A Study in Univalence

Anders Husebø
Master thesis in topology

Department of Mathematics
University of Bergen
Norway
June 2, 2014

1 Acknowledgments

First of all, I would like to thank my supervisor, professor Bjørn Ian Dundas, for all the excellent help and motivation in finishing this thesis. Also for passing on his viewpoint that everything within mathematics is interesting and worth studying. Further I would like to thank the entire Topology group at UiB which has become quite big these latter years and whose vibrant community has made studying a pleasure in stead of a chore. I would also like to thank professor Marc Aloysius Bezem for all his help in understanding the informatics side of homotopy type theory, and for the excellent seminars he has organized and given this last year.

Lastly I have to give a humongous thanks to all the friends I have made at the university. Both the people who started at the same time as me and have survived all the way to the end, and all the great people who have later joined our group and made every Pi-happy, party and many-hours coffee break the best time of my life.

2 Preface

In this master thesis we want to study the newly discovered homotopy type theory, and its models within mathematics. The main motivation is the article [KLV12] which gives a model for homotopy type theory with a univalent universe in simplicial set, a well studied category which also "models" the homotopy theory of topological spaces. Since the whole subject of homotopy type theory is recent, we also give an introduction to the field; though we will not give many of the beautiful results which can be found throughout the textbook [Uni13] which was released in 2013.

The main focus of this thesis is to understand the connection between homotopy type theory and the structure on categories which are models, so as to construct new models. Unfortunately since the subject is such a new one, there is not a lot of literature and well known methods, for instance a lot of the methods we study can be found in the 2012 (updated in 2013 and 2014) preprints [Shu12], [KLV12] and 2013 preprint [Shu13]. Combined with some pretty technical requirements, for instance in dealing with the coherence issue and the resulting way to construct a universe/object classifier, has proven this endeavor to be difficult and no complete new model has been found. We have however identified some concrete problems and possibilities that allow some categories to be models.

Contents

3 Category theory

The definition and facts in this chapter can be found in the first four chapters of [ML98].

Definition 3.1. *A category C consists of the following:*

Objects *a proper class $obj(C)$ of objects*

Morphisms *for each pair $x, y \in obj(C)$ of objects, a set $C(x, y)$ of morphisms from x to y*

Composition *for each triple $x, y, z \in obj(C)$ a function*

$$\begin{aligned} \circ : C(y,z) \times C(x,y) &\to C(x,z) \ , \\ (g,f) &\mapsto g \circ f =: gf. \end{aligned}$$

Identity *for each object $x \in obj(C)$ a morphism $id_x \in C(x,x)$ called the identity at x:*

Associativity *for any $x, y, z, w \in obj(C)$ and any $f \in C(x,y), g \in C(y,z), h \in C(z,w)$*

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

Left and right identity *for any $x, y \in obj(C)$ and any $f \in C(x,y)$*

$$id_y \circ f = f = f \circ id_x$$

There are some size issues to consider when using proper classes, so we will call a category small if the class of objects is a set. We will frequently use the notation $c \in C$ to denote $c \in obj(C)$ for C a category, and if $f \in C(x,y)$, we will use notations like $f : x \to y$ or $x \xrightarrow{f} y$.

We will use many examples of categories, most of which should be pretty familiar to the reader. In all the examples the composition, and identity should be straight forward.

Example 3.2. Probably the easiest example, and which we have already mentioned above, is the category **Set** of sets, with objects all sets, and morphisms all functions between sets.

Example 3.3. Mon, of all monoids and monoid maps.

Example 3.4. Grp, of all groups with group homomorphisms.

Example 3.5. FinSet, of all finite sets and functions between these.

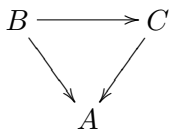
Example 3.6. Finally the category Δ of all finite ordinals $[n] = \{0 < 1 < 2 < \ldots < n\}$ for all $n \geq 0$ and all order-preserving morphisms. Perhaps an odd choice of example, but a category which will be very important to us.

There are also some ways of translating set-theoretic notions to categories. For instance we call a morphism $f : x \to y$ in a category $\mathbf{C}$ an isomorphism if there exists another morphism $g : y \to x$, such that $gf = id_x$ and $fg = id_y$; in this case we also call the objects x and y isomorphic and write $x \cong y$. Any set can be viewed as a category by taking its elements to be the objects, and the only morphisms are the identity. Any monoid can be viewed as a category with only one object and one morphism for each element in the monoid. Composition is then given by multiplication of the elements in the monoid. Now, just like groups are monoids with inverses in set theory, we can regard a group as a category with one object, and where all the morphisms are isomorphisms.

Given any category there are a few way to construct other categories from them.

Example 3.7. For any category $\mathbf{C}$, we can form the opposite category $\mathbf{C}^{op}$ with the same objects, but all arrows reversed.

Example 3.8. For any category $\mathbf{C}$, and any object $A \in \mathbf{C}$, we can form the over category $\mathbf{C}/A$ with objects morphisms of the form $B \to A$ for any $B \in \mathbf{C}$. Morphisms in $\mathbf{C}/A$ between $B \to A$ and $C \to A$ are commutative diagrams in $\mathbf{C}$ of the form



Just like in group theory we want a function between groups that preserve the group structure, we want a function between categories that preserve the category structure. Such a function is called a functor:

Definition 3.9. *A functor $F : C \to D$ between two categories is a map sending each object $c \in C$ to an object $F(x) \in D$ and each morphism $f : x \to y$ in C to a morphism $F(f) : F(x) \to F(y)$ in D, such that F preserves composition*

$$F(g \circ f) = F(g) \circ F(f),$$

whenever the left side makes sense, and for any $x \in C$, F preserves identity

$$F(id_x) = id_{F(x)}.$$

With the viewpoint above of a group as a category, we see that any group homomorphism gives a functor between two groups that are viewed as categories. Also with this definition we can form the category **Cat** of all small categories and functors between them as morphisms. Many interesting examples arise as functors between categories, so it would be nice to have a "functor of functors", so lets see what that amounts to.

Definition 3.10. *A natural transformation $\alpha : F \to G$, between two functors $F, G : C \to D$, is a function which assigns to each $x \in C$ a morphism $\alpha_x : F(x) \to G(x)$ in D such that for any morphism $f : x \to y$ in C we get a commutative diagram in D:*

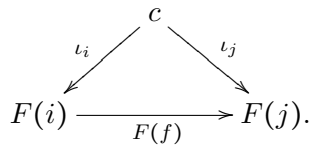
$$\begin{array}{ccc} F(x) & \xrightarrow{F(f)} & F(y) \\ {\scriptstyle \alpha_x}\downarrow & & \downarrow{\scriptstyle \alpha_y} \\ G(x) & \xrightarrow[G(f)]{} & G(y) \end{array}.$$

Or written as an equation: $G(f) \circ \alpha_x = \alpha_y \circ F(f)$, for any $x, y \in C$ and $f : x \to y$ in C.

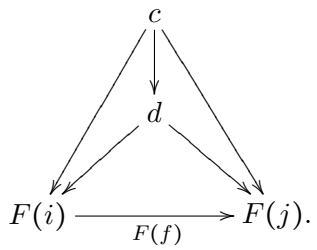
Composition of natural transformations is given in the obvious way, so that all functors between two fixed categories form a category with natural transformations as morphisms. There is especially one type of functor category we will be interested in, namely the simplicial objects in a category.

Example 3.11. Given any category $\mathbf{C}$, a functor $F : \Delta^{op} \to \mathbf{C}$ is called a simplicial object in the category $\mathbf{C}$. One very important example is functor categories where the codomain is the category **Set** of sets, in which case the functor category is called a presheaf category. Combining these two notions we get the category $\mathcal{S}$ of functors $\Delta^{op} \to \mathbf{Set}$ called simplicial sets.

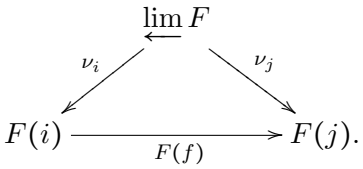
There are a number of constructions we can do within a category, and most of them are characterized by a universal property. Thus these constructions can usually be given by a universal construction which is called the limit of a diagram. A diagram in a category C is a functor $F : D \to C$, where D is a small category (or finite in which case the diagram will also be called finite). A cone over F is an object $c \in C$ with a collection of morphisms $\iota_i : c \to F(i)$ for each $i \in D$, such that for all $f : i \to j$ in D we have a commutative triangle



A map of two cones $\{c \to F(i)\}_{i \in D} \to \{d \to F(i)\}_{i \in D}$ is a map $c \to d$ such that the following diagram commutes for all possible $f : i \to j$



If the limit exists, it is a special cone called the universal cone, and we denote it by



Universality is the property that for every other cone $\{c \to F(i)\}_{i \in D}$ we have a unique map of cones $c \to \varprojlim F$.

Some categories have the property that all limits exists and these are called complete. Some important examples of limits are terminal objects, which is the limit of the empty diagram so comes with a unique morphism from any object; product of two objects, which is the limit of the diagram with two objects and no morphisms; equalizers of two morphisms, which is the limit of the diagram with two objects and two parallel morphisms between them. These are not in fact just random examples, but all limits of finite diagrams can be constructed from iterated limits of these examples.

A colimit in $\mathbf{C}$ is a limit in $\mathbf{C}^{op}$. Initial object is the colimit of the empty diagram so comes with a unique morphism to any other object.

Having isomorphisms between categories are sometimes a too strict notion to consider, as sometimes objects in different categories are related even though the categories are not isomorphic. This inspired the definition of adjoints.

Definition 3.12. *Let C and D be categories. An adjunction from between C and D is a triple: $F : C \to D$, $G : D \to C$ and*

$$\phi : C(Fx, y) \cong D(x, Gy),$$

which is an isomorphism of the morphism sets, and a natural transformation in both $x \in C$ and $y \in D$.

In such a setting we call F the left adjoint, and G the right adjoint. It is an easy exercise to see that:

Proposition 3.13. *If (F, G, ϕ) is an adjunction between C and D, then F preserves all colimits in C and G preserves all limits in D.*

4 Model categories

A model category is a convenient way of studying the homotopic properties in a category in a more abstract way, so as to easier compare different categories with seemingly equal homotopy structure. We do this by formally inverting a certain class of maps in the category, but as often happens we lose a lot of control by localizing in this way. For instance the set of maps in a category may no longer be a set after localization. To counter this, Daniel Quillen did a clever trick when he introduced the notion of a model category. He also required two other classes of maps with certain properties called fibrations and cofibrations to exist in the category and showed that this gives a way of regaining the necessary control to talk about the homotopy category. We will present model categories as they are presented in [Hov99], which differs slightly from Quillen's original presentation. We require functorial factorization to be part of the structure, but in all interesting examples this is already the case so we consider this no big inconvenience.

To define a model category we first need to give some preliminary definitions.

Definition 4.1. *Given a category* $\mathbf{C}$*, we can define the category* $Map\mathbf{C}$ *of morphisms in* $\mathbf{C}$ *and commutative squares.*

Definition 4.2. *A map* f *in* $\mathbf{C}$ *is a retract of a map* g *in* $\mathbf{C}$ *if* f *is a retract of* g *as objects of* $Map\mathbf{C}$*. That is, if there is a commutative diagram*

$$\begin{array}{ccccc} A & \longrightarrow & C & \longrightarrow & A \\ {\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow & & {\scriptstyle f}\downarrow \\ A & \longrightarrow & D & \longrightarrow & A, \end{array}$$

such that both the horizontal compositions are the identity.

Definition 4.3. *A functorial factorization is an ordered pair* $(\alpha, \beta) : Map\mathbf{C} \to Map\mathbf{C}$ *such that* $f = \beta(f) \circ \alpha(f)$ *for all* $f \in Map\mathbf{C}$*.*

Definition 4.4. *Suppose* $i : A \to B$ *and* $p : X \to Y$ *are maps in a category* $\mathbf{C}$*. Then we say that* i *has the left lifting property with respect to* p *and that* p *has the right lifting property with respect to* i*, if for every solid diagram*

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle i}\downarrow & \overset{h}{\nearrow} & \downarrow{\scriptstyle f} \\ B & \longrightarrow & Y \end{array}$$

we have the dotted arrow $h : B \dashrightarrow X$ *such that the resulting triangles commute.*

Definition 4.5. *A model structure on* $\mathbf{C}$ *consists of three subcategories of* $\mathbf{C}$ *called weak equivalences* $w\mathbf{C}$*, fibrations* $fib\mathbf{C}$ *and cofibrations* $cof\mathbf{C}$ *and two functorial factorizations* (α, β) *and* (δ, γ) *satisfying the following rules:*

1. *(2-out-of-3) If* f *and* g *are morphism in* $\mathbf{C}$ *such that* $g \circ f$ *if defined and two of* f*,*g *and* $g \circ f$ *are in* $w\mathbf{C}$ *then so is the third.*

2. *(Retracts) If* f *is a retract of* g *and* g *is in* $w\mathbf{C}$*,* $fib\mathbf{C}$ *or* $cof\mathbf{C}$ *then so is* f*.*

3. *(Lifting) Define a map to be a trivial/acyclic cofibration if it is in $cof\mathbf{C} \bigcap w\mathbf{C}$, and a trivial/acyclic fibration if it is in $fib\mathbf{C} \bigcap w\mathbf{C}$. Then fibrations have the right lifting property with respect to trivial cofibrations, and cofibrations have the left lifting property with respect to trivial fibrations.*

4. *(Factorization) For any morphism f, $\alpha(f)$ is a cofibration, $\beta(f)$ is a trivial fibration, $\delta(f)$ is a trivial cofibration, and $\gamma(f)$ is a fibration.*

Definition 4.6. *A model category is a category $\mathbf{C}$ with all small limits and colimits and a model structure on $\mathbf{C}$.*

Since a model category has all small limits and colimits it automatically has an initial object and a terminal object. We call an object fibrant if the unique map to the terminal object is a fibration, and an object cofibrant if the unique map from the initial object is a cofibration.

Definition 4.7. *A model category $\mathbf{C}$ is called cofibrantly generated if there are sets I, called the generating cofibrations, and J, called the generating trivial cofibrations, of maps such that:*

1. *The domains of the maps of I are small relative to I-cell;*

2. *The domains of the maps of J are small realtive to J-cell:*

3. *The class of fibrations is J-inj;*

4. *The class of trivial fibrations is I-inj.*

For notation and more details look in definition 2.1.17 of [Hov99].

