Utrecht University

Master's Thesis
Mathematical Sciences

A Formalization of the Algebraic Small Object Argument in `UniMath`

Author:
Dennis Hilhorst

Supervisor:
Dr. Paige North

Second Reader:
Dr. Jaap van Oosten

November 30, 2023

Graduate School of Natural Sciences

Utrecht University

Abstract

Model categories, introduced by Quillen in 1967, form the cornerstone of modern homotopy theory, providing a language and tools for this branch of mathematics. They consist of two interacting weak factorization systems. Quillen defined a transfinite construction to generate weak factorization systems and thereby model structures on a category, given sufficiently well-behaved classes of maps: the *small object argument*. Weak factorization systems, lacking algebraic structure, suffer some defects from a categorical point of view. Grandis and Tholen introduced the notion of *natural weak factorization system* to rectify these issues. Garner pointed out some problematic aspects of the small object argument: that it is not convergent, that it is not related to other known transfinite constructions and that it satisfies no universal property. He refined the small object argument to generate natural weak factorization systems in a more algebraically coherent way.

In this thesis, we elaborate, rephrase and formalize Garner's 'algebraic' small object argument. The formalization is written using the Coq proof checker, using the `UniMath` library. This is a formalization framework based on Homotopy Type Theory (HoTT). The formalization provides an air-tight confirmation of the theory through computer verified proofs.

We fill in the details in Garner's construction, add much needed intuition and redefine parts of the construction to be more direct and accessible. We rephrase the theory in more modern language, using constructions like *displayed categories* and a modern, less strict notion of *monoidal categories*, so that it is fit for formalization. We point out the interaction between the theory and the HoTT foundations, and describe some of the constructive issues we come across.

The formalization can be found at
`https://github.com/DenSinH/unimath-small-obj-arg`.

Acknowledgements

First and foremost, I would like to thank my supervisor Paige North for the many helpful meetings and her valuable feedback on my work. I would also like to thank Sina Hazratpour for his time and insight during some of these meetings. Additionally, I would like to thank the second reader Jaap van Oosten for taking the time to evaluate this thesis.

I would like to thank Merel for her support and without whom I would probably be starving or dehydrated. I would also like to thank Marco for his support, as well as my parents and brothers for always being by my side.

Contents

Chapter 1

Introduction

Model categories, first introduced by Quillen [1], form the foundation of modern homotopy theory [2]. They provide a language and tools for this branch of mathematics, placing results from the homotopy theory of topological spaces in a more general context. This has expanded the concept of homotopy theory to one that describes a more general methodology, applicable in a much broader range of subjects. In this way, model category theory plays a role in homotopy theory analogous to that of category theory in mathematics. It facilitates comparisons and allows for common proofs of seemingly unrelated results [3].

Let us sketch the problem that model categories solve. In traditional homotopy theory, meaning that of topological spaces, one commonly studies properties that are invariant under homotopy. There is a class of morphisms in the category **TOP** of topological spaces, called *weak equivalences*, which preserve these invariants. One might like to consider these weak equivalences to be isomorphisms, even though they are not. This is but one example of a more general problem in category theory. Model category theory provides a way to construct a well-behaved *homotopy category*, solving this more general problem. The homotopy category describes the category 'up to homotopy', in which case the weak equivalences are in fact isomorphisms. The construction provides results analogous to those in traditional homotopy theory, placed in a more generic context. It also allows for a global study of homotopy theoretical concepts, contrasting the more local nature of traditional homotopy theory.

The theory has been applied in many other areas of mathematics, such as derived algebraic geometry [4], condensed mathematics [5] and of course in homological algebra and K-theory as subfields of algebraic topology [2]. It even has applications in computer science [6]. Furthermore, model categories are also used to present $(\infty, 1)$-*categories*, which are categories admitting higher structure, in the form of morphisms between morphisms (2-morphisms), morphisms between those (3-morphisms) and so on [7]. This is a relatively young field of mathematics, and is still very actively studied.

Model categories describe the behavior of two interacting *weak factorization systems*, related through the weak equivalences. Weak factorization systems in turn consist of two classes of maps (the left and right class) which satisfy a dual lifting property, such that any map in the category can be factored as a left and right map. The notion of weak factorization system may be a familiar one to topologists, though perhaps not in name. In topology, the Homotopy Extension Property and the Homotopy Lifting Property define two weak factorization systems, forming a well-known model structure on **TOP** known as the Hurewicz model structure [3].

In a weak factorization system (WFS), neither the lifting property nor the factorization is defined to satisfy any additional properties, and the factorization need not be unique. Hence the adjective 'weak'. Though they have been studied extensively by category theorists, these points make it so WFSs suffer from some defects from a categorical point of view.

Grandis and Tholen aimed to rectify these defects, introducing the notion of *natural weak*

factorization system (NWFS) [8]. An NWFS is based on a *functorial factorization* whose functors underlie a comonad and a monad. This allows one to construct a canonical solution to any lifting problem between (the corresponding notion of) a left and right map, in a natural way. The added algebraic structure resolves the defects Grandis and Tholen saw in 'plain' WFSs.

In his formulation of model category theory, Quillen also described a way of *cofibrantly generating* a WFS, starting from a sufficiently well-behaved class of maps: the *small object argument* [1]. The construction may even be used to generate model structures. Useful as it is, this construction has some problematic aspects, like the notion of WFS itself.

Richard Garner remarks that the transfinite construction Quillen describes does not converge, has no universal property and is not related to any other known transfinite constructions [9]. He refined Quillen's small object argument so that it produces an NWFS in an algebraically coherent way, resolving these problematics aspects. Further extending this theory, Emily Riehl introduced the notion of *algebraic model structures* [10], which form an equivalent to model structures using NWFSs instead of WFSs.

In this thesis, we focus mostly on Garner's 'algebraic' small object argument. The goal is to elaborate and rephrase the construction in a more modern and accessible way, as well as to formalize it using `UniMath`, based on the Coq proof checker. We will also point out some more constructive issues in the theory of WFSs and in Quillen's 'classical' small object argument, which were not mentioned by Garner.

We aim to elaborate his argument, filling in the gaps and providing much needed intuition using important examples such as the standard model structure on the category **TOP** of topological spaces or **SSET** of simplicial sets, as well as a much simpler example on the category **SET** of sets. Though very extensive, the `UniMath` library still limits us in the use of high level arguments, forcing us to use more direct arguments, indirectly making for a more accessible and detailed description of the construction. Limitations in formalization also force us to use more modern constructions, such as the notion of *displayed categories* [11] and an adapted, slightly weaker notion of *monoidal categories* [12] [13].

The Coq proof checker in conjunction with the `UniMath` library is a formalization framework that is built on *Homotopy Type Theory* (HoTT), or *univalent foundations*. HoTT is a young field of mathematics, and provides a new foundation of mathematics that is fundamentally different from set theory, upon which the mathematical education of most readers is likely based. The theory combines concepts from type theory and homotopy theory, and is a foundation that is very fit to underlie a computer proof assistant [14].

