



such that for all $k > j$

$$\begin{array}{c} G_k \\ {\scriptstyle s}\Big\Downarrow{\scriptstyle t} \\ G_j \\ {\scriptstyle s}\Big\Downarrow{\scriptstyle t} \\ G_i \end{array}$$

forms a 2-category

Definition 6.0.5 — a strict ω-groupoid is a strict ∞-category in which all morphisms are invertible.

The problem is that for weak ω-groupoids, this categorical approach is not as successful: the weakness creates a mess (with laws becoming structure, coherence laws, etc...) which gets quickly out of control.

Brunerie Type Theory comes in

Now that you realize to what extent weak ω-groupoids are a mess, let us have a look at a very concise type theoretic way to define them, which is due to Guillaume Brunerie[Bru13].

Definition 6.0.6 — Brunerie type theory $\mathbb{B}$:
Brunerie type theory is given by these typing rules:

$$\frac{}{\bullet \vdash \star} \quad \text{(Base type)}$$

$$\frac{\Gamma \vdash x, y : A}{\Gamma \vdash x \simeq y} \quad \text{(Equality rule)}$$

$$\frac{\text{isContr}\,\Gamma \qquad \Gamma \vdash A}{\Gamma \vdash \mathbf{coh}^{\Gamma}_{A} : A} \quad \text{(Coherence rule)}$$

where isContr (which stands for *is contractible*) is inductively defined as follows:

Definition 6.0.7 — Contractibility
a context Γ is said to be *contractible* if and only if:

- $\Gamma = \bullet.\star$
- or: $\Gamma \overset{\text{def}}{=} \Delta.(y : A).(p : y \simeq x)$ where Δ is contractible and x is a term of type A in the context Δ

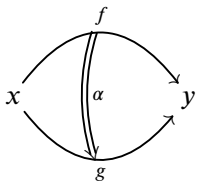
In other words:

$$\frac{\text{isContr}(\bullet.\star) \qquad \text{isContr}\,\Delta \qquad \Delta \vdash x : A}{\text{isContr}\,\Delta.(y : A).(p : x \simeq y)}$$

The intuition behind is that

- the elements of the base type $\star$ correspond to the elements of the set G_0 of the underlying globular set of the weak ω-groupoid.
- if $x, y : \star$ the elements of $x \simeq y$ are the 1-arrows of $G_1(x, y)$
- if $x, y : \star$ and $f, g : x \simeq y$, the elements of $f \simeq g$ are those of $G_1(f, g)$
- and so on…

Now, with this analogy in mind, a context $\Gamma \overset{\text{def}}{=} \bullet.(x\ y : \star).(f\ g : x \simeq y).(\alpha : f \simeq g)$ will be depicted as

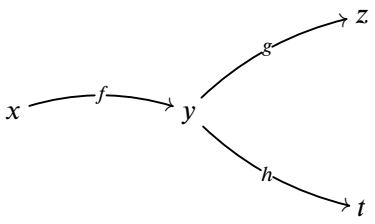


A context is **contractible** if it can be "reduced to a point", by contracting along the arrows.

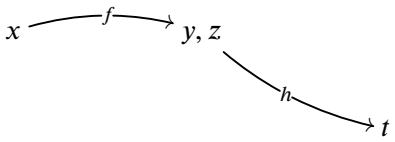
NB This analogous to the topological notion of contractibility!

■ **Example 6.2** For instance:

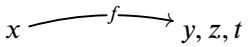
- the context $\Gamma \overset{\text{def}}{=} \bullet.(x\ y\ z\ t : \star).(f : x \simeq y).(g : y \simeq z).(h : y \simeq t)$ **is** contractible



Indeed, you can contract z and y along g:

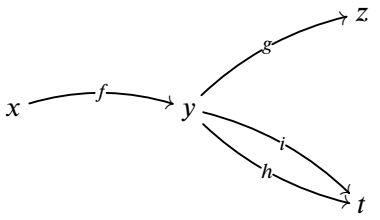


then t and y, z along h:

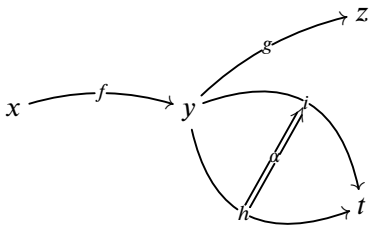


and finally x and y, z, t along f:

- but the context $\Gamma' \stackrel{\text{def}}{=} \bullet.(x\ y\ z\ t : \star).(f : x \simeq y).(g : y \simeq z).(h\ i : y \simeq t)$ is **not** contractible



because there is a *hole* between y and t (thus, we cannot contract y and t).
But if the hole is filled by a 2-arrow, like in $\Gamma'' \stackrel{\text{def}}{=} \bullet.(x\ y\ z\ t : \star).(f : x \simeq y).(g : y \simeq z).(h\ i : y \simeq t).(\alpha \simeq h\ i)$



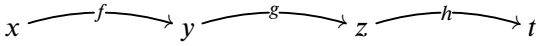
then it becomes contractible (i and h can be contracted, following which we proceed similarly to the previous example).

■

What about the laws (that become part of the structure, in ω-groupoids), such as the unit/associativity/interchange/coherence laws? They are taken care of by Brunerie's coherence rule, thanks to this notion of contractible contexts!

Let us illustrate this with an example: for instance, the *associativity law* (along the 1-arrows, to make things more simple (but the general case is quite similar)).

■ **Example 6.3** Let $x, y, z, t : \star,\ f : x \simeq y,\ g : y \simeq z,\ h : z \simeq t$ and $\Gamma_{x,y,z,t} \stackrel{\text{def}}{=} \bullet.(x\ y\ z\ t : \star).(f : x \simeq y).(g : y \simeq z).(h : z \simeq t)$



First, how to define composition along the 1-arrows?

As

- the context $\Gamma_{x,y,z,t}$ is *contractible*
- and $\Gamma_{x,y,z,t} \vdash x \simeq z$ (since $\Gamma_{x,y,z,t} \vdash x, z : \star$)

$f \,;^1_1 g : x \simeq z$ can be defined in $\Gamma_{x,y,z,t}$ as $\mathbf{coh}^{\Gamma_{x,y,z,t}}_{x \simeq z}$

Similarly, $(f \,;^1_1 g) \,;^1_1 h$ can be defined as $\mathbf{coh}^{\Gamma_{x,y,z,t}}_{x \simeq t}$, and we proceed analogously for $g \,;^1_1 h$ and $f \,;^1_1 (g \,;^1_1 h)$.

From now on, our aim is to show that there exists a term

$$\alpha^1(f, g, h) : (f \,;^1_1 g) \,;^1_1 h \simeq f \,;^1_1 (g \,;^1_1 h)$$

in $\Gamma_{x,y,z,t}$.

We are beginning to get it: we use the contractibility of $\Gamma_{x,y,z,t}$ again (and the fact that $\Gamma_{x,y,z,t} \vdash (f \,;^1_1 g) \,;^1_1 h \simeq f \,;^1_1 (g \,;^1_1 h)$)! ∎

NB Hence, Brunerie's coherence rule is very practical when it comes to expressing the unit/associativity/interchange/coherence laws!

But there's a catch (kind of): therewith comes a whole bunch of redundant terms, such as: $\mathbf{coh}^{\bullet, \star}_{\star}$, $\mathbf{coh}^{\bullet.(x:\star)}_{\mathbf{coh}^{\bullet,\star}_{\star} \simeq x}$, …

Right, so now we are beginning to get the feeling as to why Brunerie's type theory might be able to cover the notion of weak ω-groupoids. The rigorous definition of these is coming next.