Example 4.8. The model category $\mathcal{S}$ of simplicial sets is cofibrantly generated with generating cofibrations

$$I = \{\partial\Delta[n] \subseteq \Delta[n] | n \geq 0\},$$

and generating trivial cofibrations

$$J = \{\Lambda^k[n] \subseteq \Delta[n] | 0 \leq k \leq n > 0\}.$$

We will leave the rest of the study of model categories to be found in [Hov99] and will only include the theorems we will need in the rest of the text.

Theorem 4.9. *(Transfer principle [BM03])*
Let $\mathbf{D}$ be a cofibrantly generated model category and let $F : \mathbf{D} \rightleftarrows \mathbf{E} : G$ be an adjunction with left adjoint F and right adjoint G. Assume that $\mathbf{E}$ has small colimits and finite limits. Define a map f in $\mathbf{E}$ to be a weak equivalence (resp. fibration) iff $G(f)$ is a weak equivalence (resp. fibration). Then this defines a cofibrantly generated model structure on $\mathbf{E}$ provided

1. *the functor F preserves small objects;*

2. *any sequential colimit of pushouts of images under F of the generating trivial cofibrations of $\mathbf{D}$ yields a weak equivalence in $\mathbf{E}$.*

Lemma 4.10. *Under all the assumptions of Theorem , if* **D** *is right proper then so is* **E**.

Proof. Suppose we have a pullback diagram

$$\begin{array}{ccc} A & \longrightarrow & X \\ \downarrow & & \downarrow \\ B & \xrightarrow{\sim} & Y, \end{array}$$

where $X \twoheadrightarrow Y$ is a fibration, and $B \xrightarrow{\sim} Y$ is a weak equivalence. We want to know whether $A \to X$ is a weak equivalence. By the definition of the model structure the map $X \twoheadrightarrow Y$ is a fibration iff the map $GX \twoheadrightarrow GY \in \mathbf{D}$ is a fibration, and $B \to Y$ is a weak equivalence iff $UB \to UY \in \mathbf{D}$ is a weak equivalence. Furthermore right adjoints like G always preserve limits, so in fact the whole pullback diagram is sent to a pullback diagram in simplicial sets of the form:

$$\begin{array}{ccc} GA & \longrightarrow & GX \\ \downarrow & & \downarrow \\ GB & \xrightarrow{\sim} & GY. \end{array}$$

Now **D** is right proper so the map $GA \to GX$ is a weak equivalence, and hence by definition $A \to X$ is also a weak equivalence. □

5 Dependent type theory

Type theory is a form of symbolic logic. That is a specific way of writing and rules for rewriting formal symbols, where the rules we choose are meant to reflect our intuition about what these symbols should represent. This viewpoint is strengthened by interpreting propositions in first-ordered logic as types (the basic objects) of type theory, and the later interpretation of type theory as a way of doing formal homotopy theory and formalizing mathematics in proof assistants.

Dependent type theory, or more specifically Per Martin-Löf's dependent type theory is a flavor of type theory where we allow types that can depend on other types, which we will see examples of below. Dependent type theory is in itself a constructive logic, so we have no excluded middle ($A \vee \neg A$), or double-negation ($\neg\neg A \to A$). The original references for Martin-Löf's dependent type theory are [ML84] and [ML75]

There is an underlying logic in the type theory to take care of when two things are definitionally equal (written $a \equiv b$), how to form formulas, how substitution in handled, what are the free variables, what lambda abstraction is, and so on. We will not mention any of this here, but direct the interested reader to [ML75] or in appendix A of [Uni13]. One could also view definition 6.1 as a way of formalizing all the underlying logic.

There is a very important notion of universes which is crucial when talking about foundations, just as a Grothedieck universe is important for the study of set theory and category theory, but we will only talk informally about it. We have a sequence of universes $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots$. Where $\mathcal{U}_0 : \mathcal{U}_1$ is meant to denote that $\mathcal{U}_0$ "lives within" $\mathcal{U}_1$, just like we use the notation $a \in A$ in set theory to denote that a lives in A. We will usually not concern ourself with which level of the sequence of universes our type lives at, so we will frequently write $A : \mathcal{U}$ to denote that $A : \mathcal{U}_n$ for some n. Our most basic types are assumes to live in the lowest level of the hierarchy $\mathcal{U}_0$. Further our universes are assumed to be cumulative, i.e. if $A : \mathcal{U}_i$ then also $A : \mathcal{U}_{i+1}$. There are multiple ways of making the above notions formal, but we will follow the style of [Uni13] and talk a bit more informally; however in parallel all the notions are formalized expressively in the proof assistant [dt04] in the standard library at `https://github.com/HoTT/HoTT` so that everyone can look at the formal proofs of every statement.

As mentioned above the basic objects of type theory are called *types* and are the objects $A : \mathcal{U}$. Further we call the objects of the types *terms*, and write $a : A$ to mean that a is a term of the type A. A context is like a list of assumptions for proving a theorem, and if we have a finite list of term declarations we call that a context. More formally:

Definition 5.1. *We define a context inductively by*

- *The empty list is a context*
- *If Γ is a context and a is a term not appearing in Γ and A is a type which is valid in Γ, then the new list $\Gamma, a : A$ is also a context.*

We will define what it means for a type to be valid in a context below, and we will use the notation $\Gamma \vdash a : A$ to mean that declaring $a : A$ is valid in the context Γ, and by abuse of notation we will also write $\Gamma \vdash A : \mathcal{U}$ to say that A is a valid type given the context Γ. Forming a type usually consists of four steps:

1. **Formation** in which we give the rules for forming the type, maybe depending on previously formed types.

2. **Introduction** in which we give the rules for forming the terms of the type.

3. **Elimination** in which we give the ways of retrieving a term of one of the old types used to form the new type, or an induction rule for giving terms of types depending on the new type.

4. **Computation** in which we give the relationship between the terms coming from Introduction and Elimination.

We will see better how these rules work in practice by looking at some examples. We will use the natural deduction style of showing that if the formulas above the line are valid, then so is the formula under the line. Using the proposition as types interpretation we will recover the usual rules of first-order logic when we later comment on what each operation in logic should correspond to in type theory. Now we will give all the standard constructions in dependent type theory which we will use in this article, following the four steps listed above.

Binary product type

1. The formation rule

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \times B : \mathcal{U}}$$

simply states that if we have two different types A, B, then using the same context we can form a new type $A \times B$ called the (binary) product type.

2. The introduction rule

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$$

gives the way of constructing terms of the product type, given terms of the two types from which it is constructed. That is, if $a : A$ and $b : B$, we have a term $(a, b) : A \times B$. A priori this term does not have to have any connection to the two terms from which it is constructed, apart from the fact that we need to know these two terms exists and have the right type, so the connection comes in the elimination principle and its computation rule.

3. The elimination principle

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash pr_1 p : A} \qquad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash pr_2 p : B}$$

gives us a chosen way of constructing a term of A and a term of B from a single term of $A \times B$. When we later learn about function types this exactly gives two functions $A \times B \to A$ and $A \times B \to B$.

4. The computation rule

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma, \Delta \vdash pr_1(a,b) \equiv a : A} \qquad \frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma, \Delta \vdash pr_2(a,b) \equiv b : B}$$

precisely gives that the functions pr_1 and pr_2 are the first and second projections of the pair (a, b). Now all we need is the fact that all terms of $A \times B$ are pairs constructed in this way to see that product types behave exactly like the product of two sets (or more generally like the categorical product in category theory).

Function type

1. The formation rule

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A \to B : \mathcal{U}}$$

is the same as for products but with a different symbol symbol, and of course the terms have a very different form.

2. The introduction rule

$$\frac{\Gamma, x : A \vdash f(x) : B}{\Gamma \vdash \lambda x.f(x) : A \to B}$$

use lambda abstraction to avoid giving a name to the function assigning $x : A$ to the term $f(x)$ by applying the rule hidden in the $\vdash$. For instance in normal math we have for every natural number n another natural number $2 * n$, so we could use the notation $\lambda n : \mathbb{N}.2 * n$ to avoid giving a name such as "double" for this function so we reduce the number of names we use, since everything in type theory has to have a unique name.

3. The elimination rule

$$\frac{\Gamma \vdash f : A \to B \qquad \Delta \vdash t : A}{\Gamma, \Delta \vdash ft : B}$$

is chosen to mimic the behavior of the functions in sets (or morphisms in a category), but we would like to remind the reader that without any computation rule ft is simply a formal name for a term in B.

4. The computation rule

$$\frac{\Gamma, x : A \vdash f(x) : B}{\Gamma, t : A \vdash (\lambda x.f(x))t \equiv ft : B}$$

is what allows us to conclude that at least all functions constructed using lambda abstraction behave like functions ought to from our experience. So $(\lambda n : N.2 * n)(4) \equiv 2 * 4$ for instance.

Both of these are special cases of something called dependent types, which is the backbone of dependent type theory. We call a type $B : A \to \mathcal{U}$ a type family, or dependent type, because for each $a : A$ we get a type $Ba : \mathcal{U}$ depending on A. If we now mimic the constructions before with dependent types we get dependent sums and dependent products respectively, and if we specialize to the special case where $B : A \to \mathcal{U}$ is B no matter what $a : A$ we use (so in reality it is not a dependent type) we exactly get the old examples back, precisely we use $(\lambda x : A.B) : A \to \mathcal{U}$.

Dependent sum type

1. The formation rule

$$\frac{\Gamma, x : A \vdash Bx : \mathcal{U}}{\Gamma \vdash \sum_{x:A} Bx : \mathcal{U},}$$

   just like for product type, requires the knowledge of two types A and B. However as want our type B to be a dependent type, it already contains the knowledge of the type A, so we only need B and a big enough context Γ to contain A.

2. The introduction rule

$$\frac{\Gamma \vdash a : A \qquad \Gamma, a : A \vdash b : Ba}{\Gamma \vdash (a, b) : \sum_{x:A} Bx}$$

   states that the terms we can construct in the dependent sum are pairs of terms $a : A$ and $b : Ba$, but we allow the second component to be of a type that depends upon the first component.

3. The elimination/induction rule

$$\frac{\Gamma, z : \sum_{x:A} Bx \vdash Cz : \mathcal{U} \qquad \Gamma, x : A, y : Bx \vdash g : C(x, y) \qquad \Gamma \vdash p : \sum_{x:A} Bx}{\Gamma \vdash ind_{\sum_{x:A}}(z.C, x.y.g, p) : C(p)}$$

   states that any type which depend on the dependent sum can be given a term as soon as we have a term in the special case when the type depends only on pairs constructed with the introduction rule. The intuition is that all terms of dependent sum $\sum_{x:A} Bx$ are pairs (a, b) of terms $a : A$ and $b : Ba$.

4. The computation rule

$$\frac{\begin{array}{c}\Gamma, z : \sum_{x:A} Bx \vdash Cz : \mathcal{U} \qquad \Gamma, x : A, y : Bx \vdash g : C(x, y) \\ \Gamma \vdash a : A \qquad\qquad \Gamma \vdash b : Ba\end{array}}{\Gamma \vdash ind_{\sum_{x:A} Bx}(z.C, x.y.g, (a, b)) \equiv g(a, b) : C(a, b)}$$

   follow a general recipe for computation rule for induction as can be seen multiple times in [Uni13]. The induction rule gives a way of generalizing pairs to all terms of dependent sum type, and computation rule says that in the case of pairs we recover the original term.

Dependent product type

1. The formation rule

$$\frac{\Gamma, x : A \vdash Bx : \mathcal{U}}{\Gamma \vdash \Pi_{x:A} Bx : \mathcal{U}}$$

   is just the same as for dependent sum, just as function type was the same as product type, only with a different name.

2. The introduction rule

$$\frac{\Gamma, x : A \vdash f(x) : Bx}{\Gamma \vdash \lambda x.f(x) : \Pi_{x:A} Bx}$$

   is the exact same way as for functions types, only with the codomain Bx being allowed to depend on which term a of the domain A we use. This basically says that if we already have a way of assigning to each term $x : A$ a term $f(x) : Bx$ then we can collect it all together using lambda abstraction into a single term of the dependent product type.

3. The elimination rule

$$\frac{\Gamma \vdash f : \Pi_{x:A} Bx \qquad \Delta \vdash t : A}{\Gamma, \Delta \vdash ft : Bt}$$

   becomes what is expected if it is to be a generalization of elimination for the product type.

4. The computation rule

$$\frac{\Gamma, x : A \vdash f(x) : \Pi_{x:A} \qquad \Delta \vdash t : A}{\Gamma\Delta \vdash (\lambda x.f(x))t \equiv ft : Bt}$$

   is what guarantees that the lambda abstractions in dependent products acts precisely like functions where the type of the output is allowed to depend of the input.

In the examples that follow we will explain the elimination/induction rule in the viewpoint of propositions as types, in which types of the form $C : B \to \mathcal{U}$ corresponds to propositions we can prove about the type B. This viewpoint will be explained more in the end of this section.

Coproduct

1. The formation rule

$$\frac{\Gamma \vdash A : \mathcal{U} \qquad \Gamma \vdash B : \mathcal{U}}{\Gamma \vdash A + B : \mathcal{U}}$$

   is essentially the same as for binary product since coproduct $A + B$ can be formed any time we can form the types A and B.

2. The introduction rule

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash inl(a) : A + B} \qquad \frac{\Delta \vdash b : B}{\Delta \vdash inr(b) : A + B}$$

is dual to the introduction rule for products in that we only need a term of A or B to get a term om $A + B$, though with a new name to distinguish it from the original term.

3. The elimination/induction rule

$$\frac{}{ind_{A+B} : \Pi_{C:A+B\to\mathcal{U}}((\Pi_{a:A} C inl(a)) \to (\Pi_{b:B} C inr(b)) \to \Pi_{x:A+B} C x)}$$

codes a lot of information into one single type which is inhabited. If we analyze it we can read that if we want to prove that something holds for all terms in $A + B$, we can do this by proving it for all terms of A and all terms of B.

4. The computation rule

$$\frac{}{\begin{array}{l} ind_{A+B}(C, c_1, c_2, inl(a)) \equiv c_1(a) \\ ind_{A+B}(C, c_1, c_2, inr(b)) \equiv c_2(b) \end{array}}$$

says that the above proof for terms of $A + B$ which are terms coming from A or B coincides with the original proof in that case.

Now we have a few different ways of constructing new types from already existing ones, so it would be nice to have know that there does in fact exist some types in our system. We will construct these types as inductive types, so they come with a very convenient induction rule.

Unit type

1. The formation rule

$$\frac{}{\mathbf{1} : \mathcal{U}}$$

says that without the need for any context (which is another way of saying for any context) the type 1 exists.