Since the constructions by Quillen and Garner are built on set-theory based mathematics, reformulating them in HoTT provides new insights and perspectives on the theory. It will point out where we are limited, or where we in fact require less strict assumptions than in the set-theory based counterpart. The formalization removes any ambiguities and provides an air-tight confirmation of the theory through a computer verified proof.

Describing the theory of model categories in HoTT is in fact a very 'meta' thing to do. In HoTT, types can be described as ∞-groupoids, which may be presented as topological spaces or simplicial sets [14]. A well-known model structure on **SSET**, one that can be generated with the (algebraic) small object argument, can be used to present type theory itself [15].

In this thesis, we first introduce the reader to HoTT and proof checking in `UniMath` in Chapter 2. This should provide a sufficient understanding of HoTT to read the type theoretic definitions that are sprinkled throughout the thesis. We highlight some of the main differences between HoTT and set theory as a foundation. The chapter also aims to give the reader an idea of what formalization entails.

We then introduce *displayed categories* in Chapter 3. These provide a very natural framework for defining functorial factorizations and NWFSs in a way that interacts well with HoTT.

In Chapter 4, we introduce the reader to weak factorization systems and model categories. We introduce some examples and explain why model categories are useful following existing literature, but we also give new insights into some of the limitations of this theory. In Chapter 5 we describe how these limitations can be fixed by using *natural* weak factorization systems instead, after introducing their basic algebraic properties, as well as some examples.

We then get to the small object argument. We first introduce Quillen's 'classical' small object argument in Chapter 6, following a detailed description by Hovey [2]. As we go along, we follow two important examples in **TOP** and **SSET**. Though again mostly expository in nature, we also point out a constructive issue in Quillen's classical small object argument.

Finally, we get to the main part of the thesis: the 'algebraic' small object argument by Garner. In Chapter 7, we follow Garner's construction, fill in the details that he leaves out and point out the problems we run into when formalizing the theory. We add intuition in the form of examples in **TOP** and **SSET**, as well as a more trivial, but computable example in **SET**. We also redefine part of the construction to be more direct and intuitive, further making the argument accessible to those unfamiliar with the construction.

The formalization that accompanies this thesis can be found at `https://github.com/DenSinH/unimath-small-obj-arg`.

Preliminaries

In this thesis, we write $X : \mathcal{C}$ to denote that X is an object of a category $\mathcal{C}$, following HoTT conventions [14]. We write $f : X \to Y$ (following `UniMath` notation) or $f : \mathrm{Hom}(X, Y)$ to denote that f is a morphism from X to Y in $\mathcal{C}$. Following the choice adopted for the `UniMath` library, we write compositions in 'diagrammatic order', meaning that the composite of $f : X \to Y$ and $g : Y \to Z$ is written as $f \cdot g : X \to Z$.

Definitions and lemmas often correspond to formalized definitions and lemmas. Whenever this is the case, the corresponding definition or lemma is linked, and in the case of definitions, the formalized definition has often been included in the thesis. To distinguish definitions and lemmas that have been formalized in the existing `UniMath` library from those that are new in the formalization accompanying this thesis, we underline links to those from the `UniMath` library. For example, `category` leads to the definition of a category in the `UniMath` library, whereas `LNWFS_tot_monoidal` leads to a lemma in the new formalization. Code samples of definitions that are included in the thesis are always original definitions, unless explicitly stated.

As a last preliminary remark, since the thesis has been accompanied by a formalization, many proofs have either been stripped down or omitted entirely. Sometimes, a sentence is written to describe the idea of a proof, and sometimes a (usually incomplete) proof is given. These proofs serve mostly as an aid in understanding the ideas behind the formalization or are meant to clarify where proofs might differ from existing literature. They should *not* be taken as complete or formal proofs.

Chapter 2

Homotopy Type Theory and Formalization

One of the main components of this thesis is the formalization that comes along with it. This formalization is to be part of the `UniMath` library, based on the Coq proof checker. The `UniMath` library is based on *univalent foundations*. In other words, the mathematics it formalizes is based on Homotopy Type Theory (HoTT) as opposed to set theory, upon which the mathematical education the reader might have followed is likely based. More formally, the Coq proof checker is based on the *Calculus of Inductive Constructions*, which is a closely related dependent type theory [15]. For all intents and purposes though, we may just consider it to be based on HoTT, or *univalent foundations*.

The idea of HoTT is that mathematical objects are not just 'things that might be in one or more sets', but rather 'things of a certain type'. A major difference being that an object can only be of one type. Even lemmas and theorems are types, and a term of such a type corresponds to a proof of the statement. The beauty in this is that the theory becomes purely constructive: if one proves a theorem, one immediately gets an object that satisfies the statement. This idea will be made more clear throughout this chapter.

The reason that the theory is called 'Homotopy' Type Theory, is because we can also view types as spaces, and terms of a certain type as points of a certain space. Equalities become paths, equalities of equalities (indeed, not all equalities are the same) are homotopies, etc. When reading this chapter, it may be helpful, at least at the beginning, to keep this view in mind when learning about types for the first time.

This chapter is fully expository, aiming to give an introduction to HoTT, alongside an introduction to formalization in Coq / `UniMath`, starting with the most basic constructions. Later we review some theory about category theory in univalent foundations, which we will need later on in this thesis. We introduce this theory following the HoTT book [14].

2.1 Types

First of all, it is good to clarify what distinguishes (homotopy) type theory from set theory. The HoTT book makes this very clear by noting that set theory has two 'layers'. There is a deductive system of logic, and within this system, the axioms of a theory like set theory. "Set theory is not only about sets, but rather about the interplay between sets (the objects of the second layer) and propositions (the objects of the first layer)" [14]. Type theory, on the other hand, is *its own deductive system*. Instead of the two separate notions of logic and sets, there is only one basic notion: types. Everything can be formulated as a type. Even propositions (things we can prove, assume, negate etc.) are just specific types. In this way, proving a theorem becomes the same thing as constructing an element of a specific type.

This leads to another major difference between the logic in the two theories. It concerns so-called 'judgements', which are the conclusions we draw using the rules defined in a deductive system. In set theory, there is only really one type of judgement, namely that 'a proposition P has a proof'. We use rules like 'P and Q implies $P \wedge Q$' to deduce other judgements, which are of this same form. In the example, that would be the deduction that '$P \wedge Q$ has a proof'.

In type theory, this basic judgement is written as $p : P$, and is pronounced 'the term p has type P', or 'p is an element of P'. If P is a proposition, we sometimes call p a 'witness' of P. However, types can also behave more like sets in set theory. Suppose A is a type, then one might think of the type theoretical statement $a : A$ to mean something like $a \in A$ in set theory. These things are fundamentally different though, since $a : A$ is a judgement, it is not something we can prove or disprove, it just is. In contrast, $a \in A$ is a proposition about two pre-existing objects a and A, which we can prove to be true or false. It is good to think about it in precisely that way: in type theory the statement $a : A$ is no relation between pre-existing objects. We *cannot* consider a isolated from its type A. I personally like to compare this to strongly typed programming languages, where one might declare a variable `int x;` declaring that `x` is a term of type `int`. Here too, there is no way to consider `x` in isolation: it *is* an `int`, and it is not something else. It may be useful to keep this idea in mind when reading about formalization. We will introduce other useful interpretations of HoTT in this chapter as well.