7. Brunerie globular weak ω-groupoids

CwFs with additional structure

Categories with families (CwFs) only provide basic structure to interpret dependent type theories, but nothing whatsoever concerning type formers.

To interpret these, we will add some extra structure, by converting type theoretical inference rules into the language of CwFs.

Base Type

Definition 7.0.1 — a CwF has a base type $\star$ if there exists a type $\star : Ty(1)$.

Intensional Equality

Definition 7.0.2 — a CwF has (propositional) equality if for all $\Gamma : |\mathscr{C}|$, $A : \mathrm{Ty}(\Gamma)$, $x, y : \mathrm{Tm}_\Gamma(A)$, there exists a type $x \simeq y : \mathrm{Ty}(\Gamma)$ *stable under substitution*; i.e. for all $\sigma : \Delta \longrightarrow \Gamma$,

$$(x \simeq y)[\sigma] \equiv (x[\sigma] \simeq y[\sigma])$$

Definition 7.0.3 — a CwF has reflexivity if it has propositional equality and for all $\Gamma : |\mathscr{C}|$, $A : \mathrm{Ty}(\Gamma)$, $x : \mathrm{Tm}_\Gamma(A)$, there exists a term $\mathrm{refl}_x : \mathrm{Tm}_\Gamma(x \simeq x)$ *stable under substitution*; i.e. for all $\sigma : \Delta \longrightarrow \Gamma$

$$(\mathrm{refl}_x)[\sigma] \equiv \mathrm{refl}_{x[\sigma]}$$

Definition 7.0.4 — a CwF has J it has propositional equality and reflexivity, and if for all $\Gamma : |\mathscr{C}|$, $A : \mathrm{Ty}(\Gamma)$, $x : \mathrm{Tm}_\Gamma(A)$, $P(y,p) : \mathrm{Ty}(\Gamma.(y : A).(p : x \simeq y))$, if there exists a term $d : \mathrm{Tm}_\Gamma(P(x, \mathrm{refl}_x))$ then there exists a term $\mathrm{J}(y,p,d) : \mathrm{Tm}_{\Gamma.(y:A).(p:x\simeq_A y)}(P(y,p))$ *stable under substitution*; i.e. for all $\sigma : \Delta \longrightarrow \Gamma.(y : A).(p : x \simeq_A y)$

$$\mathrm{J}(y,p,d)[\sigma] \equiv \mathrm{J}\left(y[\pi_{(y:A).(x\simeq_A y)} \circ \sigma], p[\pi_{x\simeq_A y} \circ \sigma], d[\pi_{(y:A).(x\simeq_A y)} \circ \sigma]\right)$$

and such that

$$\mathrm{J}(x, \mathrm{refl}_x, d) \equiv d : \mathrm{Tm}_\Gamma(P(x, \mathrm{refl}_x))$$

CwFs with Identity and Base Type

Definition 7.0.5 — Base-typed identity type theory $\mathbb{S}^{\mathrm{Id}}_{\star}$:
Base-typed identity type theory is comprised of

- a base type rule:

$$\frac{}{\bullet \vdash \star} \quad \text{(Base type)}$$

- typing rules of intensional equality rules (introduction rule, reflexivity and the J eliminator)

Definition 7.0.6 — A base-typed identity CwF (abbreviated CwFId) is a CwF that has equality, reflexivity, J, and a base type $\star$.

NB such a CwF is a *categorical model* of base-typed identity type theory, in that all typing rules thereof can be interpreted in it.

Base-typed identity CwFs form a category CwFId whose objects are CwFIds and whose arrows are CwF-morphisms which preserve their additional structure (called CwFId-morphisms).

We denote by CwFId^s the subcategory of CwFId comprised of CwFIds and strict CwF-morphisms preserving their additional structure.

Brunerie CwFs

Definition 7.0.7 — Brunerie CwF is a CwF that has equality, a base type $\star$, and such that

- there exists a subset $\mathcal{C} \subseteq |\mathscr{C}|$ (the *contractible contexts*) such that
  - $\bullet.\star : \mathcal{C}$
  - if $\Delta : \mathcal{C},\ B : \mathrm{Ty}(\Delta),\ t : \mathrm{Tm}_{\Delta}(B)$ then

$$\Delta.(y : B).(p : y \simeq t) : \mathcal{C}$$

- *Coherence rule*: for all $\Gamma : \mathcal{C},\ A : \mathrm{Ty}(\Gamma)$, there exists a term $\mathbf{coh}^{A}_{\Gamma} : \mathrm{Tm}_{\Gamma}(A)$

NB As you may have guessed, such a CwF is a *categorical model* of Brunerie type theory.

Again, Brunerie CwFs form a category CwFB whose objects are Brunerie CwFs and arrows are CwF-morphisms which preserve their additional structure.

We denote by CwFB^s the subcategory of CwFB comprised of Brunerie CwFs and strict CwF-morphisms preserving their additional structure (called Brunerie CwF-morphisms).
Vocabulary 7.1 — a CwFB-subcategory of a Brunerie CwF $\mathscr{C}$ is a subcategory of $\mathscr{C}$ which is a Brunerie CwF with the same base type as $\mathscr{C}$ and whose types and terms sets are subsets of the corresponding ones in $\mathscr{C}$.

Syntaxes as initial objects

The next step is to show that there are initial objects in the categories CwFId^s and CwFB^s, which will come in handy later. We will only prove this result for CwFB^s, the proof for CwFId^s being analogous.

Theorem 7.0.1 There exists an initial object $\mathbb{B}$ (resp. $\mathbb{S}^{\mathrm{Id}}_{\star}$) in CwFB^s (resp. CwFId^s).

Proof. According to the theorem 3.0.3, it suffices to show that the constant functor $U : \mathrm{CwFB} \longrightarrow \mathbb{1}$ has a left adjoint (where $\mathbb{1}$ is the category with a single object).

We will resort to the left adjoint functor theorem 3.0.5, which in this case reads: if G preserves limits (which is trivial), CwFB is complete and if there exists a set $S \subseteq |\,\mathrm{CwFB}\,|$ such that for all $X : |\,\mathrm{CwFB}\,|$, there exists an object $X' \in S$ and a morphism from X' to X (solution set condition).

Lemma 7.0.2 CwFB^s is complete (has small limits).

Proof. Let I be small category and $D : I \longrightarrow \mathrm{CwFB}^s$ a diagram. Let $\mathscr{L}$ be the limit of D in Cat and ϕ be the natural transformation from $\Delta(\mathscr{L})$ to D (since we are in Cat, each Φ_i is a functor)

$\mathscr{L}$ can be equipped with a CwF structure:

- contexts are objects of $\mathscr{L}$
- for all $\Gamma : |\mathscr{L}|$, $\mathrm{Ty}(\Gamma)$ is defined to be the limit of the $\mathrm{Ty}(\Phi_i(\Gamma))$ for $i : |I|$, and ϕ the corresponding natural transformation.
- for all type A over Γ, $\mathrm{Tm}_\Gamma(A)$ is defined to be the limit of the $\mathrm{Tm}_{\Phi_i(\Gamma)}(\phi(A_i))$ for $i : |I|$, and θ the corresponding natural transformation.

As for context extension, equality, base type, and coherence type structures: they are defined pointwise (the strictness of the CwF-morphisms is used here, for context extension).