2. The introduction rule

$$\frac{}{* : \mathbf{1}}$$

gives an inhabitant of 1 called $*$, again in no/all context(s).

3. The elimination rule

$$\frac{}{ind_{\mathbf{1}} : \Pi_{C:\mathbf{1}\to\mathcal{U}}(C* \to \Pi_{x:\mathbf{1}} Cx)}$$

follows directly from the fact that 1 is inductively defined, and essentially says that if we can prove something for the term $*$, we can prove the same thing for all terms of 1.

4. The computation rule

$$\frac{}{ind_{\mathbf{1}}(C, c, *) \equiv c,}$$

says the usual thing for induction rules, i.e. that the proof for all terms of 1 is the correct one in the case of the term being $*$.

Empty type

1. The formation rule

$$\frac{}{\mathbf{0} : \mathcal{U}}$$

is similar to the one for unit type; the empty type always exists.

2. The empty type is constructed to behave as if it does not have any elements, so there are no introduction rules

3. The elimination rule

$$\frac{}{ind_{\mathbf{0}} : \Pi_{C:\mathbf{0}\to\mathcal{U}} \Pi_{z:\mathbf{0}} Cz}$$

can be read as an ex falso rule, since any type/proposition A can be written as $\lambda z : 0.A$. Hence if we somehow find ourself with an accessible term $z : 0$ then induction given that the type/proposition A is inhabited/proved, no matter what A is, since it does not in fact depend on what $z : 0$ is.

4. Since the empty type does not have any introduction rule there is no need for a computation rule.

One of the most important types of dependent type theory, and the one that differentiates extensional and intensional type theory, is the identity type:

Identity type

1. The formation rule

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma, a : A, b : A \vdash (a =_A b) : \mathcal{U}}$$

looks slightly different the former ones, since the type is constructed using terms in stead of types. The terms of identity type $a =_A b$ are called paths from a to b, when we are using the viewpoint of homotopy theory which we discuss more in chapter 7.

2. The introduction rule

$$\frac{\Gamma \vdash A : \mathcal{U}}{\Gamma, a : A \vdash refl_a : (a =_A a)}$$

says that we always have a path from a to a called $refl_a$ or the reflexivity path.

3. The induction rule

$$\frac{\Gamma, a : A, b : A, p : (a =_A b) \vdash C(x, y, p) : \mathcal{U} \qquad \Gamma, a : A \vdash t : C(a, a, refl_a)}{\Gamma, a : A, b : A, p : (a =_A b) \vdash ind_=(C, t) : \Pi_{a,b:A}\Pi_{p:(a=_A b)}C(a, b, p)}$$

is arguably the most powerful tool in homotopy type theory, since it allows us to deduce pretty much any proposition involving path between any two terms a and b as long as we can prove it in the case of reflexivity $refl_a$ on the term a.

4. The computation rule

$$\frac{\Gamma, a : A, b : A, p : (a =_A b) \vdash C(x, y, p) : \mathcal{U} \qquad \Gamma, a : A \vdash t : C(a, a, refl_a)}{\Gamma, a : A \vdash ind_=(C, t)(a, a, refl_a) \equiv t}$$

as before, since it talks about induction, says that in the case of $refl_a$ induction is the correct term.

In the beginning of type theory the role of Identity types was not properly understood, since the rules look very similar to rules for equality, in stead of the later understanding that Identity types behave more like paths and homotopies. In fact there exists a flavor of type theory called extensional type theory which add another rule to the type theory which make Identity types nothing more than equality. The rule, without writing out the context, says that if we have a $p : (a =_A b)$ then $a \equiv b$ which again implies (by induction) that $p \equiv refl_a$. However this makes the type theory undecidable, so there is no efficient way of checking whether a judgment $\Gamma \vdash x : A$ is derivable in the type theory.

We can view propositions in logic as certain types in type theory, and then a proof of the proposition corresponds to a inhabitant of the type. As remarked in the introduction to this section the logic which can be represented in type theory is necessarily constructive. We now give the translations from logic to type theory

- True is given by the unit type 1.
- False is given by the empty type 0.
- Conjunction, A and B, is given by the product type $A \times B$.
- Disjunction, A or B, is given by the sum type $A + B$.
- Implication, if A then B, is given by the function type $A \to B$.
- Equivalence, A if and only if B, is given by $(A \to B) \times (B \to A)$.
- Negation, not A, is given by $A \to 0$.

The use of equivalence here is very different from the equivalences discussed in 7.3, and we call this version logical equivalence. This is all the rules for propositional logic, but we can also give predicative logic by including quantifiers. If we have a type $P : A \to \mathcal{U}$, which we view as a proposition about the type A.

- For all $x : A$, $P(x)$ holds, is given by Π-type: $\prod_{x:A} P(x)$.
- There exists $x : A$ such that $P(x)$, is given by Σ-type: $\sum_{x:A} P(x)$.

There is some subtle problems involved in the use of disjunction and existential quantifier as used above, namely that terms of sum- and Σ-types remember the name of the term and not just the existence. Hence there is slightly more information in the type theory than in the logic, which is precisely that a term of $A + B$ "remembers" if it came from A or from B and a term of $\sum_{x:A} P(x)$ remembers exactly which x it is. There is a construction in type theory called propositional truncation $\| - \|$, which forgets the name of term and simple remembers the existence. To give this construction however would require us to go into a discussion of n-types, propositions within type theory which is not really relevant for the topics discussed in this article. It basically collapses all type to be equivalent to either 1 or 0 depending on whether it is inhabited or not, so all proposition become true of false, and the way in which it is true is unique. This gives us classical logic. The interested readers is recommended to read chapter 3 and chapter 7 of [Uni13].

6 Contextual categories

In this chapter we will follow a number of sources and try to give an exposition of the current knowledge of contextual categories. We will also look at ways of interpreting the structure found in a flavor of logic into category theory. Since this is a relatively new field of study most of the work is done in preprints so a number of the references are unfortunately unpublished and therefore not peer-reviewed. We will try to give references to published results when they are available. Contextual categories are of interest to us since it give a lot of necessary conditions for a category to be a model for type theory, and some helpful sufficient conditions. This is what directs us in chapter 9 and 10 to which categories could be of interest to show models type theory.

6.1 Contextual category

This section follows [KLV12] which is the way the model for simplicial sets in 9 is constructed, and which we try to mimic for the simplicial symmetric monoid in the same chapter. We give the definition of a contextual category first for a dependent type theory without any way of forming new types from old ones, and then give the ways to define new types afterwards.

Definition 6.1. *A contextual category* $\mathbf{C}$ *consists of the following:*

1. *A category* $\mathbf{C}$*;*
2. *a grading of objects, such that* $obj(\mathbf{C}) = \coprod_{n\geq 0} obj_n(\mathbf{C})$*;*
3. *an object* $1 \in obj_0(\mathbf{C})$*;*
4. *maps* $ft_n : obj_{n+1}(\mathbf{C}) \to obj_n(\mathbf{C})$ *(whose subscripts we usually suppress);*
5. *for each* $X \in obj_{n+1}(\mathbf{C})$*, a map* $p_X : X \to ft(X)$ *called the canonical projection from* X*;*
6. *for each* $X \in obj_{n+1}(\mathbf{C})$ *and* $f : Y \to ft(X)$*, an object* $f^*(X)$ *and a map* $q(f, X) : f^*(X) \to X$*;*

such that:

1. 1 *is the unique object in* $obj_0(\mathbf{C})$*;*
2. 1 *is a terminal object of* $\mathbf{C}$*;*
3. *for each* $X \in obj_{n+1}(\mathbf{C})$ *and* $f : Y \to ft(X)$*, we have* $ft(f^*X) = Y$*, and the square*

$$\begin{array}{ccc} f^*X & \xrightarrow{q(f,X)} & X \\ {\scriptstyle p_{f^*(X)}}\downarrow & & \downarrow{\scriptstyle p_X} \\ Y & \xrightarrow{f} & ft(X) \end{array}$$

   is a pullback (called the canonical pullback of X *along* f*); and*

4. *these canonical pullbacks are strictly functorial: that is, for* $X \in obj_{n+1}(\mathbf{C})$*,* $1^*_{ft(X)}X = X$ *and* $q(1_{ft(X)}, X) = 1_X$*; and for* $X \in obj_{n+1}(\mathbf{C})$*,* $f : Y \to ft(X)$ *and* $g : Z \to Y$*, we have* $(fg)^*(X) = g^*(f^*(X))$ *and* $q(fg, X) = q(f, X)q(g, f^*X)$*.*

This definition is perhaps best understood through the standard example in which we will be using it.

Example 6.2. Let $\mathbf{T}$ be a type theory, then there is a contextual category $Con(\mathbf{T})$ directly constructed from it as follows.

1. The category $Con(\mathbf{T})$ has as objects contexts $[x_1 : A_1, \ldots, x_n : A_n]$, for $A_i \in \mathbf{T}$, (the $[-]$ notation is just list notation) up to definitional equality and renaming of free variables in the type theory. Morphisms are given by substitution, in the following sense: a map

$$f : [x_1 : A_1, \ldots, x_n : A_n] \to [y_1 : B_1, \ldots, y_m : B_m(y_1, \ldots, y_{m-1})]$$

is given by a sequence $(f_1, f_2, \ldots, f_m)$ with

$$\begin{aligned} [x_1 : A_1, \ldots, x_n : A_n] &\vdash f_1 : B_1 \\ [x_1 : A_1, \ldots, x_n : A_n] &\vdash f_2 : B_2(f_1) \\ &\vdots \\ [x_1 : A_1, \ldots, x_n : A_n] &\vdash f_m : B_m(f_1, \ldots, f_{m-1}) \end{aligned}$$

and two such maps $[f_i], [g_i]$ are equal if for every i we have

$$[x_1 : A_1, \ldots, x_n : A_n] \vdash f_i \equiv g_i$$

Composition is given by substitution, and the identity is given by the identity substitution;

2. the grading is given by the length of the context;
3. 1 is given by the empty context $[]$;
4. $ft([x_1 : A_1, \ldots, x_n : A_n, x_{n+1} : A_{n+1}]) = [x_1 : A_1, \ldots, x_n : A_n]$;
5. for $\Gamma = [x_1 : A_1, \ldots, x_{n+1} : A_{n+1}]$ we have

$$p_\Gamma = (x_1, \ldots, x_n) : [x_1 : A_1, \ldots, x_{n+1} : A_{n+1}] \to [x_1 : A_1, \ldots, x_n : A_n]$$

by forgetting the last variable, where x_i is the substitution $[x_1 : A_1, \ldots, x_n : A_n] \vdash x_i : A_i(x_1, \ldots, x_{n-1})$;

6. for contexts

$$\Gamma = [x_1 : A_1, \ldots, x_{n+1} : A_{n+1}(x_1 \ldots, x_n)],$$

and

$$\Gamma' = [y_1 : B_1, \ldots, y_m : B_m(y_1 \ldots, y_{m-1})],$$

and a map

$$f = \left[f_i(\overrightarrow{y})\right]_{i \le n} : \Gamma' \to ft(\Gamma),$$

the pullback $f^*\Gamma$ is the context

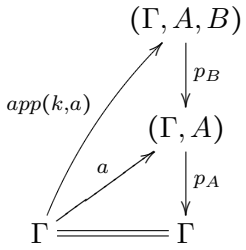
$$\left[y_1 : B_1, \ldots, y_m : B_n(y_1 \ldots, y_{m-1}), y_{m+1} : A_{n+1}(f_1(\overrightarrow{y}), \ldots, f_n(\overrightarrow{y})\right],$$

and $q(f, \Gamma) : f^*\Gamma \to \Gamma$ is given by $[f_1, \ldots, f_n, y_{n+1}]$.

For an object Γ we will in this chapter use the notation (Γ, A) to denote an object with $ft(\Gamma, A) = \Gamma$, and in this case we will write p_A in stead of $p_{(\Gamma,A)}$ for simplicity. We will now give an example of how to interpret extra structure on the type theory in a contextual category.

Definition 6.3. *A Π-type structure on a contextual category $\mathbf{C}$ consists of the following:*

1. *for each $(\Gamma, A, B) \in obj_{n+2}(\mathbf{C})$, an object $(\Gamma, \Pi(A, B)) \in obj_{n+1}(\mathbf{C})$;*
2. *for each section $b : (\Gamma, A) \to (\Gamma, A, B)$ of a dependent projection p_A, a morphism $\lambda(b) : \Gamma \to (\Gamma, \Pi(A, B))$;*
3. *for each pair of sections $k : \Gamma \to (\Gamma, \Pi(A, B))$ and $a : \Gamma \to (\Gamma, A)$, a section $app(k, a) : \Gamma \to (\Gamma, A, B)$ such that the following diagram commutes:*



4. *for $a : \Gamma \to (\Gamma, A)$ and $b : (\Gamma, A) \to (\Gamma, A, B)$, we have $app(\lambda(b), a) = ba$;*
5. *all the above structure is strictly stable under pullback: for any morphism $f : \Delta \to \Gamma$ we have*

$$(\Delta, f^*\Pi(A, B)) = (\Delta, \Pi(f^*A, f^*B)),$$
$$\lambda(f^*b) = f^*(\lambda(b)), \qquad app(f^*k, f^*a) = f^*(app(k, a)).$$

If we look at the rules for dependent product in type theory we see that these are a direct translation of those four rules, plus a rule guaranteeing that substitution through pullback is strict.

Definition 6.4. *A contextual functor $F : \mathbf{C} \to \mathbf{D}$ of contextual categories consists of a functor $F : \mathbf{C} \to \mathbf{D}$, between the underlying categories, respecting the grading, and preserving all the structure of the contextual category with equality (not just isomorphism). Further, a contextual functor of contextual categories with Π-type structure, or even more structure, should also preserve the additional structure.*

The following theorem is theorem 1.2.9 in [KLV12], but as noted there it has no complete proof known.

Theorem 6.5. *Let $\mathbf{T}$ be a type theory with any combination of type formers from chapter 5, then $Con(\mathbf{T})$ is initial among contextual categories, and contextual functors, with the same extra structure corresponding to the type formers in the type theory.*

We will assume that this theorem is true in this article, but will only use it for the following definition. The definition does hold without the theorem, but is more natural with it.

Definition 6.6. *A model of dependent type theory* $\mathbf{T}$ *with any combination of type formers, is a contextual category* $\mathbf{C}$ *together with a contextual functor* $F : Con(\mathbf{T}) \to \mathbf{C}$.

Of course in light of the theorem above the functor F is automatically given.