Then we get to the last major difference between set theory and type theory, which is the treatment of equality. Effectively, there are two notions of equality: *propositional* and *definitional*. The first one is something that is in line with what we commonly do in (set theory based) mathematics: these types of equalities are something that we can prove. For example, in set theory, one might have two objects a, b which are in a set A. One can, possibly using some hypotheses, prove that $a = b$. Similarly, in type theory, suppose $a, b : A$. Then one might show that the *identity type* $a =_A b$ is inhabited. We say that a and b are *propositionally equal*.

The second form of equality is much stronger. The notion of *definitional equality* is an equality *judgement*, at the same level as $a : A$. Meaning that this is not something that we can show to be true or not, it just *is*. One simple example is the following: suppose $f : \mathbb{N} \to \mathbb{N}$ is a function that is defined as $f(x) := x^2$, then just by evaluating the expression, we know that $f(3)$ is *definitionally* equal to 3^2. After all, if we define x to be 3, then the expression $f(x)$ is just a 'relabelling' of the expression x^2, which in turn is just a relabelling of the expression 3^2. Notice how we do not say that $f(3)$ is definitionally equal to 9? That is because it is in fact not! From the way one defines the natural numbers $\mathbb{N}$, and the way one defines multiplication, one can deduce that, indeed, $3^2 =_{\mathbb{N}} 9$, but this is *not a definitional equality*.

We can also relate this back to programming for those familiar, specifically object-oriented programming. One might consider a comparison which evaluates to a boolean as a propositional equality, whereas a definitional equality corresponds with two variables pointing to the same object. In the second case, the variables are fundamentally the same, though perhaps labelled differently, whereas in the first, the compared variables may have been constructed entirely differently.

2.1.1 Formalization

By *formalization*, we mean the process of proving theory using proof assistant software on a computer. A nice quote on the nLab explains what this entails precisely: "Generally, people tend to speak of formal arguments when referring to arguments by formalism, hence by purely symbolic manipulations following syntactic rewrite rules [16, `proof`]." Indeed, when writing proofs using the Coq proof assistant, we use *tactics* to rewrite our current *goal* step by step, until we finish the proof we are working on.

The formalization that accompanies this thesis is written with the Coq proof assistant, using the `UniMath` library. The Coq proof assistant is based on the *Calculus of Inductive Constructions*, which is closely related to HoTT [15]. Many important theorems have been proven using the Coq proof assistant already [17]. The `UniMath` library is an actively developed library that already formalizes lots of theory from many branches of mathematics [18]. From this point onwards, we will refer to the formalization framework simply as `UniMath`.

Similar to the type theoretic notation, in `UniMath`, one denotes type membership as

```
x : nat   or   x y : nat
```

where `nat` is the type of natural numbers, and one denotes *definitional* equalities as

```
x := 3   or   x := 3 : nat.
```

Propositional equalities between terms of the same type are denoted simply as `x = y`. As mentioned before, propositions are simply just specific types. When formalizing a theory, one denotes them as

```
Lemma add_comm (x y : nat) : x + y = y + x.
Proof.
    ...
Qed.
```

where `Lemma` could be any of `Lemma`, `Definition`, `Proposition`, `Theorem`... This defines a term `add_comm`, of type

$$\prod_{x,y:\mathbb{N}} x + y =_{\mathbb{N}} y + x.$$

The precise meaning of this will become clear in this chapter, but it is important to remember that lemmas denoted like this in `UniMath` are effectively just definitions of terms in large $\prod$-types. These types allow us to construct terms of a certain type (such as `x + y = y + x`) given terms of another type (such as `x y : nat`).

When formalizing a proposition or lemma like this, we try to prove a certain *goal*. In the example this is `x + y = y + x`. We prove a goal using *tactics*. These are very small steps in a proof, or a very simple construction one could do to obtain a term of a type. Some tactics use *hypotheses*, which are just terms that we may have been given in the statement of the lemma, or which we may obtain from previously shown lemmas.

We give another example to further clarify what propositional and definitional equalities are, as well as to introduce the very first, basic tactic. Suppose we have a hypothesis `x : nat`. Then the statement `x = x` is a propositional equality. It is something we may want to prove. However, it is trivial to prove this equality, since `x` is equal to `x` in the *definitional* sense as well. A proof for this in a formalization would look like

```
Lemma eq_refl (x : nat) : x = x.
Proof.
    reflexivity.
Qed.
```

where the tactic `reflexivity` is the most basic tactic available. The `reflexivity` tactic does exactly what we want in the lemma, as it gives a term of type `a = a` for any term `a : A` for any type `A`. Effectively, it turns a definitional equality into a propositional one. One more important tactic we introduce right away is the `rewrite` tactic. This tactic takes a (propositional) equality, and rewrites the equality in the goal we are trying to prove. For example, suppose we were trying to prove a goal

```
x + y = x + x
```

for `x y : nat`, and we have a *hypothesis* `H : y = x`. The tactic `rewrite H` turns the goal from `x + y = x + x` into `x + x = x + x`, which we can prove simply with `reflexivity`. As we introduce more type theoretical concepts, we will also introduce some of the tactics used in formalization.

As one might expect, it is much easier to work with definitional equalities than with propositional equalities in formalization. For example, consider the example goal `x + y = x + x` from before. Had we known that `y := x`, meaning that `y` is *definitionally* equal to `x`, we would be able to finish the proof with a single `reflexivity` statement. After all, `y` is just a different label for `x`.

The proof assistant will be able to reduce expressions on a judgemental level. It may even compute that two expressions are equal on a judgemental level. In our example, the definitional equality `y := x` allows us to finish the proof of `x + y = x + x` with a single `reflexivity` tactic, rather than having to deal with `rewrite`. This explains why we would much rather define types in a way that any equalities we might need are *definitional* rather than *propositional*, especially in formalization.

In set-based mathematics, one often defines sets using predicates, so for example in geometry, one might define the set of sections of a trivial bundle $A \times B \to B$ of sets as

$$\{\, \sigma : B \to A \times B \mid \sigma \cdot \mathrm{pr}_2 = \mathrm{id}_B \,\}.$$

It corresponds with the idea of pre-existing objects for which some proposition holds: I have a σ, and it has this property. One could do something similar in HoTT and `UniMath`, but proving statements involving such sections would involve a lot of rewriting. We prefer to define these sections in a way that the predicate becomes something 'baked into the definition'. For example, we might define some sort of helper type

```
Definition triv_bundle_section_data (A B : UU) := B -> A.
```

This allows us to define a section as

```
Definition section {A B : UU} (s : triv_bundle_section_data A B)
    : B -> A × B := λ (b : B), (s b,, b).
```

where composition with the projection yields identity in a definitional way instead. Using definitional data in this way allows the proof checker to simplify terms on a judgemental level, saving us a lot of work. This idea is more in line with the HoTT approach: I have a σ of a certain type, and the *type* has this property.