Lastly, the set $C_{\mathscr{L}}$ of contractible contexts is the smallest subset of $|\mathscr{L}|$ containing $\bullet.\star$ and such that: for all $\Delta : C_{\mathscr{L}}$, $B : \mathrm{Ty}(\Delta)$, $t : \mathrm{Tm}_\Delta(B)$, it follows that $\Delta.(y : B).(p : y \simeq t) : C_{\mathscr{L}}$.

■

Lemma 7.0.3 — Solution set condition.
There exists a weakly initial family in CwFB^s, that is, a set $S \subseteq |\,\mathrm{CwFB}^s\,|$ such that for all $X : |\,\mathrm{CwFB}^s\,|$, there exists an object $X' \in S$ and a morphism from X' to X.

Proof. Let us show that the set S of *countable* Brunerie CwFs is weakly initial CwFB^s, where a Brunerie CwF is said to be countable if it has a countable number of contexts, morphisms and all its type and term sets are countable.

Let $\mathscr{C}$ be a Brunerie CwF. We will construct an increasing family $(\mathscr{C}^n)_{n\geq 0}$ of subsets of $|\mathscr{C}|$ along with families of morphisms, types, terms and a set of contractible contexts, such that $\bigcup_n \mathscr{C}^n$ is a countable CwFB-subcategory of $\mathscr{C}$ (the result will then stand, due to the fact that there will be an obvious strict CwF-morphism from $\bigcup_n \mathbb{C}_n$ to $\mathscr{C}$).

For all $n \geq 0$, in $\mathscr{C}^n$: the contexts will be denoted by $|\mathscr{C}^n|$, the morphisms between Δ and Γ by $\mathscr{C}^n(\Delta, \Gamma)$, the set of types over Γ by $\mathrm{Ty}^n(\Gamma)$, the terms of type A over Γ by $\mathrm{Tm}^n_\Gamma(A)$, the contractible contexts C^n

Let $\mathscr{C}^0$ be empty, and for all $n \in \mathbb{N}$:

- $|\mathscr{C}^{n+1}| \stackrel{\text{def}}{=} \mathscr{C}^n \cup \{\bullet\} \cup \{\Gamma.A \mid \Gamma : \mathscr{C}^n \wedge A : \mathrm{Ty}^n(\Gamma)\}$
- morphisms of $\mathscr{C}^{n+1}$ are morphisms of $\mathscr{C}^n$ plus composition of morphisms in $\mathscr{C}^n$ and for all $\Gamma : |\mathscr{C}^n|$
  - the morphism from Γ to $\bullet$
  - id_Γ
  - for all $A : \mathrm{Ty}^n(\Gamma)$ the display map $\pi_A : \Gamma.A \longrightarrow \Gamma$
  - for all $\sigma : \mathscr{C}^n(\Delta, \Gamma)$, and $a : \mathrm{Tm}^n_\Delta(A[\sigma])$, $\langle \sigma, a \rangle$
- for all $\Gamma : |\mathscr{C}^n|$, $\quad \mathrm{Ty}^{n+1}(\Gamma) \stackrel{\text{def}}{=} \mathrm{Ty}^n(\Gamma) \cup \{\star\} \cup \left\{x \simeq y \mid A : \mathrm{Ty}^n(\Gamma) \wedge a, b : \mathrm{Tm}^n_\Gamma(A)\right\}$
- for all $\Gamma : |\mathscr{C}^n|$, $A : \mathrm{Ty}^n(\Gamma)$, $\quad \mathrm{Tm}^{n+1}_\Gamma(A) \stackrel{\text{def}}{=} \begin{cases} \mathrm{Tm}^n_\Gamma(A) \cup \left\{\mathbf{coh}^A_\Gamma\right\} & \text{if } \Gamma \in \mathcal{C}^n \\ \mathrm{Tm}^n_\Gamma(A) & \text{else} \end{cases}$
- $\mathcal{C}^{n+1} \stackrel{\text{def}}{=} \mathcal{C}^n \cup \{\bullet.\star\} \cup \left\{\Delta.(y : B).(p : y \simeq t) \mid \Delta : \mathcal{C}^n,\ B : \mathrm{Ty}^n(\Delta),\ t : \mathrm{Tm}^n_\Delta(B)\right\}$

The definition of Brunerie CwFs comprises operations of finite arity only, so that $\bigcup_n \mathscr{C}^n$ along with its structure is indeed a Brunerie CwF, and the result follows. ■

By the two preceding lemmas and the left adjoint functor theorem 3.0.5, there exists an initial object $\mathbb{B}$ in CwFB. ■

Similarly, we denote by $\mathbb{S}^{\mathrm{Id}}_\star$ the initial object in CwFId^s. These objects ($\mathbb{B}$ and $\mathbb{S}^{\mathrm{Id}}_\star$) are referred to as the *syntaxes* of their corresponding type theory.

Brunerie globular weak ω-groupoids

We can finally define Brunerie globular weak ω-groupoids!

Definition

Definition 7.0.8 — A Brunerie globular weak ω-groupoid is a weak CwF-morphism from $\mathbb{B}$ to Set

■ **Example 7.1 — Monoids as CwF-morphisms.**
Monoids can be seen as CwF-morphisms from $\mathbb{M}$ to Set, where $\mathbb{M}$ is the syntax of the following type theory:

$$\frac{}{\bullet \vdash \star} \quad \text{(Base type)}$$

$$\frac{\Gamma \vdash x, y : \star}{\Gamma \vdash x \cdot y} \quad \text{(Binary operation)}$$

$$\frac{}{\Gamma \vdash e : \star} \quad \text{(Neutral element)}$$

to which are added the associativity and unit laws. ■

Types are Brunerie globular weak ω-groupoids

Let $\mathbb{S}$ be a CwF with additional structure modelling a type theory with intensional identity, such as Martin Löf's type theory or homotopy type theory.

To show that types form Brunerie globular weak ω-groupoids, we will proceed by demonstrating that there exist CwF-morphisms

$$\mathbb{B} \xrightarrow{\Phi} \mathbb{S}^{\mathrm{Id}}_{\star} \xrightarrow{F_A} \mathbb{S} \xrightarrow{\mathbb{S}(\bullet,\_)} \mathrm{Set}$$

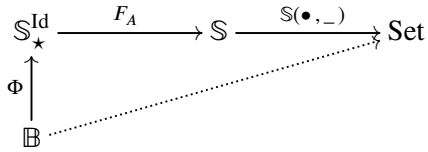
whose composition will constitute the Brunerie globular weak ω-groupoid associated with the terms of a given type A.

Theorem 7.0.4 — Types are Brunerie weak ω-groupoids.
Let $A : \mathrm{Ty}(\bullet)$ be a type in $\mathbb{S}$.

$$\mathbb{S}(\bullet, \bullet.A) \text{ forms a Brunerie weak } \omega\text{-groupoid}$$

NB As $\bullet$ is terminal, $\mathbb{S}(\bullet, \bullet.A)$ are sections of the display map (since there is only one morphism from $\bullet$ to $\bullet$: $\mathrm{id}_\bullet$), which can be seen as terms of type A over $\bullet$, by the proposition 5.0.2.