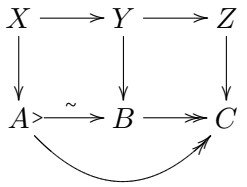
More on the connection between type theory and categories can be found in chapter D.1 and D.4 of [Joh02b].

6.2 Type-theoretic fibration category

We now switch over to different way of interpreting type theory which may seem more natural to readers more familiar with model categories than with dependent type theory. This method follows [Shu12].

Definition 6.7. *A type-theoretic fibration category is a category* $\mathbf{C}$ *with the following structure:*

1. *A terminal object* 1*;*
2. *a subcategory* $\mathbf{F} \subset \mathbf{C}$ *containing all the objects, all the isomorphisms, and all the morphisms with codomain* 1*. The morphisms of* $\mathbf{F}$ *are called fibrations, and we write them as* $A \twoheadrightarrow B$*. Further any morphism having the left lifting property with respect to a fibration is called an acyclic cofibration and is written* $A \xrightarrow{\sim} B$*;*
3. *all pullbacks of fibrations exists and are fibrations;*
4. *for every fibration* $g : A \twoheadrightarrow B$*, the pullback functor* $g^* : \mathbf{C}/B \to \mathbf{C}/A$ *has a partial right adjoint* Π_g *defined at all fibrations over* A *and whose values are fibrations over* B*.*
5. *every morphism factors as an acyclic cofibration followed by a fibration;*
6. *in the following commutative diagram:*



if $B \twoheadrightarrow C$ *and* $A \twoheadrightarrow C$ *are fibrations and* $A \xrightarrow{\sim} B$ *is an acyclic cofibration, and both squares are pullback square, then (by 3* $Y \to Z$ *and* $X \to Z$ *are both fibrations, and we also require that)* $X \to Y$ *should be an acyclic cofibration.*

An object A is usually called fibrant if the map $A \to 1$ is a fibration, so we assume that all objects in the category are fibrant. In type theory the name display map is sometimes used in stead of fibrations, and its worth noting that conditions 1, 2, 3, 4 make $\mathbf{C}$ into a display map category (see [Jac99] §10.4).

There are mainly two examples we will be interested in.

Example 6.8. Any contextual category of a type theory $Con(\mathbf{T})$ is a type-theoretic fibration category in the following way. We define the fibrations to be any morphism that is isomorphic to a $p_X : X \to ft(X)$. We will not check the actual conditions here, but refer the interested reader to [Shu12].

The other example is the notion of type-theoretic model category which we now define.

Definition 6.9. *A type-theoretic model category is a model category* $\mathbf{C}$ *with the following additional properties:*

1. *Limits preserve cofibrations;*
2. $\mathbf{C}$ *is right proper, i.e. weak equivalences are preserved by pullback along fibrations;*
3. *pullback* g^* *along any fibration has a right adjoint* Π_g*.*

Lemma 6.10. *If* $\mathbf{C}$ *is a type-theoretic model category, then its full subcategory* $\mathbf{C}_{\underline{f}}$ *of fibrant objects is a type-theoretic fibration category*

Proof. This follows almost immediately from the definition. 1 - 3 and 5 of Def 6.7 are always true for model categories. For 4 we need to prove that Π_g preserve fibration, but this is equivalent to pullback g^* preserving trivial cofibrations along fibrations. Pullback is a limit so preserve cofibrations by definition, and weak equivalences is preserved since $\mathbf{C}$ is right proper by definition. Lastly, 6 follows since cofibrations are stable under pullback, and weak equivalences between fibrations are always stable under pullback. □

In these last two examples we have one problem, which has long been a problem in interpreting type theory correctly, but has recently gained multiple different solutions. Pullback in category theory preserves structure only up to isomorphisms, but in type theory it is required to be preserved on the nose by equality. This is known as the coherence issue, and kept people from accepting the categorical semantics for a long time. In the setting of contextual categories above this is treated properly, but this in turn makes it very difficult to find examples. There are ways of fixing coherence by technical requirements on morphisms, but in [KLV12] a much more pleasing way is presented by using the thought that there should exist a hierarchy of "type of types", such as the universes from chapter 5, which should also be representable within our category.

Definition 6.11. *Let* $\mathbf{C}$ *be a category. A universe in* $\mathbf{C}$ *is an object* U *together with a morphism* $p : \tilde{U} \to U$ *and for each map* $f : X \to U$ *a choice of pullback square*

$$\begin{array}{ccc} (X;f) & \xrightarrow{Q(f)} & \tilde{U} \\ {\scriptstyle P_{(X,f)}}\downarrow & & \downarrow{\scriptstyle p} \\ X & \xrightarrow{f} & U. \end{array}$$

We often refer to a universe by U, with p and the chosen pullback understood.

Given a map $f : Y \to X$, we will write $\ulcorner f \urcorner$ for a map $X \to U$ such that $f \cong P_{(X;\ulcorner f \urcorner)}$ (sometimes even just $\ulcorner Y \urcorner$ if the map is understood). Also for a sequence of maps $f_1 : X \to U$, $f_2 : (X; f_1) \to U$, etc., we write $(X; f_1, \ldots, f_n)$ for $((\ldots(X; f_1); \ldots; f_n)$.

Definition 6.12. *Given a category* $\mathbf{C}$ *with a terminal object* 1 *and a universe* U *we define a contextual category* $\mathbf{C}_U$ *by:*

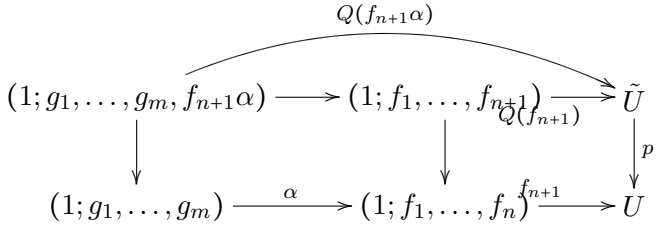
1. $obj_n(\mathbf{C}_U) = \{(f_1, \ldots, f_n) | f_i : (1; f_1, \ldots, f_{i-1}) \to U\}$*;*
2. $\mathbf{C}_U((f_1 \ldots, f_n), (g_1, \ldots, g_m)) = \mathbf{C}((1; f_1, \ldots, f_n), (1; g_1, \ldots, g_m))$*;*

3. $1_{\mathbf{C}_U} = ()$ *the empty sequence;*

4. $ft(f_1, \ldots, f_{n+1}) = (f_1, \ldots, f_n)$*;*

5. *the projection* $p_{(f_1 \ldots, f_{n+1})} = P_{(X, f_{n+1})}$ *is given by the universe structure;*

6. *given* $(f_1, \ldots, f_{n+1})$ *and a map* $\alpha : (g_1, \ldots, g_m) \to (f_1, \ldots, f_n)$ *in* $\mathbf{C}_U$*, the canonical pullback* $\alpha^*(f_1, \ldots, f_{n+1})$ *in* $\mathbf{C}_U$ *is given by* $(g_1, \ldots, g_m, f_{n+1}\alpha)$*, with projection induced by* $Q(f_{n+1}\alpha)$*:*



Lemma 6.13. *1. These data define a contextual category* $\mathbf{C}_U$*.*

2. *This contextual category is well-defined up to canonical isomorphism given just* $\mathbf{C}$ *and* $p : \tilde{U} \to U$*, independently of the choice of pullbacks and terminal object.*

All contextual categories can be given in exactly this way as the next lemma will show:

Lemma 6.14. *Let* $\mathbf{C}$ *be a small contextual category. Consider the universe* U *in the presheaf category* $[\mathbf{C}^{op}, Set]$ *given by*

$$U(X) = \{Y | ft(Y) = X\}$$
$$\tilde{U}(X) = \{(Y, s) | ft(Y) = X, s \text{ a section of } p_Y\},$$

with the obvious choice for projection map. Then $[\mathbf{C}^{op}, Set]_U$ *is isomorphic, as a contextual category, to* $\mathbf{C}$*.*

Both of these lemmas should be straight forward to prove according to [KLV12], the second with help from the Yoneda lemma.

6.3 The role of (locally) cartesian closure in models for dependent type theory

This section will follow [See84] which is the first appearance of the connection between type theories with Π-types, and locally cartesian closed categories. Though it does not handle the problem of coherence, this problem was ultimately solved in [CD11]. We study this connection since it gives the most requirements on categories that are models for type theory, which does not have as much with a choice of model structure, so is not something we can easily ignore as we will see later when we try to construct a model in the category of simplicial symmetric monoids.

The connection starts with the realization that objects indexed over other objects behave like over categories. More formally we have the theory of hyperdoctrines (which was already know to model full first order logic).

Definition 6.15. *For* $\mathbf{C}$ *a category with finite limits, a* $\mathbf{C}$*-indexed category is a functor* $\mathbf{P} : \mathbf{C}^{op} \to \mathbf{Cat}$*. We use the notation* $\mathbf{P}(f) \coloneqq f^* : \mathbf{P}(B) \to \mathbf{P}(A)$ *for the functor induced from* $f : A \to B$ *in* $\mathbf{C}$*.*

Definition 6.16. *A* $\mathbf{C}$*-indexed category* $\mathbf{P}$ *is a hyperdoctrine if*

1. *for each object* A *of* $\mathbf{C}$*,* $\mathbf{P}(A)$ *is cartesian closed,*
2. *for each* $f : A \to B$ *of* $\mathbf{C}$*,* f^* *preserves exponents,*
3. *for each* $f : A \to B$ *of* $\mathbf{C}$*,* f^* *has adjoints* $\sum_f \dashv f^* \dashv \prod_f$*,*
4. $\mathbf{P}$ *satisfy the Beck condition: if*

$$\begin{array}{ccc} D & \xrightarrow{h} & C \\ {\scriptstyle k}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & B \end{array}$$

is a pullback in $\mathbf{C}$*, then for any object* ϕ *of* $\mathbf{P}(C)$*,* $\sum_k h^* \phi \to f^* \sum_g \phi$ *is a isomorphism in* $\mathbf{P}(A)$*.*

As is perhaps obvious from the above notation we have for any category $\mathbf{C}$ with finite limits a $\mathbf{C}$-indexed category which is also called $\mathbf{C}$. That is for any $A \in \mathbf{C}$ we have $\mathbf{C}(A) \coloneqq \mathbf{C}/A$ the over category so that f^* is precisely the pullback. Almost from the definition we have that $\mathbf{C}$ is a locally cartesian closed category iff the $\mathbf{C}$-indexed category $\mathbf{C}$ is a hyperdoctrine.

We will now show that even by the most naive of interpretation of type theory in category we get a locally cartesian closed category, and has finite limits so is in in particular cartesian closed also. Given a type theory $\mathbf{M}$, we define a category $\mathbf{C}(\mathbf{M})$ with objects closed types in $\mathbf{M}$ and morphisms $A \to B$ closed terms of the type $A \to B$.

Theorem 6.17. *1. If* $\mathbf{M}$ *has function types, then* $\mathbf{C}(\mathbf{M})$ *is a category.*

2. *If* $\mathbf{M}$ *has* Σ*-types and (extensional) identity types, then* $\mathbf{C}(\mathbf{M})$ *has all finite limits.*
3. *If* $\mathbf{M}$ *has function types, then* $\mathbf{C}(\mathbf{M})$ *has all exponent, i.e. is cartesian closed.*
4. *If* $\mathbf{M}$ *has* Π*-types, then* $\mathbf{C}(\mathbf{M})$ *is locally cartesian closed.*

Proof. 1. For any object A we have identity id_A given by $\lambda x : A.x$. Given $f : A \to B$ and $g : B \to C$ (which can in fact be read as both morphisms in the category and as terms of function type), then we have $g \circ f : A \to C$ defined as $\lambda x : A.g(f(x))$

2. We have a terminal object in $\mathbf{C}(\mathbf{M})$ given by the unit type 1. And for any object A we have a morphism $A \to 1$ given by $\lambda x : A.*$ (where $*$ is the unique term in 1. It is an easy lemma in type theory that this morphism is unique.

   Binary products given by binary products, i.e. for objects A and B then $A \times B$ is given by $\sum_{x:A}(\lambda x : A.B)$. First and second projections are again themselves, and satisfy the universal property.

   Given $s, t : A \rightrightarrows B$, the equalizers of s, t is given by $\sum_{x:A}(s(x) =_B t(x))$, with inclusion into A given by the first projection.

   It is a well known fact that terminal object, binary products and equalizers suffices to have all finite limits.

3. This does in fact follow from the existence of terminal object and being locally cartesian closed, since $\mathbf{C}(\mathbf{M}) \cong \mathbf{C}(\mathbf{M})/1$. However it just as easy to see that the exponential object B^A is just the type $A \to B$. The isomorphism $A \times C \to B \cong C \to B^A$ is given by $t : A \times C \to B$ being sent to $\lambda y : C.\lambda x : A.t(x, y) : C \to B^A$, and $s : C \to B^A$ being sent to $\lambda z : A \times B.s(pr_2(z))(pr_1(z)) : A \times C \to B$. That these are inverses is a standard result.

4. This is theorem 3.2 in [See84], so we will not repeat it here since the method used there is somewhat convoluted and of no immediate interest.

□

There are other interesting ways to model type theory in categories which are symmetric monoidal closed using linear type theory following [Lam68] and [Lam69]. This removes the need to require cartesian closed, which not all closed monoidal categories satisfy, and which perhaps is not necessary to model type theory. Unfortunately univalence does not seem to have been properly explored in this model, which would be a very interesting direction to explore.

7 Homotopy theory in type-theoretic fibration categories

If we have a type-theoretic fibration category we can define quite a few of the notions of homotopy theory and get that any type-theoretic fibration category is in fact a category of fibrant objects. In this section we will follow [Uni13] and [Shu12]

7.1 In the type theory

In this subsection we will work exclusively in the dependent type theory given in chapter 5. The first thing we note is that identity types behave like groupoid thanks to the induction rule. That is whenever we have $p : a =_A b$ and $q : b =_A c$, for some $a, b : A$ we have also $p^{-1} : b =_A a$ and $q \circ p : a =_A c$. More formally we have the following lemma.

Lemma 7.1. *The following types are inhabited:*

1. $\prod_{A:\mathcal{U}} \prod_{a,b:A} (a =_A b) \to (b =_A a)$
2. $\prod_{A:\mathcal{U}} \prod_{a,b,c:A} (a =_A b) \to (b =_A c) \to (a =_A c)$

Proof. We will only give the formal proof in the first case, and then give an informal proof of both cases. This is because the induction rule 3 for equality is quite complex, and informally is just says that we can replace any path by reflexivity.