2.1.2 Homotopy Type Theory

So far, what we have discussed mostly boils down to type theory, not specifically *homotopy* type theory. The central idea in homotopy type theory is that we can view types as spaces, and terms of types as points in the corresponding space. With this interpretation in mind, (propositional) equalities between terms of a certain type correspond with paths in the corresponding space. See for example Figure 2.1 where an equality $p : a =_A b$ is depicted for terms $a, b : A$. These types are commonly referred to as *identity types*. With this interpretation in mind, it may be easier to see why equalities form their own type, corresponding with the path space of a certain type. In this type of equalities, we can again consider the type of equalities, i.e. the type $p =_{a=_A b} q$ for terms $p, q : a =_A b$. Terms of this type correspond with homotopies, see Figure 2.2. We can actually repeat this process with higher homotopies, and keep iteratively constructing higher *identity types*.

Using this interpretation, the `rewrite` tactic introduced above effectively allows us to procedurally concatenate paths to obtain the desired equality. For example, if we have a goal `x = z`, with hypotheses `H0 : x = y` and `H1 : y = z`, a proof might look like

```
rewrite H0. rewrite H1. reflexivity.
```

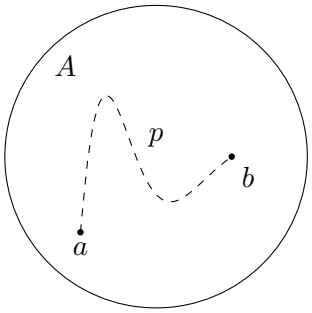


Figure 2.1: Propositional equality p of two terms $a, b : A$.

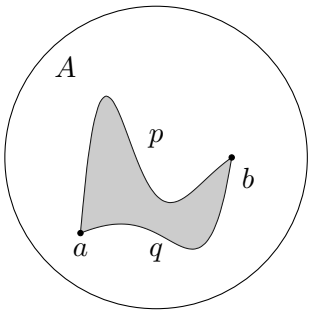


Figure 2.2: Example of a term of type $p =_{a=_A b} q$ for $a, b : A$ and $p, q : a =_A b$.

This concatenates the paths `H0` and `H1` to form a path of type `x = z`.

The more basic `reflexivity` tactic gives us a constant path refl_a for any term $a : A$ for any type A. Another operation one might want to do is inverting paths. This is done with the `symmetry` tactic, which turns a goal of the form `x = y` into `y = x`.

Example 2.1. *One of the most basic examples of a type is the* unit *type, defined as* `unit` *in* `UniMath`*. It is a type that has only one term,* `tt` *in* `UniMath`*. It may be interpreted as the one-point space* $*$*.*

Example 2.2. *Another basic example is the* empty type*. It is a type that contains* no *elements, denoted* `empty` *in* `UniMath`*. It corresponds with the empty space* $\emptyset$*.*

2.2 Constructions on Types

We now have a (very) basic understanding of what (homotopy) type theory entails, the differences with set theory, some basic introduction to formalization in `UniMath` and the homotopic interpretation of types and (propositional) equalities. We go on to discuss some constructions on types, as well as their corresponding concepts in `UniMath` and their interpretation in HoTT. Firstly though, in order to be able to properly define these constructions, we first introduce simple *function types*, as well as the idea of *type universes*, allowing us to construct *dependent types*.

2.2.1 Function Types

We have already seen an example of this before, when we introduced the function $f : \mathbb{N} \to \mathbb{N}$ mapping x to x^2. A function type $f : A \to B$ is exactly what one might expect it to be: it is a map from A to B. The interpretation in HoTT is also what one expects it to be, f is a continuous map from A to B. When formalizing, we can use the `apply` tactic in combination with a term `f : A -> B` to turn a goal of the form `B` (prompting us to construct a term of type `B`) into the goal `A` (prompting us to construct a term of type `A`). We simply tell the proof

assistant `apply f`. After all, if we can construct a term of type `A`, we can use `f` to obtain a term of type `B`. This allows us to kind of 'work backwards' in a proof. There is also a tactic that we can use to take a more 'forward' route: `exact`. If we have a hypothesis `a : A`, the tactic `exact (f a)` then tells to proof assistant that we have found a term of type `B` (which was our goal) and finishes the proof.

It is important to note that function types act functorially on paths (or equalities). In other words, the image of a path $p : a =_A b$ under a function $f : A \to B$ is again a path. This allows for even more useful constructions, such as the `maponpaths` lemma in `UniMath`, which says the following.

Lemma 2.3 (`maponpaths`)**.** *Given a function* `f : A -> B` *between types* `A` *and* `B`*, we have*

```
a = a' -> f a = f a'.
```

Note that in this lemma, the conclusion was a function type, giving a term of type `f a = f a'`, when passed a term of type `a = a'`. This leads to the third interpretation of type theoretical constructions: the logical interpretation. In this interpretation, types can be seen as propositions, terms of types as proofs, and functions as implications. In this interpretation, the empty type may be seen as the proposition that is always false, and the unit type as that which is always true.

2.2.2 Type Universes

So far, we have just been saying 'A is a type' to introduce types. Formally though, one should say that $A : \mathcal{U}$, or in words: A is in the universe $\mathcal{U}$. Simply put, a universe is a type whose elements are types. Similar to set theory, we might have a desire for a universe of types $\mathcal{U}_\infty$ that contains all types including $\mathcal{U}_\infty$ itself. This would give rise to all sorts of problems, like being able to encode Russell's paradox [14].

Instead, we introduce a hierarchy of universes

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \dots$$

where every universe $\mathcal{U}_i$ is an element of the next universe $\mathcal{U}_{i+1}$. One has to be careful with this definition, but for the purposes of this thesis, it is enough to just think of a single universe $\mathcal{U}$. Formally, one could take this to mean that the judgement $A : \mathcal{U}$ means that $A : \mathcal{U}_i$ for some i, or one could just think of it as if we have a 'large enough universe $\mathcal{U}$', or a 'universe of small types' that is sufficient for our purposes in this thesis. In `UniMath`, this judgement is denoted as `A : UU`.

Universes are an important tool to construct *type families*, or *dependent types*. They are function types $A \to \mathcal{U}$ for some $A : \mathcal{U}$. They correspond with families of sets in set theory. Homotopy theoretically, we may think of a type family $B : A \to \mathcal{U}$ as a fibration with base space A with fiber $B(a)$ over any $a : A$, see Figure 2.3. Logically, we could view it as a predicate B on a type A.

2.2.3 $\prod$-types

Much more interesting than plain function types are $\prod$*-types*, or *dependent function types*. Simply put, these types are functions from a domain type to a variable codomain type, dependent on the term in the domain on which the function is applied. Set theoretically, one may interpret this as a product of sets, hence the name $\prod$-type.