Proof. It suffices to show that we have:



where Φ, F_A, and $\mathbb{S}(\bullet, \_)$ are CwF-morphisms such that

- F_A is a CwF-morphism mapping the base type $\star : \mathrm{Ty}(\bullet)$ to $A : \mathrm{Ty}(\bullet)$.
- The hom-functor $\mathbb{S}(\bullet, \_)$ is viewed as the CwF-morphism:
  - $\mathbb{S}(\bullet, \_) : \mathbb{S} \longrightarrow \mathrm{Set}$ (where Set is regarded as a CwF)
  - for all $\Gamma : |\mathbb{S}|$,

$$\phi_\Gamma \overset{\text{def}}{\equiv} \begin{cases} \mathrm{Ty}(\Gamma) & \longrightarrow \mathrm{Ty}(\mathbb{S}(\bullet, \Gamma)) \\ A & \longmapsto \big(\mathbb{S}(\bullet, \Gamma) \ni \sigma \longmapsto \mathrm{Tm}_\bullet(A[\sigma])\big) \end{cases}$$

  - for all $\Gamma : |\mathbb{S}|$, $A : \mathrm{Ty}(\Gamma)$,

$$\theta_{\Gamma,A} \overset{\text{def}}{\equiv} \begin{cases} \mathrm{Tm}_\Gamma(A) & \longrightarrow \mathrm{Tm}_{\mathbb{S}(\bullet,\Gamma)}\big(\sigma \longmapsto \mathrm{Tm}_\bullet(A[\sigma])\big) \\ a & \longmapsto \big(\mathbb{S}(\bullet, \Gamma) \ni \sigma \longmapsto \underbrace{a[\sigma]}_{:\mathrm{Tm}_\bullet(A[\sigma])}\big) \end{cases}$$

What remains to be proved is the existence of a CwF-morphism $\Phi : \mathbb{B} \longrightarrow \mathbb{S}^{\mathrm{Id}}_{\star}$.

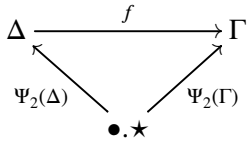
Existence of Φ:

The idea is to show that $\mathbb{S}^{\mathrm{Id}}_{\star}$ is a model of Brunerie type theory (that is, Brunerie typing rules can be in $\mathbb{S}^{\mathrm{Id}}_{\star}$), so that $\mathbb{S}^{\mathrm{Id}}_{\star} : |\,\mathrm{CwFB}\,|$ and by initiality of $\mathbb{B}$ in CwFB^s, there exists a Brunerie CwF-morphism Φ (which is in particular a CwF-morphism) from $\mathbb{B}$ to $\mathbb{S}^{\mathrm{Id}}_{\star}$.
But before that, we will need the following lemma:

Lemma 7.0.5 — Initiality of $\bullet.\star$.
The context $\bullet.\star$ (referred to as *the base-typed context*) is initial in $\mathbb{S}^{\mathrm{Id}}_{\star}$

Proof. The proof goes as follows:

1. firstly, we show that the co-slice category $(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_{\star}$ is a model of base-typed identity type theory; i.e. the base type and the rules of intensional equality (introduction rule, reflexivity and the J eliminator) can be interpreted therein (demonstration below).
2. Then, by initiality of $\mathbb{S}^{\mathrm{Id}}_{\star}$ in CwFId^s, there exists a strict (everything is preserved "on the nose") CwFId-morphism Ψ from $\mathbb{S}^{\mathrm{Id}}_{\star} \longrightarrow (\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_{\star}$.
3. Next, by composing Ψ with the forgetful CwFId-morphism $U : (\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_{\star} \longrightarrow \mathbb{S}^{\mathrm{Id}}_{\star}$ (whose action on objects is taking the first component, and which acts as the identity on morphisms), we get a CwFId-morphism $U \circ \Psi : \mathbb{S}^{\mathrm{Id}}_{\star} \longrightarrow \mathbb{S}^{\mathrm{Id}}_{\star}$ which is bound to be the identity by uniqueness of $\mathrm{id}_{\mathbb{S}^{\mathrm{Id}}_{\star}}$ (since $\mathbb{S}^{\mathrm{Id}}_{\star}$ is initial in CwFId).
4. It follows that for any $\Gamma : |\mathbb{S}^{\mathrm{Id}}_{\star}|$, $\Psi(\Gamma) \overset{\mathrm{def}}{=} (\Psi_1(\Gamma), \Psi_2(\Gamma))$, where $\Psi_1(\Gamma) = \Gamma$ (since, by composing with the forgetful functor afterwards, one obtains the identity) and $\Psi_2(\Gamma) : \bullet.\star \longrightarrow \Gamma$ (so $\bullet.\star$ is *weakly initial*).
   Besides, for any $f : \Delta \longrightarrow \Gamma$, $\Psi(f)$ has to be f (still because the identity is obtained by composing with U), so that any diagram:



   commutes.
   Therefore, with the notations used in 5.0.3: for any $f : \bullet.\star \longrightarrow \Gamma$, it follows (with $\Delta \overset{\mathrm{def}}{=} \bullet.\star$) that

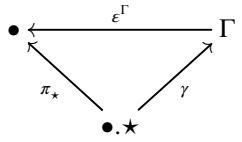
$$\Psi_2(\Gamma) = f \circ \underbrace{\Psi_2(\bullet.\star)}_{= \mathrm{id}_{\bullet.\star} \text{ as } \Psi(\bullet.\star) = \Psi(\bullet).\phi_{\Gamma}(\star) = (\bullet,\pi_\star).(\star,\upsilon_\star) = (\bullet.\star,\langle\pi_\star,\upsilon_\star\rangle)} = f \qquad \text{(see below)}$$

   As a result, $\bullet.\star$ is initial in $\mathbb{S}^{\mathrm{Id}}_{\star}$.

Proof that $(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_{\star}$ is a model of base-typed identity type theory:

– $(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_{\star}$ is a CwF such that:
  * the empty context is $(\bullet, \pi_\star)$, where $\pi_\star : \bullet.\star \longrightarrow \bullet$

    NB indeed, for all context $(\Gamma, \gamma) : |(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_{\star}|$, there exists a unique map $\varepsilon^{\Gamma} : \Gamma \longrightarrow \bullet$ ($\bullet$ is terminal), and the following diagram commutes:



since $\pi_\star : \bullet.\star \longrightarrow \bullet$ is the only morphism from $\bullet.\star$ to $\bullet$ (for $\bullet$ is terminal)

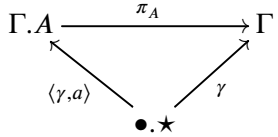
* for all $(\Gamma, \gamma) : |(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_\star|$ (so $\gamma : \bullet.\star \longrightarrow \Gamma$),

$$\mathrm{Ty}((\Gamma,\gamma)) \stackrel{\text{def}}{=} \left\{(A,a) \mid A : \mathrm{Ty}(\Gamma) \text{ and } a : \mathrm{Tm}_{\bullet.\star}(A[\gamma])\right\}$$

* for all $(\Gamma, \gamma) : |(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_\star|$, $(A, a) : \mathrm{Ty}((\Gamma, \gamma))$,

$$\mathrm{Tm}_{(\Gamma,\gamma)}((A,a)) \stackrel{\text{def}}{=} \left\{t \mid t : \mathrm{Tm}_\Gamma(A) \text{ and } t[\gamma] = a\right\}$$