1. $\lambda A.\lambda a.\lambda b.\lambda p.ind_{=_A}((\lambda x.\lambda y.\lambda p.(b =_A a)), (\lambda a.refl_a), a, b, p) :$
   $\prod_{A:\mathcal{U}} \prod_{a,b:A} (a =_A b) \to (b =_A a)$

□

Arguably the informal proof is more plain to read and more informative

1. Given the information $a, b : A$ and $p : a =_A b$, we want to construct a term p^{-1} of the type $b =_A a$. With induction it suffices to do this in the case when $b \equiv a$ and $p \equiv refl_a$, in which case the type we want to construct a term of is $a =_A a$. In this case we have the obvious choice of choosing $refl_a^{-1} = refl_a$. The rest follows from the induction principle.
2. Given $a, b : A$, $p : a =_A b$ and $q : b =_A c$, we want to construct a term $p \cdot q : a =_A c$. By induction of p it suffices to consider the case when $b \equiv a$ and $p \equiv refl_a$, in which case the type of q becomes $a =_A c$. Now we could have stopped here and chosen $refl_a \cdot q \equiv q$, but for symmetry reasons we do induction of q also. Now we have $c \equiv a$ and $q \equiv refl_a$, and we can choose $refl_a \cdot refl_a \equiv refl_a$.

Of course we also need to check that these paths (i.e. terms of the identity types) satisfy the groupoid axioms. By which we mean that we want paths of paths (i.e. homotopies) giving associativity, identity, and inverse as follows. For the rest of the theory we will only give informal proofs, and we will suppress all subscripts on identity types for readability.

Lemma 7.2. *Suppose we are given $A : \mathcal{U}$, $a, b, c, d : A$, $p : a = b$, $q : b = c$ and $r : c = d$. Then:*

1. $p = refl_a \cdot p$ *and* $p = p \cdot refl_b$.
2. $p \cdot p^{-1} = refl_a$ *and* $p^{-1} \cdot p = refl_b$.
3. $(p^{-1})^{-1} = p$.
4. $(p \cdot q) \cdot r = p \cdot (q \cdot r)$.

Proof. All of these are a simple application of induction which reduces the lemma simply to the computation rules of the definition of *cdot* and $^{-1}$. In all cases we only have to prove $refl_a = refl_a$ which we inhabit by $refl_{refl_a}$, showing that we are indeed dealing with paths of paths here. □

Thanks to induction we can prove that any function is continuous in the sense that it preserve paths. It can also be said to act functorially when view from a more categorical perspective.

Lemma 7.3. *Suppose $f : A \to B$ is a function and $a, b : A$, then we have a term of the type:*

$$ap_f : (a =_A b) \to (f(a) =_B f(b)$$

Proof. By induction it suffices to assume $p : a =_A b$ is $refl_a$. And then we define $ap_f(p) \equiv refl_{f(a)} : f(a) =_B f(a)$, and then use introduction rule for function type. Finally we lift it to the general case with the induction. □

This if course has all the properties we would expect, and we will only list them below without proof. Which in all cases is only induction and reflexivity anyways.

Lemma 7.4. *Given $f : A \to B$, $g : B \to C$, $p : a =_A b$ and $q : b =_A y$, we have:*

1. $ap_f(p \cdot q) = ap_f(p) \cdot ap_f(q)$.
2. $ap_f(p^{-1}) = (ap_f(p))^{-1}$.
3. $ap_g(ap_f(p)) = ap_{g \circ f}(p)$.
4. $ap_{id_A}(p) = p$.

One final property we want to mention that comes from the induction principle for identity types is the following.

Lemma 7.5. *Suppose that P is a type family over A, and that $p : a =_A b$. Then there is a term $p_* : P(a) \to P(b)$.*

Proof. By induction we can assume that $p \equiv refl_a$ and then we define $p_* \equiv id_{P(a)} : P(a) \to P(a)$ □

This has the following very important property:

Lemma 7.6. *Let $P : A \to \mathcal{U}$ be a type family and $u : P(a)$ for some $a : A$. Then for any $p : a = b$ we have a term*

$$lift(u, p) : (a, u) = (b, p_*(u))$$

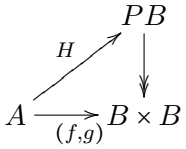
in $\sum_{(a:A)} P(a)$

Which tells us that dependent type behave like fibrations $pr_1 : \sum_{(a:A)} P(a) \twoheadrightarrow A$, because they have the path lifting property required of fibrations. With this knowledge we look at how we can define some of the homotopy theory in the categorical model of type-theoretic fibration category

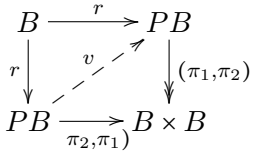
7.2 In the categorical model

In this subsection we are working in a type-theoretic fibration category $\mathbf{C}$, and all objects and morphisms are assumed to be in $\mathbf{C}$. By 5 of definition 6.7 in chapter 6 we have a factorization of the diagonal morphism as $B \tilde{\rightarrowtail} PB \twoheadrightarrow B \times B$ and we call the object PB the path object of B. We know from classical homotopy theory that we can define right homotopies with the help of path objects, without the need for an interval object which is out of reach in a type-theoretic fibration category.

We define a (right) homotopy between $f, g : A \rightrightarrows B$ to be a lift of $A \to B \times B$ to a path object for B. That is a diagram



We denote a homotopy by $H : f \sim g$. Strictly speaking this depends on a choice of path object for B, but since the map $B \tilde{\rightarrowtail} PB$ is always an acyclic cofibration any path object factors through any other so the homotopy relation is independent on any choice. There is a map defined in Lemma 2.4 of [Shu12] called c, which is such that if $H : f \sim g$ and $K : g \sim h$, then $c(H, K) : f \sim h$, so it concatenates homotopies. The map $B \tilde{\rightarrowtail} PB$ is usually just called r (since it is the reflexivity term from the identity type), and we easily see that $rf : f \sim f$. Finally the definition of $\sim$ is obviously symmetric, but we could also see it by considering the following diagram, where we denoted the map $PB \twoheadrightarrow B \times B$ by (π_1, π_2):



We define a map $f : A \to B$ to be a homotopy equivalence if there is a map $g : B \to A$ and homotopies $gf \sim id_A$ and $fg \sim id_B$. Now we can define the homotopy equivalences to be the weak equivalences $w\mathbf{C}$ and note that they satisfy the following properties

- $w\mathbf{C}$ contains all the isomorphisms.
- $w\mathbf{C}$ satisfy the 2-out-of-3 property, i.e. if any two of f, g, or gf are in $w\mathbf{C}$ then so is the third.
- Any diagonal $B \to B \times B$ factors as a map in $w\mathbf{C}$ followed by a fibration.
- Any pullback of a fibration in $w\mathbf{C}$ is also in $w\mathbf{C}$.

The proof of all of these can be found in [Shu12]. These properties, together with 1 - 3 of Definition 6.7 make any type-theoretic fibration category into a category of fibrant objects in the sense of [Bro73], where the weak equivalences are the homotopy equivalences.

7.3 Some more words on equivalences in type theory

We have a notion of homotopy in our type theory also. Let $f, g : \prod_{x:A} P(x)$ be two dependent functions. We define a homotopy $f \sim g$ in the following way:

$$f \sim g :\equiv \prod_{x:A} (f(x) = g(x)).$$

This can be shown to be an equivalence relation, using familiar properties of the identity type. Normally one would be tempted to call this a proof of equality between functions, as is the case in set theory, but there are no rules in the type theory so far that allow us to deduce equality of the two function from a proof of homotopy. There is an axiom we can add to our theory which says precisely that, called function extensionality, giving an equivalence of types. Before we can state this axiom though we have to define precisely what we mean by two types being equivalent.

There are multiple ways of defining equivalences, and large parts of chapter 4 of [Uni13] discusses the differences and the way they are all equivalent. We will choose the easiest to state which has all the properties we will need. Let $f : A \to B$, then we have a type called

$$isequiv(f) :\equiv \left(\sum_{g:B\to A} (f \circ g \sim id_B) \right) \times \left(\sum_{h:B\to A} (h \circ f \sim id_A) \right)$$

That is we have a right inverse g of f, such that the composition is homotopic to the identity, and likewise a left inverse h. Now we say two types are equivalent if the exists a function between them that is an equivalence. We use the following notation:

$$A \simeq B :\equiv \sum_{f:A\to B} isequiv(f)$$

Now we can define the function extensionality axiom

Axiom 7.7 (Function extensionality axiom) *For any A, B, f and g we have*

$$(f = g) \simeq (f \sim g).$$

We can note that we easily know of a map $(f = g) \to (f \sim g)$, namely by induction on $f = g$ it suffices to define it in the case $g \equiv f$ and the path begin reflexivity, in which case take $\lambda x.refl_{f(x)} : \prod_{x:A}(f(x) = f(x))$. Function extensionality is a very nice axiom to have included in our type theory as it simplifies a lot of computation, and also makes possible to prove nice new theorems we would like. There is however an even nicer axiom we will focus more on, and which is well known to imply function extensionality. This theorem is called the univalence axiom.

To define univalence we need a universe type $\mathcal{U}$, such that all other types are elements of this type. This is something we already assumed in chapter 5, but since some authors do not use this terminology from the start we remark upon it here. It is in fact possible to define everything up to univalence without a type of types, so if we were interested in studying dependent type theories without univalence we could do with a weaker model. However having a type of types enables us, for any $A, B : \mathcal{U}$, to define the type $A =_{\mathcal{U}} B$. Now we have a presumably even more strong notion of two types being equivalent and we are at conflict upon which to use. This is however where the univalence axiom of Voevodsky makes it appearance:

Axiom 7.8 (Univalence axiom) *For any* $A, B : \mathcal{U}$ *we have*

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B).$$

Again the function $(A =_{\mathcal{U}} B) \to (A \simeq B)$ is quite easy to construct by doing induction of $A = B$, and then defining $id_A : A \to A$, which is easily seen to be an equivalence in the sense of $isequiv(id_A)$ being inhabited.

Since univalence actually depend on a choice of universe $\mathcal{U}$, the axiom does in fact state that the universe $\mathcal{U}$ is univalent. What one usually does is assume that all universes in the type theory are univalent.

8 Monoids in monoidal categories

We will now recall some facts about the category **SMon** of Symmetric Monoids. The most basic facts can be found in Kelly's book [Kel82] chapter 1, or Dundas, Goodwillie, McCarthy [DGM13] Appendix A.9.

8.1 Monoidal category

A monoidal category $\mathcal{V} = (\mathcal{V}_0, \otimes, I, \alpha, l, r)$ consists of a category $\mathcal{V}_0$, a functor $\otimes : \mathcal{V}_0 \times \mathcal{V}_0 \to \mathcal{V}_0$, and object $I \in \mathcal{V}$, and three natural isomorphisms $a_{XYZ} : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$, $l_X : I \otimes X \to X$, $r_X : X \otimes I \to X$, such that the following two diagrams commute:

$$\begin{array}{ccccc} ((W \otimes X) \otimes Y) \otimes Z & \xrightarrow{a} & (W \otimes X) \otimes (Y \otimes Z) & \xrightarrow{a} & W \otimes (X \otimes (Y \otimes Z)) \\ \downarrow{\scriptstyle 1\otimes a} & & & & \uparrow{\scriptstyle a\otimes 1} \\ (W \otimes (X \otimes Y)) \otimes Z & & \xrightarrow[a]{} & & W \otimes ((X \otimes Y) \otimes Z) \end{array} , \tag{8.1}$$

$$\begin{array}{ccc} (X \otimes I) \otimes Y) & \xrightarrow{a} & X \otimes (I \otimes Y) \\ & {\scriptstyle r\otimes 1}\searrow \quad \swarrow{\scriptstyle 1\otimes l} & \\ & X \otimes Y & \end{array} . \tag{8.2}$$

We will be most interested in the case where $\mathcal{V}_0$ is the category **Set** of small sets, or **sSet** of simplicial sets. We also have the representable functor $U := \mathcal{V}(I, -) : \mathcal{V}_0 \to \mathbf{Set}$, we denote it by U because in many cases it is isomorphic to the normal forgetful functor. A monoidal category is furthermore called symmetric if there in addition is a natural isomorphism $c_{XY} : X \otimes Y \to Y \otimes X$ such that the following diagrams commute:

$$\begin{array}{ccc} X \otimes Y & \xrightarrow{c} & Y \otimes X \\ & {\scriptstyle 1}\searrow \quad \downarrow{\scriptstyle c} & \\ & X \otimes Y & \end{array} \tag{8.3}$$

$$\begin{array}{ccccc} (X \otimes Y) \otimes Z & \xrightarrow{a} & X \otimes (Y \otimes Z) & \xrightarrow{c} & (Y \otimes Z) \otimes X \\ \downarrow{\scriptstyle c\otimes 1} & & & & \downarrow{\scriptstyle a} \\ (Y \otimes X) \otimes Z & \xrightarrow[a]{} & Y \otimes (X \otimes Z) & \xrightarrow[1\otimes c]{} & Y \otimes (Z \otimes X) \end{array} \tag{8.4}$$

$$\begin{array}{ccc} X \otimes I & \xrightarrow{c} & I \otimes X \\ & {\scriptstyle r}\searrow \quad \swarrow{\scriptstyle l} & \\ & X & \end{array} \tag{8.5}$$

Note that 8.5 defines r in terms of l and c so we really only need one of r or l in the definition of a symmetric monoidal category. There is one more important property we can demand of a monoidal category (and one which both our standard examples satisfy), namely being closed. If the tensor product is the usual cartesian product $\otimes = \times$, and the unit is the terminal object $I = *$ (in which case the category is called cartesian monoidal), being closed coincides with the usual notion of a cartesian closed category.