Given a type $A : \mathcal{U}$ and a type family $B : A \to \mathcal{U}$, we construct the type of *dependent functions*

$$\prod_{a:A} B(a).$$

As mentioned, these types are like functions with variable codomains. It says that given a term $a : A$, we can use a term of a $\prod$-type like this to obtain a term $b : B(a)$. This may be clarified with the homotopy theoretical interpretation of $\prod$-types: the *space of sections*, as drawn in Figure 2.3. Types like this are extremely useful, as most lemmas in formalization will actually just be large $\prod$-types. Recalling the example `add_comm` from before, of type

$$\prod_{xy:\mathbb{N}} x + y =_{\mathbb{N}} y + x,$$

which in fact is a 'nested' $\prod$-type

$$\prod_{x:\mathbb{N}} \prod_{y:\mathbb{N}} x + y =_{\mathbb{N}} y + x.$$

The term `add_comm` of this type gives us a term of type $x + y =_{\mathbb{N}} y + x$ when passed terms $x, y : \mathbb{N}$. Unpacking the nested $\prod$-type notation, given an $x : \mathbb{N}$, we get a term of type

$$\prod_{y:\mathbb{N}} x + y =_{\mathbb{N}} y + x,$$

which, given any $y : \mathbb{N}$, gives us a term of type $x + y =_{\mathbb{N}} y + x$. Indeed, note that the codomains here depend on the term of the domain type that we pass in:

$$\prod_{y:\mathbb{N}} x + y =_{\mathbb{N}} y + x$$

depends (only) on the $x : \mathbb{N}$ that we passed in, and the type $x + y =_{\mathbb{N}} y + x$ depends on the $y : \mathbb{N}$ that we passed in. In this second case, we did not explicitly mention that $x+y =_{\mathbb{N}} y+x$ depends on x. This is because this term is 'captured' when applying the outer $\prod$-type to x. This concept may be familiar to people who have programmed in a programming language that has 'lambda functions', which allow one to define functions that 'capture' variables in the current context or scope that one is programming in. Effectively, the term x is *fixed* in the term inside the $\prod$-type.

Just as another example, the `maponpaths` lemma, or Lemma 2.3, may be written as a $\prod$-type. In `UniMath`, this is written as

```
∏ (a1 : A1) (a2 : A2) ..., B a1 a2 ...
```

We get a judgement

```
maponpaths :
    ∏ (A B : UU), ∏ (f : A -> B),
        ∏ (a1 a2 : A), a1 = a2 -> f a1 = f a2.
```

This judgement reflects the idea that 'by proving a theorem, we immediately get an object that satisfies the statement', exhibiting the purely constructive nature of HoTT.

Logically, one may interpret a $\prod$-type as a 'for all' statement, or we may see the $\prod$ notation as a $\forall$ clause. After all, a $\prod$-type

$$\prod_{a:A} B(a)$$

for $A : \mathcal{U}$ and $B : A \to \mathcal{U}$ gives a term of type $B(a)$ *for all* $a : A$. In other words, we may interpret the $\prod$-type above as the logical statement

$$\forall (a : A), B(a).$$

Example 2.4. *Plain function types are just $\prod$-types with a constant type family. That is to say, the type $A \to B$ is just the type*

$$\prod_{a:A} B.$$

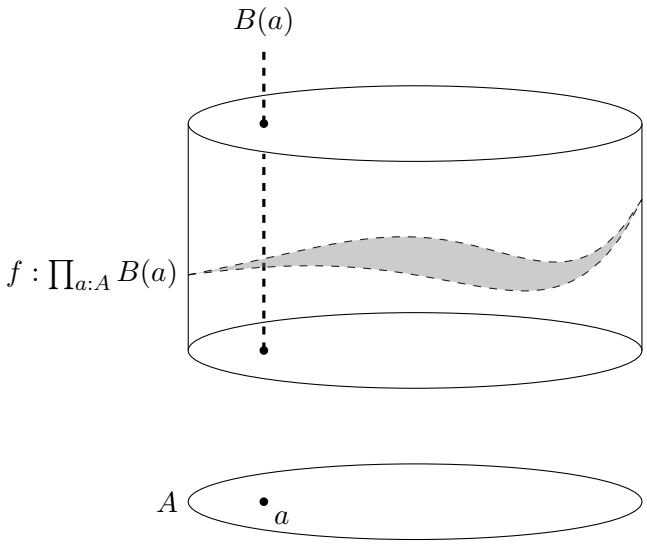


Figure 2.3: An example of a section $f : \prod_{a:A} B(a)$ from A into the fiber bundle $B : A \to \mathcal{U}$.

Equalities in $\prod$-types

Traditionally, one would expect two functions $f, g : A \to B$ to be the same whenever they are equal on all values. In HoTT, we can not simply do this. We define the notion of a homotopy $f \sim g$.

Definition 2.5. *Let $f, g : A \to B$ for two types $A, B : \mathcal{U}$. We define*

$$(f \sim g) := \prod_{a:A} f(a) =_B g(a).$$

We say that f and g are homotopic*.*

Indeed, since all constructions in HoTT are defined to be continuous, this defines a homotopy. We could in fact do the same for more general $\prod$-types. For a type $A : \mathcal{U}$ and a type family $B : A \to \mathcal{U}$, let $f, g : \prod_{a:A} B(a)$. Then we define

$$(f \sim g) := \prod_{a:A} f(a) =_{B(a)} g(a).$$

The fact that $(f \sim g)$ can be identified with $f = g$ is not a trivial thing. We commonly just *assume* this to hold. This is what we refer to as the *function extensionality* axiom. For details, we refer to [14, Section 2.9]. It is not uncommon to assume this axiom though. It is assumed in many places in the `UniMath` library itself [18], as well as in other formalization frameworks [19].

2.2.4 $\sum$-types

The other class of dependent types are the *$\sum$-types*, or *dependent pair types*. Set theoretically, they correspond with disjoint sums of sets, hence the name $\sum$-types. This correspondence will be clear shortly.

Given a type $A : \mathcal{U}$ and a type family $B : A \to \mathcal{U}$, a dependent pair type is written as

$$\sum_{a:A} B(a),$$

and terms consist of pairs (a, b) where $a : A$ and $b : B(a)$. We define the *canonical projections*

$$\mathrm{pr}_1 : \sum_{a:A} B(a) \to A : (a, b) \mapsto a$$

and

$$\mathrm{pr}_2 : \sum_{a:A} B(a) \to B : (a, b) \mapsto b,$$

defined as `pr1` and `pr2` in `UniMath`. We refer to A as the *base type* and the types $B(a)$ as the *fiber types*.

In formalization, $\sum$-types are very useful to define types that indicate the existence of terms of a certain type, that satisfy certain properties. This leads us to the logical interpretation of the $\sum$-type: the existence statement, or the $\exists$ clause. Effectively, one could read terms of the type

$$\sum_{a:A} B(a)$$

as 'I have an $a : A$, for which $b : B(a)$'. In `UniMath`, they are denoted as

```
∑ (a1 : A1) (a2 : A2) ..., B a1 a2 ...
```

using the same 'nested' notation as we did for $\prod$-types. Commonly in formalization, one wants to define a certain property on a type, and then a 'type of objects that have this property'. This is usually done using a $\sum$-type. For example, one might define a type `isaprop` that indicates whether some type A is a 'proposition' (this will be defined later). One can then define the 'type of propositions' as

$$\texttt{hProp} := \sum_{A:\mathcal{U}} \texttt{isaprop}\ A.$$

In HoTT, we may interpret $\sum$-types as a fiber bundle, as drawn out in Figure 2.4.