* *Universal property of context extension*: if $(\Gamma, \gamma) : |(\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_\star|$, $(A, a) : \mathrm{Ty}((\Gamma, \sigma))$ (where $a : \mathrm{Tm}_{\bullet.\star}(A[\gamma])$, the object $\big(\big(\Gamma.A, \langle\gamma, a\rangle\big), \pi_A\big)$



represents the functor

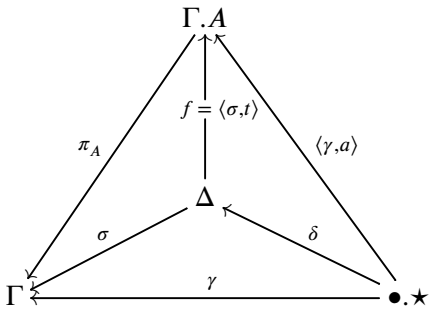
$$\begin{cases} \big((\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_\star\big)\big/(\Gamma,\gamma) & \longrightarrow \mathrm{Set} \\ \big(\big(\Delta,\delta\big),\sigma\big) & \longmapsto \mathrm{Tm}_{(\Delta,\delta)}(\underbrace{(A,a)[\sigma]}_{=(A[\sigma],a) \text{ since } A[\gamma]=A[\sigma\circ\delta]=A[\sigma][\delta]}) \end{cases}$$

Let us show that for all $\big(\big(\Delta,\delta\big),\sigma\big) : \Big|\big((\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_\star\big)\big/(\Gamma,\gamma)\Big|$,

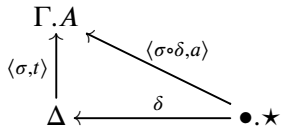
$$\mathrm{Hom}_{\left((\bullet.\star)\backslash\mathbb{S}^{\mathrm{Id}}_\star\right)\big/(\Gamma,\gamma)}\Big(\ \big(\big(\Delta,\delta\big),\sigma\big)\,,\big(\big(\Gamma.A,\langle\gamma,a\rangle\big),\pi_A\big)\ \Big) \simeq \mathrm{Tm}_{(\Delta,\delta)}\left((A[\sigma],a)\right)$$

Let f be a morphism in the hom-set. f is in particular a morphism from Δ to Γ, so by the universal property of context extension in $\mathbb{S}^{\mathrm{Id}}_\star$, there exists a term $t : \mathrm{Tm}_\Delta(A[\sigma])$ such that

$$f = \langle\sigma, t\rangle : \Delta \longrightarrow \Gamma.A$$



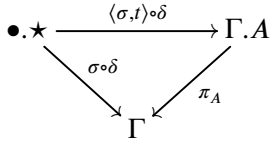
We associate to f the term $t : \mathrm{Tm}_{(\Delta,\delta)}\left((A[\sigma], a)\right)$. Indeed, let us show that $t[\sigma] = a$. Due to the fact that $\gamma = \sigma \circ \delta$ and that



commutes:

$$\langle \sigma \circ \delta, a \rangle = \langle \sigma, t \rangle \circ \delta$$

Besides, as



commutes, by the corollary 5.0.1:

$$\langle \sigma, t \rangle \circ \delta = \langle \sigma \circ \delta, t[\sigma] \rangle$$

It follows that

$$\langle \sigma \circ \delta, a \rangle = \langle \sigma \circ \delta, t[\sigma] \rangle$$

hence (owing to the isomorphism 5.1)

$$a = t[\sigma]$$

- the base type is interpreted as $(\star, v_\star)$ (which makes sense, since $\star : \mathrm{Ty}(\bullet)$ and $v_\star : \mathrm{Tm}_{\bullet.\star}(\star[\pi_\star])$)
- The rules related to intensional equality become:
  - *Formation*:
    $$\frac{(\Gamma, \gamma) \vdash x, y : (A, a)}{(\Gamma, \gamma) \vdash (x \simeq y, \mathrm{refl}_a)}$$
    since $\mathrm{refl}_a : \mathrm{Tm}_{\bullet.\star}(\underbrace{(x \simeq y)[\gamma]}_{\equiv (x[\gamma] \simeq y[\gamma]) \equiv (a \simeq a)})$
  - *Introduction*:
    $$\frac{(\Gamma, \gamma) \vdash x : (A, a)}{(\Gamma, \gamma) \vdash \mathrm{refl}_x : (x \simeq x, \mathrm{refl}_a)}$$
    since $(\mathrm{refl}_x)[\gamma] \equiv \mathrm{refl}_{x[\gamma]} \equiv \mathrm{refl}_a$
  - *Elimination*:
    $$\frac{\begin{array}{l}(\Gamma, \gamma) \vdash x : (A, a) \\ \Big(\Gamma.(y : (A, a)).(p : (x \simeq y, \mathrm{refl}_a)), \langle \gamma, \overbrace{a}^{:\mathrm{Ty}_\Gamma(A[\gamma])}, \overbrace{\mathrm{refl}_a[\pi_{(A,a)}]}^{:\mathrm{Ty}_{\Gamma.(y:A)}(\overbrace{(x \simeq y)[\gamma]}^{\equiv a \simeq a})} \rangle\Big) \vdash \big(P(y,p), p'(y,p)\big) \\ (\Gamma, \gamma) \vdash d : (P(x, \mathrm{refl}_x), p'(x, \mathrm{refl}_x))\end{array}}{\Big(\Gamma.(y : (A, a)).(p : (x \simeq y, \mathrm{refl}_a)), \langle \gamma, a, \mathrm{refl}_a \rangle\Big) \vdash \mathrm{J}(y, p, d) : \big(P(y,p), p'(y,p)\big)}$$

■

$\mathbb{S}^{\mathrm{Id}}_\star$ *is a model of Brunerie type theory*:

With the previous lemma, we are almost done: as the base type and the equality type can be trivially interpreted in $\mathbb{S}^{\mathrm{Id}}_\star$, all that remains to be done is to interpret Brunerie's coherence law in $\mathbb{S}^{\mathrm{Id}}_\star$.

$$\frac{\mathrm{isContr}\,\Delta \qquad \Delta \vdash A}{\Delta \vdash \mathbf{coh}^{\Delta}_{A} : A} \quad \text{(Coherence rule)}$$

where

$$\mathrm{isContr}(\bullet.\star)$$
$$\frac{\mathrm{isContr}\,\Delta \qquad \Delta \vdash x : A}{\mathrm{isContr}\,\Delta.(y : A).(p : x \simeq y)}$$

We proceed by structural induction on the contractible context Δ:

- **Base case** ($\Delta \overset{\text{def}}{\equiv} \bullet.\star$): if $A : \mathrm{Ty}(\bullet.\star)$, then by initiality of $\bullet.\star$, there exists a (unique) morphism $s : \bullet.\star \longrightarrow \bullet.\star.A$, to which can be associated a term of type A over $\bullet.\star$, by the proposition 5.0.2, because

$$\pi_A \circ s = \mathrm{id}_{\bullet.\star} \qquad \text{(where } \pi_A : \bullet.\star.A \longrightarrow \bullet.\star \text{ is the display map)}$$

  since the only morphism in $\mathrm{Hom}_{\mathbb{S}^{\mathrm{Id}}_{\star}}(\bullet.\star, \bullet.\star)$ is the identity ($\bullet.\star$ is initial).