Definition 8.1. *A monoidal category is closed if each functor $- \otimes Y : \mathcal{V}_0 \to \mathcal{V}_0$ has a right adjoint $\underline{\mathcal{V}}(Y, -)$; so we have a natural isomorphism*

$$\mathcal{V}(X \otimes Y, Z) \cong \mathcal{V}(X, \underline{\mathcal{V}}(Y, Z)) \tag{8.6}$$

Note two important special cases, if X is replaces by I or $W \otimes X$. In the first case we get an isomorphism

$$\mathcal{V}(Y, Z) \cong \mathcal{V}(I \otimes Y, Z) \cong \mathcal{V}(I, \underline{\mathcal{V}}(Y, Z) = U(\underline{\mathcal{V}}(Y, Z)).$$

So that $\underline{\mathcal{V}}(Y, Z)$ normally called the internal hom-object, since its underlying set is the normal hom-set $\mathcal{V}(Y, Z)$, and it is an internal object in the monoidal category. In the second case we get the isomorphisms:

$$\begin{aligned}\mathcal{V}(W, \underline{\mathcal{V}}(X, \underline{\mathcal{V}}(Y, Z))) \cong& \mathcal{V}(W \otimes X, \underline{\mathcal{V}}(Y, Z)) \\ \cong& \mathcal{V}((W \otimes X) \otimes Y, Z) \\ \cong& \mathcal{V}(W \otimes (X \otimes Y), Z) \\ \cong& \mathcal{V}(W, \underline{\mathcal{V}}(W \otimes Y, Z)).\end{aligned}$$

So we also get an isomorphism of internal hom-objects:

$$\underline{\mathcal{V}}(W \otimes Y, Z) \cong \underline{\mathcal{V}}(X, \underline{\mathcal{V}}(Y, Z)). \tag{8.7}$$

The following is taken from [Por08].

If we have a monoidal category $\mathcal{V}$ we can form the category of monoids $\mathbf{Mon}\mathcal{V}$, with objects triples $(C, C \otimes C \xrightarrow{m} C, I \xrightarrow{e} C)$, with $C \in \mathcal{V}$. We further require our operation m to be associative, and the unit $e(I)$ to be a unit of the operation, by requiring the following diagrams to commute:

$$\begin{array}{ccc} C \otimes C \otimes C & \xrightarrow{1 \otimes m} & C \otimes C \\ {\scriptstyle m \otimes 1}\downarrow & & \downarrow{\scriptstyle m} \\ C \otimes C & \xrightarrow[m]{} & C \end{array}$$

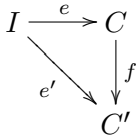
$$\begin{array}{ccccc} C \otimes I & \xrightarrow{1 \otimes e} & C \otimes C & \xleftarrow{e \otimes 1} & I \otimes C \\ & \searrow & \downarrow{\scriptstyle m} & \swarrow & \\ & & C & & \end{array}$$

If we also have that the following diagram commutes, we call the monoid a symmetric monoid, and we use the notation $\mathbf{SMon}\mathcal{V}$ for the category of symmetric monoids in $\mathcal{V}$:

$$\begin{array}{ccc} C \otimes C & \xrightarrow{c} & C \otimes C \\ & {\scriptstyle m}\searrow & \downarrow{\scriptstyle m} \\ & & C \end{array}$$

A monoid morphism $(C, m, e) \to (C', m', e')$ is then a morphism $f : C \to C'$ such that the following diagrams commute:

$$\begin{array}{ccc} C \otimes C & \xrightarrow{m} & C \\ {\scriptstyle f \otimes f}\downarrow & & \downarrow{\scriptstyle f} \\ C' \otimes C' & \xrightarrow[m']{} & C' \end{array}$$



Thanks to the rich structure $\mathbf{Mon}\mathcal{V}$ inherits from being a category of models of an algebraic theory, we know: (example 5.2.2) [Bor94]

Lemma 8.2. *The category* $\mathbf{Mon}\mathcal{V}$ *is locally finitely presentable.*

So as a corollary we get:

Corollary 8.3. $\mathbf{Mon}\mathcal{V}$ *is both complete and cocomplete.*

For definition and more results about locally finitely presentable categories and algebraic theories we direct the reader to the above reference.

There is a standard way of lifting the tensor product $\otimes$ from $\mathcal{V}$ to a tensor product $\oplus$ in $\mathbf{Mon}\mathcal{V}$, simply define:

$$(C_1, m_1, e_1) \oplus (C_2, m_2, e_2) := (C_1 \otimes C_2, m, e),$$

where

$$m : (C_1 \otimes C_2) \otimes (C_1 \otimes C_2) \xrightarrow{1 \otimes c \otimes 1} (C_1 \otimes C_1) \otimes (C_2 \otimes C_2) \xrightarrow{m_1 \otimes m_2} C_1 \otimes C_2$$

$$e : I \cong I \otimes I \xrightarrow{e_1 \otimes e_2} C_1 \otimes C_2.$$

We suppress all associativity from these formulas, because from a well know result [ML98] XI.3 we know that any monoidal category is equivalent to a strict monoidal category (one in which all the natural isomorphisms are identity), so there is always a coherent choice of which natural isomorphism to choose. Also any symmetric monoidal category can be strictified to a permutative category (also known as a strict symmetric monoidal category), as show in [Isb69]. This tensor product is simply the pointwise tensor product coming from $\mathcal{V}$, and this makes $\mathbf{Mon}\mathcal{V}$ into a symmetric monoidal category, and does give the coproduct in $\mathbf{SMon}\mathcal{V}$:

$$(C_1, m_1, e_1) \xrightarrow{i_1} (C_1, m_1, e_1) \oplus (C_2, m_2, e_2) \xleftarrow{i_2} (C_2, m_2, e_2)$$

$$i_1 : C_1 \cong C_1 \otimes I \xrightarrow{1 \otimes e} C_1 \otimes C_2$$

$$i_2 : C_2 \cong I \otimes C_2 \xrightarrow{e \otimes 1} C_1 \otimes C_2.$$

We will now look at a different monoidal product on **Set** and show that it gives the structure of a closed symmetric monoidal category.

8.2 The category of symmetric monoids in sets

We remember that the category of small sets **Set** with cartesian product $\times$, and the terminal object $*$ as tensor product and unit, makes **Set** into a (cartesian) closed symmetric monoidal category (where the natural isomorphisms a, r, l, c are suppressed). Then it is possible to construct the category of symmetric monoids in **Set**, which we will call **SMon**: with objects triples $(C, C \times C \xrightarrow{m} C, * \xrightarrow{e} C)$, with $C \in \mathbf{Set}$ and $m(a, b) = a + b$, for $a, b \in C$ and $e(*) = 1_C$.

Now we will take some care to construct the tensor product to be such that **SMon** is a closed symmetric monoidal category, with the internal hom-object coming from the fact that **Set** is symmetric monoidal.

The forgetful functor $U : \mathbf{SMon} \to \mathbf{Mon} \to \mathbf{Set}$ has a left adjoint $Sym : \mathbf{Set} \xrightarrow{(-)^*} \mathbf{Mon} \xrightarrow{q} \mathbf{SMon}$, which we will use the next few paragraphs to define. If we start with a set A, we can construct the free monoid A^* as the set of finite words with letters in A including the empty word, defined recursively as

- $\varnothing \in A^*$
- $al \in A^*$, if $a \in A$ and $l \in A^*$.

In this way each word is a finite string of elements $a \in A$, and we call these elements the letters of the word. If $\varnothing$ appears anywhere in the word we suppress it, unless it appears just by itself. The monoid operation is concatenation of words, written $a \cdot b = ab$ if a and b are words, defined recursively in a

- $\varnothing \cdot b = b$
- $(cl) \cdot b = c(l \cdot b)$, if $a = cl$ with $c \in A$ and $l \in A^*$.

Let us look at an example:

Example 8.4. If $S = \{a, b\}$ is a set of two elements, we have a monoid S^*. Some of the words in S^* include $s_0 = \varnothing$, $s_1 = a$, $s_2 = b$, $s_3 = aa$, $s_4 = ab$, $s_5 = bb$, $s_6 = aaabbbaabbabaaab$, and infinitively many more. Concatenation is simply written as $s_1 \cdot s_2 = a \cdot b = ab = s_4$, $s_2 \cdot s_1 = ba$, or $s_0 \cdot s_5 = \varnothing \cdot bb = bb = s_5$. So we see that the empty word always takes the role of the identity, and that the monoid is in general not symmetric.

This free construction makes A^* into a monoid with concatenation as operation and the empty word as identity. Alternatively we can define $A^* = \coprod_{n \geq 0} A^{\times n}$. Also if we have a function $f : A \to B$, we get a function $f^* : A^* \to B^*$ defined recursively as:

- $f(\varnothing) = \varnothing$
- $f(al) = f(a)f(l)$, for $a \in A$ and $l \in A^*$.

This map is by definition a monoid map, so there is nothing really to check there. This makes $(-)^* : \mathbf{Set} \to \mathbf{Mon}$ into a functor, which is left adjoint to the forgetful functor $U : \mathbf{Mon} \to \mathbf{Set}$. As we saw in example 8.4 the monoid product is not symmetric. There is however a way to rectify this by disregarding the order of the letters in the words. More precisely we have an action of the symmetric group of order n on words of length n (that is, has n letters). The length of a word is defined recursively as:

- $length(\varnothing) = 0$
- $length(al) = 1 + length(l)$, where $a \in A$ and $l \in A^*$.

This completes the definition of the map $(-)^* : \mathbf{Set} \to \mathbf{Mon}$, and we now define the map $q : \mathbf{Mon} \to \mathbf{SMon}$ by taking quotient of equivalence classes. If we now have a word w of length n, it is of the form $w = a_1 a_2 \dots a_n$ with $a_i \in A$ for $i = 1, 2, \dots n$. The action is then given by $\sigma \cdot (a_1 a_2 \dots a_n) = a_{\sigma^{-1}(1)} a_{\sigma^{-1}(2)} \dots a_{\sigma^{-1}(n)}$ for $\sigma \in \Sigma_n$. We easily see that $id \cdot w = w$ and $\tau \cdot (\sigma \cdot w) = (\tau\sigma) \cdot w$, so it is indeed an action. As we remember from group theory every action of a group gives an equivalence relation on the set, so we can quotient out by this relation to get a new set. However we also see that concatenation plays nicely together with

the equivalence classes, since $\overline{v} \cdot \overline{w} = \overline{vw}$ where $\overline{v}$ denotes the equivalence class of v in the quotient. So the quotient is in fact a monoid under concatenation, and it should be quite evident that it has now been turned into a symmetric operation so we have achieved our goal. That is, if A is a set, we get:

$$SymA := \coprod_{n\in\mathbb{N}} (A^{\times n}/\Sigma_n).$$

Since this is a symmetric monoid, we will use + for the monoid operation.

We will now construct an internal hom-object, and show that it has a left adjoint, giving **SMon** the structure of a closed monoidal category. The main ideas here are taken from the notes of Harold Simmons [Sim]. For any $B, C \in \mathbf{SMon}$, we have the set of morphisms $\mathbf{SMon}(B,C)$, and this can be given the pointwise symmetric monoid structure which we denote $\underline{\mathbf{SMon}}(B,C)$. For any $f,g \in \mathbf{SMon}(B,C)$, we define $f+g : B \to C$ by $(f+g)b = (fb) + (gb)$, and an identity element to be the constant function $id(b) = 1_C$. This is a symmetric monoid structure since

$$\begin{aligned}(f+g)(b_1+b_2) &= f(b_1+b_2) + g(b_1+b_2)\\ &= f(b_1) + f(b_2) + g(b_1) + g(b_2)\\ &= f(b_1) + g(b_1) + f(b_2) + g(b_2)\\ &= ((f+g)(b_1) + (f+g)(b_2)),\end{aligned}$$

for any $f,g : B \to C$ and $b_1, b_2 \in B$, and it is clearly symmetric by the same calculation.

This construction is functorial in the following sense: let $h : B_2 \to B_1$ and $l : C_1 \to C_2$ be morphisms of monoids. Then we have pre- (denoted by an upper star) and post composition (denoted by a lower star) fitting in a diagram of the form:

$$\begin{array}{ccc}\mathbf{SMon}(B_1,C_1) & \xrightarrow{l_*} & \mathbf{SMon}(B_1,C_2)\ ,\\ {\scriptstyle h^*}\downarrow & & \downarrow{\scriptstyle h^*}\\ \mathbf{SMon}(B_2,C_1) & \xrightarrow{l_*} & \mathbf{SMon}(B_2,C_2)\end{array}$$

which commutes since composition is associative $h^* l_* g = l \circ g \circ h = l_* h^* g$, for any $g : B_1 \to C_1$. We also note that these morphisms are actually in **SMon**,

$$h^*(g_1+g_2)b = (g_1+g_2)(hb) = ((g_1(h)b)) + ((g_2(h)b)) = (h^* g_1 b) + (h^* g_2 b)$$

$$l_*(f_1+f_2)b = l((f_1+f_2)b) = l(f_1 b + f_2 b) = (l(f_1 b)) + (l(f_2 b)) = (l_* f_1 b) + (l_* f_2 b).$$

All this shows that $\underline{\mathbf{SMon}}(-,-) : \mathbf{SMon}^{op} \times \mathbf{SMon} \to \mathbf{SMon}$ is a functor. This gives an endofunctor $\underline{\mathbf{SMon}}(B,-) : \mathbf{SMon} \to \mathbf{SMon}$, and we want to show that this has a left adjoint $- \otimes B : \mathbf{SMon} \to \mathbf{SMon}$, for all $B \in \mathbf{SMon}$.

We construct the monoid $A \otimes B$ as the bilinearization of $A \times B$, which is the same construction used to make the tensor product for abelian groups, which fortunately works in our case also. More concretely we form the free monoid $Sym(A \times B)$ on the underlying set $A \times B$ (remember that $\times$ is the coproduct in **SMon** so $A \times B$ is in fact a symmetric monoid, with the pointwise symmetric monoid structure from A and B). Then we divide out by the equivalence relation generated by the following relations:

$$\begin{aligned}(a_1 + a_2, b_1) &\sim (a_1, b_1) + (a_2, b_1)\\ (a_1, b_1 + b_2) &\sim (a_1, b_1) + (a_1, b_2)\\ (1_A, b_1) &\sim (1_A, 1_B) \sim (a_1, 1_B),\end{aligned}$$

for all $a_1, a_2 \in A$ and $b_1, b_2 \in B$. We denote the equivalence class of the pair $(a, b) \in A \otimes B$ by $a \otimes b$. This does in fact give $A \otimes B$ the structure of a symmetric monoid, with operation

$$\left(\sum_{a\otimes b\in A\otimes B} n_{a\otimes b} a \otimes b\right) + \left(\sum_{a\otimes b\in A\otimes B} m_{a\otimes b} a \otimes b\right) = \sum_{a\otimes b\in A\otimes B} (n_{a\otimes b} + m_{a\otimes b}) a \otimes b$$

with $n_{a\otimes b}, m_{a\otimes b} \geq 0$ denoting the number of appearances of $a \otimes b$ in the word, and identity $1_A \otimes 1_B$.