Example 2.6. *The most basic examples of dependent pair types are the* product types. *Given $A, B : \mathcal{U}$, we define the* product type $A \times B$ *as the type consisting of pairs (a, b) with $a : A$ and $b : B$. This is the same as a $\sum$-type with a constant type family:*

$$\sum_{a:A} B.$$

We use the same notation for the canonical projections on product types. Logically, one may read a product type as a logical conjunction $A \wedge B$.

Equalities in $\sum$-types

Like other types, we want to be able to compare terms of $\sum$-types. In this case, it may be interesting to see how equalities of $\sum$-types interact with equalities of terms in the base type. It should be clear that the canonical projection pr_1 projects an equality of terms $(a_1, b_1), (a_2, b_2) : \sum_{a:A} B(a)$ down to an equality $a_1 =_A a_2$, just from looking at Figure 2.5. We cannot however do this with pr_2, since b_1 and b_2 are, in general, *not* of the same type. This makes it tough to compare (a_1, b_1) and (a_2, b_2).

We can still define a way to do this though. Let us first introduce the *transport lemma* [14, Lemma 2.3.1], corresponding to the types `transportf` and `transportb` in `UniMath`, which stand for 'transport forwards' and 'transport backwards'. This lemma forms a parallel with the path lifting property in topology [14].

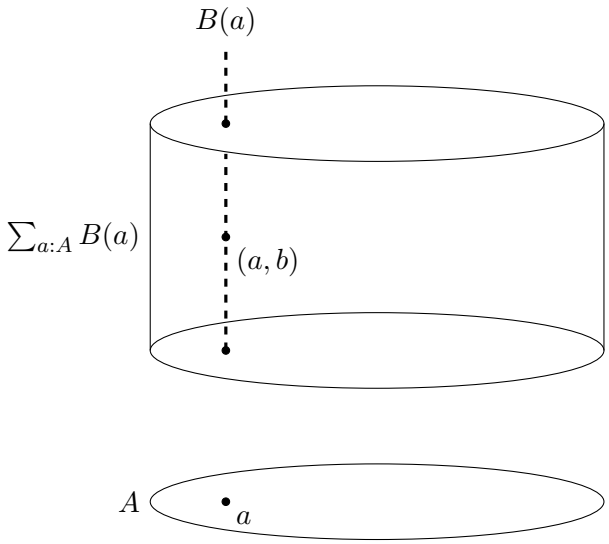


Figure 2.4: An example of a $\sum$-type $\sum_{a:A} B(a)$.

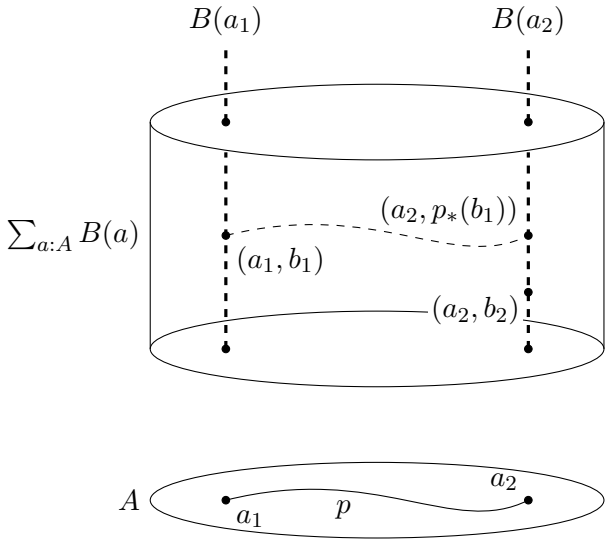


Figure 2.5: A homotopy theoretical interpretation of the transport lemma.

Lemma 2.7 (Transport)**.** *Given a type $A : \mathcal{U}$ and a type family $B : A \to \mathcal{U}$, a path $p : a_1 =_A a_2$ allows us to* transport *a pair $(a_1, b_1) : \sum_{a:A} B(a)$ to obtain a term $(a_2, p_*(b_1)) : \sum_{a:A} B(a)$.*

Or, in `UniMath`:

```
Lemma transportf {A : UU} (B : A -> UU) {a1 a2 : A} (p : a1 = a2) :
    B a1 -> B a2.
```

Homotopically, the idea of the proof is that a path in A lifts to one in $\sum_{a:A} B(a)$, see Figure 2.5.

Note that $p_*(b_1)$ and b_2 are of the same type: $B(a_2)$. This allows us to (indirectly) compare b_1 and b_2, by using a path in $B(a_2)$ from $(a_2, p_*(b_1))$ to (a_2, b_2) in Figure 2.5. In this way, one can show the following.

Lemma 2.8 (`two_arg_paths_b`, `two_arg_paths_f`)**.** *Suppose $(a_1, b_1), (a_2, b_2) : \sum_{a:A} B(a)$ for a type $A : \mathcal{U}$ and a type family $B : A \to \mathcal{U}$. Then we get a term*

$$p_\Sigma : (a_1, b_1) =_{\sum_{a:A} B(a)} (a_2, b_2)$$

if and only if we have terms

$$p_A : a_1 =_A a_2$$

and

$$p_B : (p_A)_*(b_1) =_{B(a_2)} b_2.$$

To make it clear that equalities are *dependent* in this way, we might denote them as

$$(a_1, b_1) =_* (a_2, b_2).$$

2.2.5 Summary

To finish off the introduction to basic HoTT, let us summarize the constructions we introduced, alongside their set theoretical analogue and their logical and homotopic interpretations in a table [14, Table 1].

Types	Sets	Logic	Homotopy
A	set	proposition	space
$a : A$	element	proof	point
$B : A \to \mathcal{U}$	predicate	family of sets	fibration
`unit`,`empty`	$\{ \emptyset \}$, $\emptyset$	true, false	$*$, $\emptyset$
$A \times B$	set of pairs	$A \wedge B$	product space
$A \to B$	set of functions	$A \implies B$	function space
$\sum_{a:A} B(a)$	disjoint sum	$\exists (a : A), B(a)$	total space
$\prod_{a:A} B(a)$	product	$\forall (a : A), B(a)$	space of sections
$a =_A b$	$\{ (a, a) \mid a \in A \}$	equality	path space

There are some more constructions on types, such as the disjoint union $A \sqcup B$ or inductive types. There are also more techniques we can use on types, such as induction or recursion. For these ideas, we refer to the HoTT book. The constructions from this chapter should be sufficient to read and understand the HoTT aspects of this thesis.

2.3 n-types

One of the basic notions of homotopy theory is that of a homotopy n-type: a space containing no interesting homotopy above dimension n. For example, a 0-type is effectively a set (with contractible path components), while a 1-type may contain non-trivial paths (like the circle). These types of spaces are often referred to as *n-truncated spaces*. We can define a similar notion in HoTT, though it turns out to be convenient to start two levels below 0, with (-1)-types being *mere propositions* and (-2)-types being *contractible types*.