- **Inductive step**: if Δ is contractible, $x : \mathrm{Tm}_{\Delta}(A)$ and $P(y,p) : \mathrm{Ty}\left(\Delta.(y : A).(p : x \simeq y)\right)$, let us show that there exists a term of type $P(y,p)$ over $\Delta.(y : A).(p : x \simeq y)$. By induction,

$$\mathbf{coh}^{\Delta}_{P(x,\mathrm{refl}_x)} : \mathrm{Tm}_{\Delta}\left(P(x,\mathrm{refl}_x)\right)$$

  so the J eliminator provides us with such a term

$$\mathrm{J}\left(y, p, \mathbf{coh}^{\Delta}_{P(x,\mathrm{refl}_x)}\right) : \mathrm{Tm}_{\Delta.(y:A).(p:x\simeq y)}\left(P(y,p)\right)$$

As a consequence: for example, the type $x \simeq y$ over the context $\bullet.(x, y : \star)$ in $\mathbb{B}$ is sent:

- first to the type $x \simeq y$ over $\bullet.(x, y : \star)$ in $\mathbb{S}^{\mathrm{Id}}_{\star}$ by Φ
- which is then sent to the type $x \simeq y$ over $\bullet.(x, y : A)$ by F_A
- which finally sent to the type

$$\begin{cases} \mathbb{S}(\bullet, A)^2 & \longrightarrow \mathrm{Set} \\ (t_1, t_2) & \longmapsto \mathrm{Tm}_{\bullet}(t_1 \simeq t_2) \end{cases}$$

■

So the CwF-morphism from $\mathbb{B}$ to Set sends $x \simeq y$ over $\bullet.(x, y : \star)$ to a function which takes two terms of type A over $\bullet$ (two points a, b in G_0 from the ω-groupoid point of view) and returns the set of terms of type $a \simeq b$ in $\bullet$ (the set $G_1(a,b)$ of the ω-groupoid), which corresponds to the intuition.

8. Agda Implementation

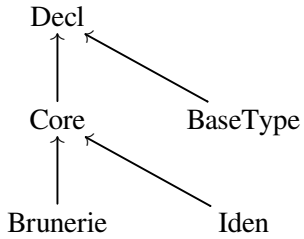
Agda is a dependently typed functional programming language and interactive proof assistant that we use in order to formalize what has been done on paper within type theory.

NB For example projects in Agda carried out during this internship, such as the pigeonhole principle, look at this repository: `https://github.com/youqad/Type-Theory`

Everything is not implemented from scratch: we use the "Type Theory in Type Theory"[AK16] framework developed by Thorsten Altenkirch and Ambrus Kaposi (`https://bitbucket.org/akaposi/tt-in-tt/src`), which features categories with families.

In a nutshell:

- all that is related to the syntax is placed inside the `tt-in-tt/TT` folder. What concerns us are the files related to these ones (with their dependence):



- all that is related to the models/semantics is placed at the root `tt-in-tt/`.

■ **Example 8.1** the folder `StandardModel` pertains to the CwF Set. ■

CwFs are defined in `TT/Core.agda` and `TT/Decl.agda`. The additional structures of CwFs are implemented in separate files, which are gathered appropriately when using CwFs with additional structures:

- `TT/Iden.agda` is intensional equality (equality, refl and J)
- `TT/BaseType.agda` is the base type rule
- `TT/Brunerie.agda` is Brunerie type theory

For the record, `Brunerie.agda` contains the following rules:

```
record Brunerie {i j k}{d : Decl}(c : Core {i}{j} d) :
                                                Set (i ⊔ j ⊔ lsuc k) where
  open Decl d
  open Core c

  field
    ⋆ : Ty •

    _≃_ : ∀{Γ} {A} → (a : Tm Γ A) → (b : Tm Γ A) → Ty Γ
```

```
    ≃[] : ∀{Γ Θ A}{u v : Tm Γ A}{σ : Tms Θ Γ} → u ≃ v [ σ ]T
                                              ≡ (u [ σ ]t) ≃ (v [ σ ]t)

    isContr     : Con → Set k

    base-contr  : isContr (• , ⋆)

    rec-contr   : ∀ {Γ} {A} → {t : Tm Γ A}
                            → isContr Γ
                            → isContr (Γ , A , (t [ wk ]t) ≃ vz )

    coherence   : ∀{Δ Γ} → {A : Ty Δ} {δ : Tms Γ Δ}
                         → isContr Δ → Tm Γ (A [ δ ]T)

    coherence[] : ∀{Δ Γ Θ} → {A : Ty Δ} {δ : Tms Γ Δ}
                             {p : isContr Δ} {σ : Tms Θ Γ}
                           → (coherence {A = A} {δ = δ} p)
                              [ σ ]t ≡[ TmΓ= [][]T ]≡
                                     coherence {δ = δ ∘ σ} p
```

So

- `TT/Decl.agda` + `TT/Core.agda` + `TT/Brunerie.agda` defines Brunerie type theory, whose syntax (initial object) is $\mathbb{B}$
- `TT/Decl.agda` + `TT/Core.agda` + `TT/BaseType.agda` + `TT/Iden.agda` defines based-type identity type theory, whose syntax is $\mathbb{S}^{\mathrm{Id}}_{\star}$

Speaking of syntax: in this framework, we just *postulate* (take it as an axiom) the existence of a syntax (initial object), for a given type theory (no left adjoint functor theorem!).

These are located in the subfolder corresponding to the type theory.

■ **Example 8.2** For example, the syntax $\mathbb{B}$ of Brunerie type theory is implemented in `TT/Brunerie/Syntax.agda`, as follows:

```
module TT.Brunerie.Syntax where

open import TT.Decl.Syntax public
open import TT.Core.Syntax public
open import TT.Brunerie
open import Agda.Primitive

postulate
  syntaxBrunerie : Brunerie {k = lzero} syntaxCore

open Brunerie syntaxBrunerie public
```

■

Besides, each of these subfolders contains a file `Rec.agda` which is the **recursor** for the type theory, which enable us to create *models* of this type theory.

Finally, CwF-morphism are defined in `TT/Decl/Morphism.agda` and `TT/Core/Morphism.agda`, so that a Brunerie weak ω-groupoid can be defined as follows:

```
module TT.Brunerie.WeakOmegaGroupoid {i} where

open import Agda.Primitive
open import lib

open import TT.Decl.Morphism
open import TT.Core.Morphism
open import TT.Brunerie
open import TT.Brunerie.Syntax public
open import StandardModel.Decl
open import StandardModel.Core


-- Brunerie weak ω-groupoids


weak-ωGroupoid = Σ (Decl syntaxDecl (d {i}))
                   (λ d → Core syntaxCore (c {i}) d)
```

Conclusion

On the whole, we have seen that a definition of weak ω-groupoids could not be as "straightforward" as for strict ω-groupoids, let alone a definition which could be internalized within Type Theory.

Then, we have defined Brunerie globular weak ω-groupoids as being the CwF-morphisms from $\mathbb{B}$ (the syntax of Brunerie type theory) to Set (seen as a CwF), before showing that

> In a type theory $\mathbb{S}$ with intensional equality, types form Brunerie globular weak ω-groupoids.

Such a result is a first step towards getting rid of the univalence axiom, if it can successfully be formalized within type theory, which has been started in Agda.

Bibliography

Front cover background image: https://clairejones.deviantart.com/art/Chaos-64871619

Articles

[AK16] Thorsten Altenkirch and Ambrus Kaposi. 'Type theory in type theory using quotient inductive types'. In: 51.1 (2016), pages 18–29 (cited on page 60).

[Bru13] Guillaume Brunerie. 'Syntactic grothendieck weak?-groupoids'. In: *Unpublished note, available at uf-ias-2012. wikispaces. com* (2013), page 191 (cited on page 47).

[Cap17] Paolo Capriotti. 'Models of Type Theory with Strict Equality'. In: *arXiv preprint arXiv:1702.04912* (2017) (cited on page 29).

Books

[AT10] Samson Abramsky and Nikos Tzevelekos. *Introduction to categories and categorical logic*. Springer, 2010, pages 3–94 (cited on page 12).

[Bou07] N. Bourbaki. *Théorie des ensembles*. Bourbaki, Nicolas. Springer Berlin Heidelberg, 2007. ISBN: 9783540340355. URL: https://books.google.fr/books?id=VDGifaOQogcC (cited on page 7).