We then have an adjunction, given by the pair:

$$\begin{array}{ccc} & f \longmapsto & f^\sharp \\ \mathbf{SMon}(A \otimes B, C) \cong & & \mathbf{SMon}(A, \underline{\mathbf{SMon}}(B, C)) \\ & g_\flat \longleftarrow & g \end{array}$$

with the defining equations

$$f^\sharp ab = f(a \otimes b) \qquad g_\flat(a \otimes b) = gab,$$

for $a \in A, b \in B$. The required naturality conditions now follow directly from the construction, and the two functions are easily seen to be inverses of each other. By defining $(f_1 + f_2)^\sharp = f_1^\sharp + f_2^\sharp$ and $(g_1 + g_2)_\flat = (g_1)_\flat + (g_2)_\flat$, for $f_1, f_2 \in \mathbf{SMon}(A \otimes B, C)$ and $g_1, g_2 \in \mathbf{SMon}(A, \underline{\mathbf{SMon}}(B, C))$, we get also:

$$\underline{\mathbf{SMon}}(A \otimes B, C) \cong \underline{\mathbf{SMon}}(A, \underline{\mathbf{SMon}}(B, C))$$

this is of course a completely general fact, as we noticed in 8.7, so there is really nothing to check.

9 Simplicial models

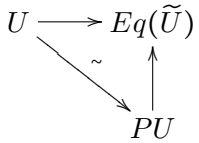
We will give a short exposition on Voevodsky's model of a univalent universe in simplicial sets based on [KLV12]. Then we explain some of the things which also work in simplicial symmetric monoids, and some of the things which do not.

9.1 Simplicial sets

In simplicial sets $\mathcal{S}$ we have representable objects $\Delta[n]$, given by $\Delta[n]_m = \Delta([m],[n]$, such that for any simplicial set X we have $\mathcal{S}(\Delta[n], X) \cong X_n$. Mimicking the idea that we have a universe in presheaf categories given by $U(X) = \{Y | ftY = X\}$ from lemma 6.14, we define a presheaf universe $\mathbf{U}_\alpha$ such that $\mathbf{U}_\alpha(X)$ is the set of well-ordered morphisms $Y \twoheadrightarrow X$ where each fiber Y_x has cardinality $< \alpha$. We will not use any other cardinality than α, so we just use the notation $\mathbf{U} := \mathbf{U}_\alpha$. Then we can prove that $\mathbf{U}$ is representable, and is represented by the simplicial set U given by $U_n := \mathbf{U}(\Delta[n])$. And we further get another simplicial set $\widetilde{U}$ with a fibration $\pi : \widetilde{U} \twoheadrightarrow U$ which is universal among well-ordered fibrations with fibers of cardinality $< \alpha$, and weakly universal among all fibrations of cardinality $< \alpha$. Further it can be proved that the simplicial set U is fibrant, by using minimal fibrations and properties of the left and right adjoints of the pullback.

Now we have a fibrant universe representing all fibrations in the category, it only remains to show that the universe is univalent. We then want an object representing weak equivalences between fibrations $E_1 \twoheadrightarrow B$, $E_2 \twoheadrightarrow B$ over a common base, i.e. types in a context. We already know that morphisms between fibrations over a common base can be represented since $\mathcal{S}$ is locally cartesian closed, so we have a internal "hom-object" $Hom(E_1, E_2) \to B$, and since $\mathcal{S}$ is right proper this morphism is also a fibration. Now [KLV12] show that there is a simplicial subset $Eq_B(E_1, E_2)$ which represents weak equivalences as we desired, and which is also fibrant in $\mathcal{S}/B$.

The final step is now to show that the universe U is univalent, which is a question of a canonical map $PU \to Eq(\widetilde{U})$ being a weak equivalence, i.e. that $Eq(\widetilde{U})$ can be seen as a path object for the universe U (remember that path objects are the models for identity types, so this is in fact equivalent to the univalence axiom from type theory). The object PU is a path object for U, i.e. a factorization of the diagonal as $U \xrightarrow{\sim} PU \twoheadrightarrow U \times U$, and $Eq(\widetilde{U}) := Eq_{U\times U}(pr_1^*\widetilde{U}, pr_2^*\widetilde{U})$, with $pr_1, pr_2 : U \times U$ the two projections. Since weak equivalences have the 2-out-of-3 property, this is equivalent to the map $U \to Eq(\widetilde{U})$ in the following diagram in $\mathcal{S}/U \times U$



being a weak equivalence. The map $U \to Eq(\widetilde{U})$ is essentially given by a diagonal map. [KLV12] does succeed in showing that the universe U is univalent and hence constructing the first good model for homotopy type theory.

9.2 Simplicial symmetric monoids

We will be working in the category of simplicial symmetric monoids $s\mathbf{SMon} := \mathbf{Cat}(\Delta^{op}, \mathbf{SMon})$. We can give $s\mathbf{SMon}$ the structure of a model category by lifting the model structure from simplicial sets through the adjunction $Sym : \mathbf{Set} \rightleftarrows \mathbf{SMon} : U$ by using the transfer principle 4.10.

Lemma 9.1. *$s\mathbf{SMon}$ is a simplicial model category, with $A \to B$ a weak equivalence (or fibration) iff $UA = s\mathbf{SMon}(*, A) \to s\mathbf{SMon}(*, B) = UB$ is a weak equivalence (or fibration) of simplicial sets.*

Proof. This is a standard argument from [GJ99] II.6 (Cor 6.6 in fact). □

We want to construct a universe representing all α-small fibrations in some sense, so we try to mimic the construction in $\mathcal{S}$ from [KLV12].

Definition 9.2. *Let $Sym\Delta[n]$ be the simplicial monoid given as:*

$$(Sym\Delta[n])[m] := Sym(\Delta[n]_m) = Sym(\Delta([m], [n])),$$

Lemma 9.3. *Since the category of simplicial symmetric monoids $s\mathbf{SMon}$ is an enriched presheaf category we know that every object is given as a colimit of the representable functors $Sym\Delta[n]$. See Mac Lane III.7 [ML98].*

Because of this Lemma it gives some credibility to the construction of simplicial monoids over the special objects $Sym\Delta[n]$, so we are somewhat justified in giving the following definition which is the same as in [KLV12].

Definition 9.4. *U is the simplicial set, which will be give a symmetric monoid structure, given as*

$$U := \left\{ [n] \mapsto \left[Y \xrightarrow{f} Sym\Delta[n] \right] \right\},$$

with Y a simplicial monoid and f a fibration, with fiber Y_x of cardinality $< \alpha$ for each $x \in Sym\Delta[n]$, and $[-]$ meaning isomorphism classes of maps (where the isomorphisms are to be taken in the over category $\mathbf{SMon}/Sym\Delta[n]$). If we have a map $[n'] \to [n]$ in Δ, we get an induced map $U_n \to U_{n'}$ given by pullback along the induced map $Sym\Delta[n] \to Sym\Delta[n']$. The monoid structure is given by the pullback over the common base:

$$[Y \to Sym\Delta[n]] * \left[Y' \to Sym\Delta[n]\right] = \left[Y \times_{Sym\Delta[n]} Y' \to Sym\Delta[n]\right].$$

This gives U_n the structure of a symmetric monoid for all n, with the identity $(Sym\Delta[n] = Sym\Delta[n]$ as the unit. Standard isomorphisms for pullback give associativity and identity since we are taking isomorphism classes of maps over $Sym\Delta[n]$.

We have by the Yoneda embedding for **SMon**-enriched functors a one to one correspondence:

$$U_n \cong s\mathbf{SMon}(Sym\Delta[n], U),$$

where we will use the notation $\ulcorner f \urcorner : Sym\Delta[n] \to U$ for the map that $\left[Y \xrightarrow{f} Sym\Delta[n] \right] \in U_n$ corresponds to.

Definition 9.5. *Let $\widetilde{U}$ be the simplicial symmetric monoid given as*

$$\widetilde{U}_n := \left\{ \begin{bmatrix} & & Y \\ & \overset{s}{\nearrow} & \downarrow f \\ Sym\Delta[n] & = & Sym\Delta[n] \end{bmatrix} \right\}$$

the set of isomorphism classes of diagrams of the above form, such that s is a section of the fibration $f \in U_n$. The monoid structure is given by

$$\begin{bmatrix} & & Y \\ & \overset{s}{\nearrow} & \downarrow f \\ Sym\Delta[n] & = & Sym\Delta[n] \end{bmatrix} * \begin{bmatrix} & & Y' \\ & \overset{s'}{\nearrow} & \downarrow f' \\ Sym\Delta[n] & = & Sym\Delta[n] \end{bmatrix} = \begin{bmatrix} & & Y \times_{Sym\Delta[n]} Y' \\ & \nearrow & \downarrow \\ Sym\Delta[n] & = & Sym\Delta[n] \end{bmatrix},$$

where the dotted line is the unique induced map coming from the pullback, since we have a map $s : Sym\Delta[n] \to Y$ and a map $s' : Sym\Delta[n] \to Y'$, which both agree when mapping back to $Sym\Delta[n]$ since they are both sent to the identity.

We have a map $\pi : \widetilde{U} \to U$ given by forgetting the section, and:

Lemma 9.6. *$\pi : \widetilde{U} \to U$ is a fibration.*

Proof. We look at squares:

$$\begin{array}{ccc} Sym\Lambda^k[n] & \longrightarrow & \widetilde{U} \\ \downarrow & & \downarrow \\ Sym\Delta[n] & \xrightarrow{\ulcorner f \urcorner} & U \end{array}$$

where the left vertical map is precisely one of the generating acyclic cofibrations, and ask whether there exists a lifting. We can factor through the pullback to get a diagram of the form:

$$\begin{array}{ccccc} Sym\Lambda^k[n] & \longrightarrow & \bullet & \longrightarrow & \widetilde{U} \\ \downarrow & & \downarrow f & & \downarrow \\ Sym\Delta[n] & = & Sym\Delta[n] & \xrightarrow{\ulcorner f \urcorner} & U \end{array},$$

where by the definition of U, f is a fibration, so we have a lifting in the diagram on the right, and hence a lifting in the outer diagram as desired. □

Unfortunately this fibration is not a universal fibration in the category $s\mathbf{SMon}$, by the following example, so we will cease to study this object much further since it does not have the desired property.

Example 9.7. Consider the diagram.

$$\begin{array}{ccc} & & \widetilde{U} \\ & & \downarrow \pi \\ \mathbb{N} & \xrightarrow[f]{} & U \end{array} \tag{9.1}$$

Here we consider $\mathbb{N}$ as a discrete simplicial symmetric monoid, so any map $\mathbb{N} \to U$ is uniquely given by specifying what the map is in the 0–simplexes, $\mathbb{N} \to U_0 = \{Y \to Sym\Delta[0]\} = \{Y \to \mathbb{N}\}$. Since f is a monoid morphism it has to send $0 \in \mathbb{N}$ to $\mathbb{N} = \mathbb{N} \in U$. Furthermore the only inverse image of the identity in $\widetilde{U}$ is the identity $(\mathbb{N} = \mathbb{N}, \mathbb{N} = \mathbb{N}) \in \widetilde{U}$. So for any fibration over $\mathbb{N}$ which has a nontrivial fiber over 0, we see that this is not a pullback in the above diagram, so specifically we see that π is not a universal fibration.

9.3 Object of weak equivalences

In this section we will study the weak equivalences between fibrations, so we fix two fibrations $E_1 \twoheadrightarrow B$, and $E_2 \twoheadrightarrow B$. We want to prove an **SMon**-enriched natural isomorphism

$$s\mathbf{SMon}/B \left(\begin{array}{cc} X & Eq_B(E_1, E_2) \\ \downarrow{\scriptstyle f}, & \downarrow \\ B & B \end{array} \right) \cong F(X, f),$$

where $Eq_B(E_1, E_2)$ is the object we wish to defined, and $F : s\mathbf{SMon}/B \longrightarrow \mathbf{Set}$ is the functor defined as:

$$F(X, f) = \left\{ \begin{array}{ccc} f^*E_1 & \xrightarrow{\sim} & f^*E_2 \\ & \searrow & \downarrow \\ & & X \end{array} \right\},$$

i.e. the set of weak equivalences between the pullbacks of $E_1 \twoheadrightarrow B$ and $E_2 \twoheadrightarrow B$ along $f : X \to B$.

Since $s\mathbf{SMon}$ has a model structure defined by lifting along a right adjoint, we have by 4.10 that $s\mathbf{SMon}$ is right proper. So pullback preserves all weak equivalences, and we get the following lemma showing that F is a meaningful functor.

Lemma 9.8. *If $f : E_1 \to E_2$ be a weak equivalence over B and $g : B' \to B$, then the induced map $g^*E_1 \to g^*E_2$ is a weak equivalence.*

Furthermore, if $s\mathbf{SMon}$ were locally cartesian closed, so that we would have an internal hom object in $s\mathbf{SMon}/B$ and we could choose the functor F to have codomain $s\mathbf{SMon}$, we get the desired representability by showing the following lemma. Unfortunately $s\mathbf{SMon}$ is not locally cartesian closed so we are missing the crucial ingredient to get an object of weak equivalences, and hence to define univalence in a way following [KLV12]. It is not even possible to define some object $Eq_B(E_1, E_2)$ in a meaningful way mimicking [KLV12] since it is well known that the set of weak equivalences very seldom can be given any symmetric monoid structure. For instance the set of invertible matrices, i.e. isomorphisms of vector spaces, is not an abelian group under sum of matrices since the sum of two invertible matrices is in general not invertible. The lemma itself still holds though, and it is a nice lemma so we finish the section with it.