For the latter of these two, we define a type `iscontr` as

```
Definition iscontr (A : UU) : UU :=
    ∑ (cntr : A), (∏ a : A, a = cntr).
```

Interpreting this definition, it says that there is a point `contr` in A, such that there is a path from a to `contr` for any $a : A$. Indeed, this corresponds with the notion of a contractible space in topology, as all constructions in HoTT are defined to be continuous. We define the notion of n-type recursively. In `UniMath`, this type is called `isofhlevel`:

$$\texttt{isofhlevel}\ (n : \mathbb{N})\ (A : \mathcal{U}) := \begin{cases} \texttt{iscontr}\ A & \text{if } n = -2 \\ \prod_{a,b:A} \texttt{isofhlevel}\ n'\ (a =_A b) & \text{if } n = n' + 1. \end{cases}$$

Let us interpret the most important cases: $n = -2$ through 0. For $n = -2$, we ask that the type is contractible. In other words, that it is non-empty and there is a path from any point to any other point. Types like this behave like the unit type `unit`.

2.3.1 Mere Propositions

More interesting and much more important are the (-1)-types: the *mere propositions*. The definition `isofhlevel` tells us that the *path space is contractible* for types of this truncation level. Effectively, this means that a type like this is either empty or contractible. After all, if a (-1)-type $A : \mathcal{U}$ is not empty, then the canonical point `contr` yields a path between any $a, b : A$, and any other path is the same as `contr` in a continuous way. Types like this can be interpreted as something that is either true or false: a logical proposition! This family of types is so important that `isofhlevel` for the level of propositions has its own name in `UniMath`: `isaprop`. This allows us to define a 'type of propositional types':

$$\texttt{hProp} := \sum_{A:\mathcal{U}} \texttt{isaprop}\ A.$$

Propositional types have a few important properties in formalization. The first one being the way they interact with $\sum$-types, and the second in the way one saves proofs in formalization.

It is very useful to define a $\sum$-type where the fiber types are all propositional. In `UniMath`, we call a type family $P : A \to \mathcal{U}$ over $A : \mathcal{U}$ for which every fiber is propositional a *predicate* (`isPredicate`), i.e.

```
Definition isPredicate {A : UU} (P : A -> UU) := ∏ a : A, isaprop (P a).
```

A $\sum$-type with propositional fibers, i.e. a type

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

can be used to mimic sets that are defined through predicates in set theory: $\{\, a \in A \mid P(a) \,\}$. Most importantly though, it is very easy to compare terms of $\sum$-types like this, as we only have to compare the projections on the first coordinate. After all, Lemma 2.8 combined with the contractibility of the path space of the fibers, tells us the following.

Lemma 2.9 (`subtypePath`)**.** *Suppose we have terms $(a_1, p_1), (a_2, p_2)$ of a $\sum$-type*

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

with a type $A : \mathcal{U}$ and a predicate $P : A \to \mathcal{U}$, then we get a term

$$(a_1, p_1) =_{\sum_{a:A} P(a)} (a_2, p_2)$$

whenever we have a term

$$a_1 =_A a_2.$$

Example 2.10 (`isapropempty`, `isapropunit`)**.** *The most basic examples of propositional types are the empty type* `empty` *and the unit type* `unit`*.*

Furthermore, knowing that types are propositional allows us to optimize the proofs in the proof assistant a lot. In `UniMath`, there are two ways to finish a proof: with `Defined` and with `Qed`. The former makes the proof *transparent*, allowing us to see how exactly some term is built up. This may be useful when defining a function or a specific composition of morphisms in a category for example. Suppose we want to define a map `square : nat -> nat`. We would want the proof to be transparent, since it may allow the proof assistant to deduce that `square 0` is definitionally equal to `0` for example.

Saving a proof with `Qed` makes the proof *opaque*. Basically, when using the term corresponding to a lemma saved this way in another proof, the proof checker will know that a term

like this exists, but it will not be able to 'unfold' the term. This makes it so that the terms that the proof checker constructs and checks are a lot smaller and simpler. If the type of a lemma or definition is of a propositional type, we in fact do not care what the term looks like, since the type will be contractible anyway. We just want to know that such a term exists! This allows us to optimize our proofs in a way that the proof checker becomes much faster, and the produced terms become easier to comprehend as a human. For comparison, a proof term constructed with only 6 tactics may already take up over 60 lines to print when saved with `Defined`, whereas saving a term with `Qed` constructs a term that is always only a single line.

2.3.2 Sets

The last n-type that we will discuss in more detail is the level 0. As mentioned before, these correspond to sets, and the corresponding instance of `isofhlevel` also has its own name in `UniMath`: `isaset`. Like `hProp`, we define a 'type of sets':

$$\texttt{hSet} := \sum_{A:\mathcal{U}} \texttt{isaset}\ A.$$

Interpreting what it means for a type $A : \mathcal{U}$ to be of this truncation level: for any terms $a, b : A$ the path space is propositional. That is to say, there is either no path, or the path space is contractible. This means that a type like this consists of contractible connected components: it is a set!

The most important property of sets we use in formalization is that the path spaces are propositional. This is useful when defining $\sum$-types of the form

$$\sum_{a:A} B_1(a) =_{B(a)} B_2(a)$$

for a type $A : \mathcal{U}$, a type family $B : A \to \mathcal{U}$ and dependent functions $B_1, B_2 : \prod_{a:A} B(a)$, where every $B(a)$ is a set. This makes it so the fiber types of this $\sum$-type are all propositional, like in the previous section.

Example 2.11 (`isasetaprop`)**.** *Any proposition type is a set. More generally, any n-type is also a n'-type for any $n' \geq n$.*

Example 2.12 (`isasetnat`)**.** *The type $\mathbb{N}$ (or* `nat` *in* `UniMath`*) of natural numbers is a set.*

Other n-types, though interesting in their own right, are not very interesting for the purposes of this thesis. What is interesting and useful though, is the concept of *propositional truncation*, allowing us to define propositional types from any other type.

2.3.3 Propositional Truncation

Sometimes, instead of needing a term of a certain type A, it is sufficient (or perhaps even necessary!) to only know of the mere existence of a term of A. That is to say, we may only want to know that 'a term of type A exists', and do not need to know exactly what this term is or how it is constructed. We can do this by *propositionally truncating* A. The idea is to define a (propositional) type that indicates whether A is *inhabited*.

Definition 2.13 (`ishinh_UU`, `isapropishinh`)**.** *Let $A : \mathcal{U}$, we define the* propositional truncation *of A by*

$$\|A\| := \prod_{P:\texttt{hProp}} ((A \to P) \to P) : \texttt{hProp}.$$

```
Definition ishinh_UU (A : UU) : UU :=
  ∏ P : hProp, ((A -> P) -> P)
```

Remark. *The fact that this type is indeed propositional uses function extensionality (*isweqtoforallpathsAxiom*, used in the proof of* impred*).*

The type says that for any propositional type P (which we can interpret to mean either `empty` or `unit`), we have a term of type

$$(A \to P) \to P.$$

Now if we take P to be `unit`, this tells us that we can find a term $(A \to \texttt{unit}) \to \texttt{unit}$. This is simple, we just map any $f : A \to \texttt{unit}$ to the canonical element $\texttt{tt} : \texttt{unit}$. This makes sure that any two elements in $\|A\|$ are equal.