[Lum10] Peter LeFanu Lumsdaine. *Higher categories from type theories*. Carnegie Mellon University, 2010 (cited on page 5).

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013 (cited on page 9).

Appendix

Boolean Type: $\mathbf{2}$

It comprises two elements: $0_{\mathbf{2}}, 1_{\mathbf{2}} : \mathbf{2}$

- *Introduction*: We can derive

$$x \mapsto \begin{cases} c_0 \text{ if } x = 0_{\mathbf{2}} \\ c_1 \text{ else if } x = 1_{\mathbf{2}} \end{cases} : \mathbf{2} \longrightarrow C$$

  provided that $C : \mathscr{U}$ and $c_0, c_1 : C$
- *Recursor*: It acts as an "if-then-else" statement:

$$\mathrm{rec}_{\mathbf{2}} :\equiv C, c_0, c_1, x \mapsto \begin{cases} c_0 \text{ if } x = 0_{\mathbf{2}} \\ c_1 \text{ else if } x = 1_{\mathbf{2}} \end{cases} : \prod_{C:\mathscr{U}} C \longrightarrow C \longrightarrow \mathbf{2} \longrightarrow C$$

- *Eliminator*:

$$\mathrm{ind}_{\mathbf{2}} :\equiv C, c_0, c_1, x \mapsto \begin{cases} c_0 \text{ if } x = 0_{\mathbf{2}} \\ c_1 \text{ else if } x = 1_{\mathbf{2}} \end{cases} : \prod_{C:\mathbf{2}\longrightarrow\mathscr{U}} C(0_{\mathbf{2}}) \longrightarrow C(1_{\mathbf{2}}) \longrightarrow \prod_{x:\mathbf{2}} C(x)$$

Product Type: $A \times B$

- *Introduction*: If $A, B : \mathscr{U}$:

$$A \times B :\equiv \prod_{x:\mathbf{2}} \mathrm{rec}_{\mathbf{2}}(\mathscr{U}, A, B, x)$$

- *Pairs*

$$(a, b) :\equiv \mathrm{ind}_{\mathbf{2}}(\mathrm{rec}_{\mathbf{2}}(\mathscr{U}, A, B, \_), a, b, \_)$$

- *Projections*

$$\pi_1 :\equiv p \mapsto p(0_{\mathbf{2}}) \qquad \pi_2 :\equiv p \mapsto p(1_{\mathbf{2}})$$

NB we could define the product type from scratch this way:

$$\mathrm{rec}_{A\times B} :\equiv C, f, (a,b) \mapsto f\ a\ b : \prod_{C:\mathcal{U}} (A \longrightarrow B \longrightarrow C) \longrightarrow A \times B \longrightarrow C$$

$$\mathrm{ind}_{A\times B} :\equiv C, f, (a,b) \mapsto f\ a\ b : \prod_{C:A\times B\longrightarrow\mathcal{U}} \left(\prod_{x:A}\prod_{y:B} C((x,y))\right) \longrightarrow \prod_{p:A\times B} C(p)$$

$$\pi_1 :\equiv \mathrm{rec}_{A\times B}(A, \lambda a, b.a, \_)$$

$$\pi_2 :\equiv \mathrm{rec}_{A\times B}(B, \lambda a, b.b, \_)$$

Σ-Types / Dependent pair types

Let's suppose that $A : \mathcal{U}$, $B : A \longrightarrow \mathcal{U}$, $a : A$, $b : B(a)$

Then we can derive

- *Introduction*:

$$(a,b) : \underbrace{\sum_{x:A} B(x)}_{\text{also written as } \sum x : A, B(x)}$$

- *Recursor*:

$$\mathrm{rec}_{\sum_{x:A} B(x)} :\equiv C, g, (a,b) \mapsto g\ a\ b : \prod_{C:\mathcal{U}} \left(\prod_{x:A} B(x) \longrightarrow C\right) \longrightarrow \left(\sum_{x:A} B(x)\right) \longrightarrow C$$

- *Eliminator*:

$$\mathrm{ind}_{\sum_{x:A} B(x)} :\equiv C, g, (a,b) \mapsto g\ a\ b : \prod_{C:\sum_{x:A} B(x)\longrightarrow\mathcal{U}} \left(\prod_{a:A}\prod_{b:B(a)} C((a,b))\right) \longrightarrow \prod_{p:\sum_{x:A} B(x)} C(p)$$

Product Type $A \times B$

If B is constant:

$$A \times B \equiv \sum_{x:A} B$$

Projections:

$$\pi_1 :\equiv (a,b) \mapsto a : \left(\sum_{x:A} B\right) \longrightarrow A \qquad \pi_2 :\equiv (a,b) \mapsto b : \prod_{p:\sum_{x:A} B(x)} B(\pi_1(p))$$

Coproduct Type / Disjoint union: $A + B$

- *Introduction*: If $A, B : \mathcal{U}$:

$$A + B :\equiv \sum_{x:\mathbf{2}} \mathrm{rec}_{\mathbf{2}}(\mathcal{U}, A, B, x)$$

- Left and Right Injections:

$$\mathrm{inl} :\equiv a \mapsto (0_{\mathbf{2}}, a) \qquad \mathrm{inr} :\equiv b \mapsto (1_{\mathbf{2}}, b)$$

Categorical origin

That is all fine and dandy, but how are we supposed to construct those constructors, generally speaking? And where do they even come from?

When Category Theory enters the scene

In order to understand where the constructors come from, we will stick to one example: the *product type*.

Product Type: $A \times B$

Let us assume $A, B : \mathcal{U}$ are two fixed types, in what follows. Let $\mathscr{C}_{A,B}$ be the category whose

- **Objects** are of the form:

$$(C, f), \text{ where } C : \mathcal{U} \text{ and } f : A \longrightarrow B \longrightarrow C$$

- **Morphisms/Arrows** are:

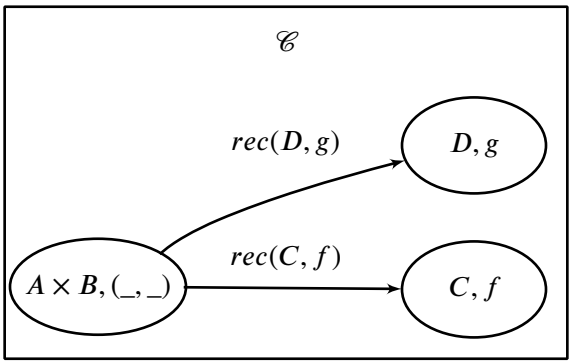
$$\mathscr{C}_{A,B} \overset{\text{def}}{=} \left\{ h : C \longrightarrow D \mid \text{ for all } a : A, b : B, \quad h(f\ a\ b) = g\ a\ b \right\}$$

Recursor

Assumption 8.0.1 $\mathscr{C}_{A,B}$ has an **initial object** $(A \times B, (\_, \_))$, i.e.: for all object (C, f), there **exists** a **unique** morphism in $\mathscr{C}_{A,B}\Big((A \times B, (\_, \_)),\ (C, f)\Big)$

Let us call this unique morphism, by any chance:

$$\mathrm{rec}_{A\times B}(C, f, \_) : \mathscr{C}_{A,B}\Big((A \times B, (\_, \_)),\ (C, f)\Big)$$



That is, for all $a : A,\ b : B$:

$$\mathrm{rec}_{A\times B}(C, f, (a, b)) \equiv f\ a\ b : C$$

which is the **computation/β-rule**!