Lemma 9.9. *Let $f : E_1 \to E_2$ is a morphisms in s**SMon** over B. If for one 0-simplex $Sym\Delta[0] \xrightarrow{b} B$ in each connected component, the induced map $f_b : b^*E_1 \to b^*E_2$ is a weak equivalence, then f is a weak equivalence.*

Proof. (following [KLV12]) We can assume that B is connected, otherwise we use the result in each connected component. Take some vertex $b : Sym\Delta[0] \to B$, and let $F_i \coloneqq b^*E_i$. Then we have a diagram like this, for every $e : Sym\Delta[0] \to F_1$

$$\begin{array}{ccccccccc}
\pi_{n+1}(B,b) & \longrightarrow & \pi_n(F_1,e) & \longrightarrow & \pi_n(E_1,e) & \longrightarrow & \pi_n(B,b) & \longrightarrow & \pi_{n-1}(F_1,e) \\
\downarrow{\scriptstyle id} & & \downarrow{\scriptstyle \pi_n(f_b)} & & \downarrow{\scriptstyle \pi_n(f)} & & \downarrow{\scriptstyle id} & & \downarrow{\scriptstyle \pi_{n-1}(f_b)} \\
\pi_{n+1}(B,b) & \longrightarrow & \pi_n(F_2,f(e)) & \longrightarrow & \pi_n(E_2,f(e)) & \longrightarrow & \pi_n(B,b) & \longrightarrow & \pi_{n-1}(F_2,f(e))
\end{array}.$$

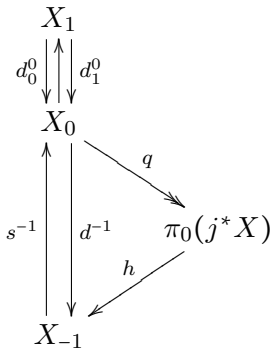
So by the five lemma $\pi_n(f)$ must be an isomorphisms since each of the $\pi_n(f_b)$ are. On the bottom of the diagram we also have an isomorphism since $\pi_0(B,b) = 0$, because B is connected. □

10 Various models

In this section we will mainly study a category which could be called many-pointed simplicial sets. It is a generalization of the category $\mathcal{S}_*$ to a presheaf category $\mathbf{A}$, since $\mathcal{S}_*$ lacks some of the nice properties required to model type theory. The idea is to give a model for type theory, preferably with univalence, in the category of Γ-spaces, which are functors $\Gamma^{op} \to \mathcal{S}_*$. Γ-spaces give rise to a way to study connective spectra, and the hope was to study the interplay between type theory and the theory of spectra. Unfortunately the theory of models for type theory turned out to be too technical and not well studied enough for the author to quite so far.

10.1 Many-pointed simplicial sets

Let $\mathcal{A}$ denote the a slight generalization of Δ which contain Δ as a full subcategory. $\mathcal{A}$ has one object for each natural number $[n] = \{0 \to 1 \to \ldots \to n\}$ and also one object $[-1] = *$, which is initial in $\mathcal{A}$, and also a retract of $[0]$. We look at the presheaf category $\mathbf{A} := [\mathcal{A}^{op}, \mathbf{Set}]$. We have the inclusion $j : \Delta \hookrightarrow \mathcal{A}$ which induces a forgetful functor $j^* : \mathbf{A} \to \mathcal{S}$, by precomposition with j. This functor has a left adjoint j_* given by the left Kan extension, and since we want to use this to define the model structure on $\mathbf{A}$ through theorem 4.10 we want to understand j_* better. By inspection we have that any two maps $X_1 \to X_{-1}$ must be equal since $[-1]$ is initial, so the map $d^{-1} : X_0 \to X_{-1}$ (which is not the inverse of a map d, but just the map for the map induced by the map $[-1] \to [0]$ in $\mathcal{A}^{op}$) factors through the surjective map $X_0 \twoheadrightarrow \pi_0(j^*X)$. So we get the following picture.



To see this, suppose that we have $y \in \pi_0(j^*X)$, it must the in the image of a $x \in X_0$ since q is surjective. We define the map $\pi_0(j^*X) \to X_{-1}$ by sending $y \mapsto d^{-1}(x)$. This is well defined since if we have any $x' \in X_0$ which is sent to y it must be in the same path component, so there must be an element $p \in X_1$ such that $x = d_0^0(p)$ and $x' = d_1^0(p)$ and hence they must be sent to the same element $d^{-1}(x)$ in X_{-1}.

This implies that if we have an isomorphism $\mathbf{A}(j_*X, Y) \cong \mathcal{S}(X, j^*Y)$, we must have that $(j_*X)_n \cong X_n \coprod \pi_0(X)$ for $X \in \mathcal{S}$, where we set $X_{-1} = \varnothing$. To see this assume we have a map

$f : X \to j^*Y$, i.e. a diagram as follows in $\mathcal{S}$:

$$\begin{array}{ccc} X_2 & \xrightarrow{f_2} & Y_2 \\ \downarrow\uparrow & & \downarrow\uparrow \\ X_1 & \xrightarrow{f_1} & Y_1 \\ {\scriptstyle d_0^0}\downarrow\uparrow{\scriptstyle d_1^0} & & {\scriptstyle d_0^0}\downarrow\uparrow{\scriptstyle d_1^0} \\ X_0 & \xrightarrow{f_0} & Y_0 \end{array}$$

then we get a diagram as follows in $\mathbf{A}$:

$$\begin{array}{ccc} \pi_0(X) \coprod X_2 & \xrightarrow{s_0^1 s^0 s^{-1} \circ h \circ \pi_0(f) \coprod f_2} & Y_2 \\ \downarrow\uparrow & & \downarrow\uparrow \\ \pi_0(X) \coprod X_1 & \xrightarrow{s^0 s^{-1} \circ h \circ \pi_0(f) \coprod f_1} & Y_1 \\ {\scriptstyle d_0^0}\downarrow\uparrow{\scriptstyle d_1^0} & & {\scriptstyle d_0^0}\downarrow\uparrow{\scriptstyle d_1^0} \\ \pi_0(X) \coprod X_0 & \xrightarrow{s^{-1} \circ h \circ \pi_0(f) \coprod f_0} & Y_0 \\ & & \searrow^{q} \\ {\scriptstyle s^{-1}}\uparrow\downarrow{\scriptstyle d^{-1}} & \pi_0(j^*Y) & {\scriptstyle s^{-1}}\uparrow\downarrow{\scriptstyle d^{-1}} \\ & \nearrow_{\pi_0(f)} \quad \searrow_{h} & \\ \pi_0(X) & \xrightarrow{h \circ \pi_0(f)} & Y_{-1}. \end{array}$$

To see that the above diagram commutes, and is the lower part of a diagram in $\mathbf{A}$, is a technical exercise which can be checked. I have given full name to the first few maps to outline how the proof starts. One could note that there is a choice of map $\pi_0(X) \to Y_2$, since there are two degenerate maps $s_0^1, s_1^1 : Y_1 \to Y_2$ but thanks to the simplicial identities these are in fact equal as maps $Y_0 \to Y_2$ so the map is well defined.

If on the other hand we have a map $g : j_*X \to Y \in \mathbf{A}$, we have in each degree $n \geq 0$ a map $g_n : \pi_0(X) \coprod X_n \to Y_n$, so we have a map $X_n \to Y_n$, and we can forget all information about $\pi_0(X)$ and get a map $X \to j^*Y$. It is now clear that the set of maps $f : X \to j^*Y \in \mathcal{S}$ are isomorphic to the set of maps $j_*X \to Y \in \mathbf{A}$ by the above discussion.

Since the functor j_* simply adds the set of path-components in each degree (even the degree -1 where there was nothing before) it is easy to see that it sends small objects in $\mathcal{S}$ to small objects in $\mathbf{A}$. Further is we take pushouts diagrams of the form

$$\begin{array}{ccc} \Lambda^k[n] \coprod * & \longrightarrow & X_a \\ \downarrow & & \downarrow \\ \Delta[n] \coprod * & \longrightarrow & X_{a+1}, \end{array}$$

where the left side is the generating cofibrations, it follows by the same arguments as in $\mathcal{S}$ that the map on the right side is a weak equivalence. Also by the same arguments as in simplicial sets sequential colimits of weak equivalences is again a weak equivalence. And hence by the transfer theorem 4.10 the category $\mathbf{A}$ is a cofibrantly generated model structure.

We now have a cofibrantly generated model structure on $\mathbf{A}$ given by lifting the model structure on $\mathcal{S}$ through the adjunction $j^* : \mathbf{A} \rightleftarrows \mathcal{S} : j_*$, and just as for $s\mathbf{SMon}$ lemma 4.10 gives that $\mathbf{A}$ is a right proper model structure. Since $\mathbf{A}$ is a presheaf category it is a topos, and hence locally cartesian closed.

The advantage of working in cofibrantly generated model categories is that is gives us tools for describing the cofibrations as something other than just maps with lifting properties. The cofibrations in a cofibrantly generated model category, with generating cofibrations and generating acyclic cofibrations I and J can be described in the following two equivalent way. Cofibrations are given as $I - cof$, i.e. the maps with left lifting property with respect to the maps with the right lifting property with respect to the maps in I. Equivalently they can be given as retracts of maps in $I - cell$, i.e. closure under pushouts and sequential colimits of maps in I. The set I is the set of injections $\{\partial\Delta[n] \coprod * \hookrightarrow \Delta[n] \coprod *|0 \leq n\}$ (since $\pi_0(\partial\Delta[n]) = \pi_0(\Delta[n]) = *$), and pushouts of injections in any topos are injections. Further the sequential colimit of injections is again an injection. Lastly any retract of an injection is an injection. So the set of cofibrations is precisely the set of injections.

Further we know that limits always preserve injections, so it preserves the cofibrations in $\mathbf{A}$. This gives that $\mathbf{A}$ is a type-theoretic model category (Def 6.9) so its subcategory of fibrant objects $\mathbf{A_f}$ is a type-theoretic fibration category.

10.2 Possible future work

If we could construct a univalent universe in $\mathbf{A}$ we could use methods similar to the methods in [Shu12] and [Shu13] to get a univalent universe in $(\mathbf{A}^{\Gamma^{op}})_\mathbf{f}$ the category of fibrant objects in a category which could be called many pointed Gamma-spaces. The category Γ^{op} is a skeleton of the category of finite pointed sets.

Then it would be a problem of looking at the full subcategory of ordinary Gamma-spaces which is the category $\mathcal{S}_*^{\Gamma^{op}}$, formed from the full subcategory of $\mathbf{A}$ of all many-pointed spaces having just the one point set $*$ as its -1 simplex, i.e. normal pointed simplicial sets $*/\mathcal{S} \cong [\Delta^{op}, Ens_*]$.

There is another category which could be of interest to examine further, namely the category $\mathbf{Cat}$ of small categories. It is a cartesian closed category, but not locally cartesian closed since pullback does not preserve coequalizers, see remarks following Corollary 1.5.3 in [Joh02a]. We have however that for any fibration $f : A \to B$ the pullback $f^* : \mathbf{Cat}/B \to \mathbf{Cat}/A$ has a right adjoint $\prod_f$. The canonical model structure on $\mathbf{Cat}$, found in [JT91], is given by a functor is

- a weak equivalence if it is an equivalence of categories.
- a cofibration if it is an isocofibration, i.e. if it is injective on objects.
- a fibration if it is an isofibration.

Definition 10.1. *An isofibration is a functor $F : E \to B$ such that for any object $e \in E$ and any isomorphism $\phi : F(e) \cong b$, there exists an isomorphism $\psi : e \cong e'$ such that $F(\psi) = \phi$.*

Now we could try to continue as for $\mathbf{A}$ and show that this model structure give $\mathbf{Cat}$ the structure of a model category with limits that preserve cofibrations, and which is right proper. This would mean that $\mathbf{Cat}$ is a type-theoretic model category, and we could study $\mathbf{Cat_f}$ the subcategory of fibrant objects which would be a type-theoretic fibration category.

References

[BM03] Clemens Berger and Ieke Moerdijk. Axiomatic homotopy theory for operads. *Comment. Math. Helv.*, 78(4):805–831, 2003.

[Bor94] Francis Borceux. *Handbook of categorical algebra. 2*, volume 51 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994. Categories and structures.

[Bro73] Kenneth S. Brown. Abstract homotopy theory and generalized sheaf cohomology. *Trans. Amer. Math. Soc.*, 186:419–458, 1973.

[CD11] Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and Martin-Löf type theories. In *Typed Lambda Calculi and Applications*, pages 91–106. Springer, 2011.

[DGM13] Bjørn Ian Dundas, Thomas G. Goodwillie, and Randy McCarthy. *The local structure of algebraic K-theory*, volume 18 of *Algebra and Applications*. Springer-Verlag London Ltd., London, 2013.

[dt04] The Coq development team. The Coq proof assistant reference manual, 2004. Version 8.0.

[GJ99] Paul G. Goerss and John F. Jardine. *Simplicial homotopy theory*, volume 174 of *Progress in Mathematics*. Birkhäuser Verlag, Basel, 1999.

[Hov99] Mark Hovey. *Model categories*, volume 63 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 1999.

[Isb69] John R. Isbell. On coherent algebras and strict algebras. *J. Algebra*, 13:299–307, 1969.

[Jac99] Bart Jacobs. *Categorical logic and type theory*, volume 141 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1999.

[Joh02a] Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium. Vol. 1*, volume 43 of *Oxford Logic Guides*. The Clarendon Press, Oxford University Press, New York, 2002.

[Joh02b] Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium. Vol. 2*, volume 44 of *Oxford Logic Guides*. The Clarendon Press, Oxford University Press, Oxford, 2002.

[JT91] André Joyal and Myles Tierney. Strong stacks and classifying spaces. In *Category theory (Como, 1990)*, volume 1488 of *Lecture Notes in Math.*, pages 213–236. Springer, Berlin, 1991.

[Kel82] Gregory M Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

[KLV12] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. *arXiv preprint arXiv:1211.2851*, 2012.

[Lam68] Joachim Lambek. Deductive systems and categories. I. Syntactic calculus and residuated categories. *Math. Systems Theory*, 2:287–318, 1968.

[Lam69] Joachim Lambek. Deductive systems and categories. II. Standard constructions and closed categories. In *Category Theory, Homology Theory and their Applications, I (Battelle Institute Conference, Seattle, Wash., 1968, Vol. One)*, pages 76–122. Springer, Berlin, 1969.

[ML75] Per Martin-Löf. About models for intuitionistic type theories and the notion of definitional equality. In *Proceedings of the Third Scandinavian Logic Symposium (Univ. Uppsala, Uppsala, 1973)*, pages 81–109. Stud. Logic Found. Math., Vol. 82. North-Holland, Amsterdam, 1975.

[ML84] Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory. Lecture Notes*. Bibliopolis, Naples, 1984. Notes by Giovanni Sambin.

[ML98] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer verlag, 1998.

[Por08] Hans-E. Porst. On categories of monoids, comonoids, and bimonoids. *Quaest. Math.*, 31(2):127–139, 2008.

[See84] R. A. G. Seely. Locally Cartesian closed categories and type theory. *Math. Proc. Cambridge Philos. Soc.*, 95(1):33–48, 1984.

[Shu12] Michael Shulman. Univalence for inverse diagrams, oplax limits, and gluing, and homotopy canonicity. *arXiv preprint arXiv:1203.3253*, 2012.

[Shu13] M. Shulman. The univalence axiom for elegant Reedy presheaves. *ArXiv e-prints*, July 2013.

[Sim] Harold Simmons. The tensor product of commutative monoids. `http://www.cs.man.ac.uk/~hsimmons/DOCUMENTS/PAPERSandNOTES/TPoCM.pdf`.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.