NB In the general case, the recursor is required to be **weakly initial**, i.e. there's no constraint of uniqueness. But in the case of product type, it happens to be **initial** (we have the uniqueness).

In the particular case of product type: the **uniqueness principle/η-rule** is enforced by the *uniqueness hypothesis*:

$$\begin{aligned}\forall a : A,\ b : B,\ \mathrm{rec}_{A\times B}(C, f, (a, b)) &\equiv g((a, b)) \\ \Longrightarrow \mathrm{rec}_{A\times B}(C, f, \_) &\equiv g\end{aligned}$$

By the way:

$$\mathrm{rec}_{A\times B} : \prod_{C:\mathscr{U}} (A \longrightarrow B \longrightarrow C) \longrightarrow A \times B \longrightarrow C$$

which is exactly what we were after!

Induction

Preliminaries: fibrations

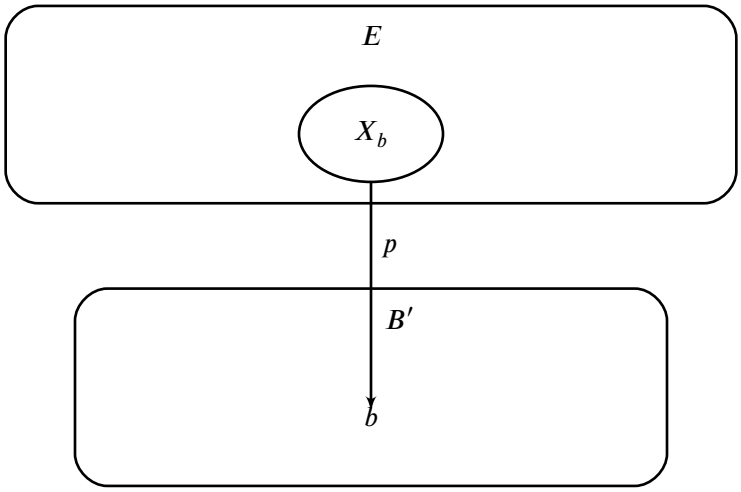
Fibrations of types

Before we adress the issue of eliminators, we shall dwell on the concept of **fibrations**.

Definition 8.0.1 — A fibration of sets it is just a function $p : E \longrightarrow B'$, usually represented (vertically) as

What will particularly get our attention is what we call **fibers**:

Definition 8.0.2 — Fiber X_b A subset of $X_b \subseteq E$ which is an inverse image of a point $b \in B'$



So all the fibers form a family $X : B' \longrightarrow Set$

NB But upon considering the family, the key point is that, instead of talking of the class of all sets, all the information lies in two simple sets and a function.

And **reciprocally**, if we are given a family $(X_b)_{b \in B'}$ of disjoint sets, a fibration of sets can be retrieved, with $E \overset{\text{def}}{=} \bigsqcup_{b \in B'} X_b$ and $p \overset{\text{def}}{=} E \longrightarrow B', X_b \ni x \mapsto b$

$$\begin{array}{c} E \overset{\text{def}}{=} \bigsqcup X_b \\ \Big\downarrow p \\ B' \end{array}$$

Fibrations of types

Back to types. This time, let's suppose we're given a family $X : B' \longrightarrow \mathcal{U}$

Similarly, with $E :\equiv \sum_{b:B'} X_b$ and $p :\equiv (b, e) \mapsto b : E \longrightarrow B'$

then

$$\begin{array}{c} E \\ \Big\downarrow p \\ B' \end{array}$$

is a fibration associated with X

Fibration in $\mathscr{C}_{A,B}$

$$\begin{array}{c}(E,f)\\ \big\downarrow{\scriptstyle \pi}\\ (B',g)\end{array}$$

is a **fibration** in $\mathscr{C}_{A,B}$ *iff* $\pi : \mathscr{C}_{A,B}\Big(\big(E,f\big),\big(B',g\big)\Big)$ and

$$\begin{array}{c}E\\ \big\downarrow{\scriptstyle \pi}\\ B'\end{array}$$

is a fibration

Towards the eliminator

Definition 8.0.3 — A section $s : \mathscr{C}_{A,B}\Big(\big(D,g\big),\big(C,f\big)\Big)$ **of** $\pi : \mathscr{C}_{A,B}\Big(\big(C,f\big),\big(D,g\big)\Big)$ it is a **right inverse** of π, i.e. $\pi \circ s = id_{(C,f)}$

In what follows, let $C : A \times B \longrightarrow \mathscr{U}$ and $E :\equiv \sum_{p:A\times B} C(p)$

From now on, let us assume that

Assumption 8.0.2 for each fibration $\pi : \mathscr{C}_{A,B}\Big(\big(E,f\big),\big(A\times B,(\_,\_)\big)\Big)$ associated with C, there exists a section $s : \mathscr{C}_{A,B}\Big(\big(A\times B,(\_,\_)\big),\big(E,f\big)\Big)$ of π

$$\begin{array}{c}(E,f)\\ \big\downarrow{\scriptstyle \pi}\big\uparrow{\scriptstyle s}\\ (A\times B,(\_,\_))\end{array}$$

NB with the notations of the section **Fibration of types**, we have now $B' \overset{\text{def}}{\equiv} A \times B$, and $X \overset{\text{def}}{\equiv} C$.

Let $\pi : \mathscr{C}_{A,B}\Big(\big(E,f\big),\big(A\times B,(\_,\_)\big)\Big)$ be a fibration associated with C.

Hence:

- *π is a fibration associated with C*:

  $$\text{for all } \; p : A\times B,\; c : C(p), \quad \pi((p,c)) :\equiv p$$

- *π is a morphism*:

  $$\pi(f\ a\ b) \equiv (a,b)$$

Let $s \overset{\text{def}}{\equiv} (s_1, s_2)$, then:

- $\underbrace{\pi} \circ s \equiv id_{A\times B}$:

first projection

$$s_1 \equiv id_{A\times B}$$

- *s is a morphism*:

$$s((a,b)) \equiv f\ a\ b :\equiv (f_1\ a\ b,\ f_2\ a\ b) : E$$

Thus, for all $a : A,\ b : B$:

$$s((a,\ b)) \equiv \Big((a,b)\ ,\ \underbrace{f_2\ a\ b}_{\equiv\ \underbrace{s_2}_{:\prod_{p:A\times B} C(p)}\ ((a,b)) : C((a,b))}\Big) : E$$

As s_2 depends only on $C : \mathcal{U}$ and $f_2 : \prod_{x:A}\prod_{y:B} C((x,y))$, it can be written as:

$$\mathrm{ind}_{A\times B}(C,\ f_2,\ \_) : \prod_{p:A\times B} C(p)$$

As a result:

Proposition 8.0.3

$$\mathrm{ind}_{A\times B} : \prod_{C:A\times B\to\mathcal{U}} \Big(\prod_{x:A}\prod_{y:B} C((x,y))\Big) \to \prod_{p:A\times B} C(p)$$

and

$$\mathrm{ind}_{A\times B}(C, f_2, (a,b)) = f_2\ a\ b$$

which is exactly what we wanted!

The same goes for the reverse: if you assume that you are given the eliminator, you can likewise show that each fibration $\pi : \mathscr{C}_{A,B}\Big(\big(E,f\big), \big(A\times B, (\_,\_)\big)\Big)$ associated with C has a section $s : \mathscr{C}_{A,B}\Big(\big(A\times B, (\_,\_)\big), \big(E,f\big)\Big)$.