More interesting is the case of the empty type. It tells us that we can map any term $f : A \to \texttt{empty}$ to a point in `empty`. Obviously, such a point does not exist, so this is only possible if $A \to \texttt{empty}$ is the empty type as well, which true only if A is *not* empty. After all, if A is the empty type, the identity function $\mathrm{id}_{\texttt{empty}} : \texttt{empty} \to \texttt{empty}$ is still defined. This case makes sure that A is not empty.

Remark. *It is important to remember that we can* not *(in general) obtain a term of type A given a term of type $\|A\|$.*

Remark. *In fact, we can define a function* `empty` $\to A$ *for any type $A : \mathcal{U}$. This corresponds with the idea of* proof by contradiction*: a false assertion implies anything! It can also be interpreted as the unique map from the initial object $\emptyset$ in the category of topological spaces. In a similar vein, $A \to$* `empty` *is how one defines* logical negation *in HoTT.*

The propositional truncation has a universal property.

Lemma 2.14 (hinhuniv)**.** *Let $A : \mathcal{U}$ and $P :$* `hProp` *and let $f : A \to P$. Then we can construct a term of type P if we have a term $a : \|A\|$.*

Intuitively, this tells us that we can show that a proposition is true if it holds for all $a : A$ and we know that A is not empty. A more general statement also holds, giving us an easy way to work with propositional truncations.

Lemma 2.15 (hinhfun)**.** *Let $A, B : \mathcal{U}$ and let $f : A \to B$. Then we can construct a term of type $\|B\|$ if we have a term $a : \|A\|$.*

This statement tells us that B is not empty if A is not empty and we have a function $f : A \to B$. This, in combination with the canonical projection

$$\texttt{hinhpr} : A \to \|A\|,$$

give us a way to 'unpack' the propositional truncation in some proofs. In formalization, this allows us to work with 'actual terms $a : A$', as opposed to just the mere existence of such terms.

A common construction using propositional truncation is that of *mere existence*. The truncation of a $\sum$-type effectively tells us that 'there is some' term of the base type, for which there is a term of the fiber type. This is useful, since $\sum$-types are in general, and in fact almost never, propositional. Like for other types, we may just want to know that there is such a term, and we do not really care what it is precisely. This construction is so common, that it is denoted using the logical notation, as

$$\exists_{a:A} B(a) := \left\| \sum_{a:A} B(a) \right\|.$$

Remark. *One can define a more general notion of* truncated types*, allowing one to truncate a type at* any *homotopy level, not just -1 [14, Chapter 7].*

2.4 Univalence

Perhaps one of the most powerful axioms one may assume in HoTT is the *Univalence Axiom*. It provides a simple notion of equality of types to work with. Generally though, one does not want to assume this axiom. Let us first introduce a basic notion of *type equivalence*.

Definition 2.16 (isweq)**.** *Given two types $A, B : \mathcal{U}$, we say that A* is equivalent to *B if there is a map $f : A \to B$ such that*

$$\texttt{isweq}\ f := \left(\sum_{g:B\to A} g\cdot f \sim \mathrm{id}_B\right) \times \left(\sum_{h:B\to A} f\cdot h \sim \mathrm{id}_A\right).$$

We say that f is a weak equivalence*. We denote this as $A \simeq B$.*

This gives some notion of 'isomorphisms' between types. Obviously, we can find a term

$$\texttt{eqweqmap} : (A =_{\mathcal{U}} B) \to (A \simeq B).$$

The Univalence Axiom says that this map `eqweqmap` is itself a weak equivalence.

Definition 2.17 (The Univalence Axiom)**.** *The map* `eqweqmap` *is an equivalence.*

This allows us to work with equivalent types as if they are the same. This is an assumption one would rather not use, but it is important to learn about the Univalence Axiom in order to understand the notion of *univalent categories* that we are about to introduce.

The Univalence Axiom is a stronger axiom than function extensionality, used to compare (dependent) function types. In fact, univalence implies function extensionality [14]. When proving statements in HoTT, one would of course rather not use either of these axioms, but function extensionality may be a necessary evil. Function extensionality is an axiom that is assumed in many other formalization frameworks, such as the Lean proof checker [19], and is used even in basic constructions in HoTT and `UniMath`, such as that of propositional truncation.

At this point, it is good to point out another difference between HoTT and other foundations, lying in the system of logic defining the foundations. The logic of types is fundamentally different from 'classical logic', as the *law of excluded middle* (LEM) need not be true. This law states that any proposition is either true, or its negation is true. The corresponding statement in HoTT is in fact inconsistent with the Univalence Axiom. When stated only for propositional types, though not necessarily true, it may be assumed and *is* consistent with the Univalence Axiom, with similar consequences as in set-theory based mathematics [14]. To elaborate on this slightly: a naive type-theoretic analogue to the LEM would mean that we would be given a term of any non-empty type $A : \mathcal{U}$, providing some sort of canonical choice operator. After all, proving 'A' would be the same as constructing an term of this type, and the LEM would give such a term, knowing that A is not empty. The LEM may be true for some types, which are called *decidable* (`decidable` in `UniMath`).

Similarly, the *axiom of choice* need not be true in general, but it may be assumed, again with similar consequences as in set-theory based mathematics. In the theory of model categories, it turned out to be necessary to assume the axiom of choice for certain statements to hold. The more coherent theory of natural weak factorization systems 'fixes' these constructive issues.

2.5 Category Theory

Finally, we introduce some notions of category theory in HoTT. After all, this thesis is about formalizing concepts in *model category theory* in HoTT. In set-based mathematics, a category

consists of a *set* A_0 of objects, and for each $X, Y \in A_0$ a *set* of morphisms $\mathrm{Hom}(X, Y)$. In HoTT, one might naively define a category as a *type* A_0 of objects and *types* of morphisms. If we allow these types to have arbitrary higher homotopies, this gives us some notion of an $(\infty, 1)$-category, which we call a *precategory* (also `precategory` in `UniMath`).

However, if we restrict the types $\mathrm{Hom}(X, Y)$ of morphisms to be mere sets, we get the notion of a *category* (also `category` in `UniMath`). In the rest of this thesis, we will only be considering categories. Interestingly enough, the 'homset' refinement is sufficient to show the results that we want. There are even more refined notions of categories though. If one also requires the type A_0 of objects to be a set, we speak of a *strict category* (or a `setcategory` in `UniMath`).

Example 2.18 (`hSet`)**.** *The precategory* **SET** *of sets (*`HSET` *in* `UniMath`*), defined as one expects, is a category.*

Example 2.19 (`type_precat`)**.** *The precategory of types, where* $A_0 := \mathcal{U}$ *with morphisms* $\mathrm{Hom}(A, B) := A \to B$ *is* not *a category.*

Example 2.20. *The precategory* **CAT** *of categories, where objects are categories and morphisms are functors, is* not *a category.*

Example 2.21 (`cat_of_setcategory`)**.** *The precategory of strict categories (or* `setcategory` *in* `UniMath`*), where objects are* strict *categories and morphisms are functors, is in fact a category.*

Perhaps the most important refinement is the notion of a *univalent category*. Consider the following.

Lemma 2.22 (`idtoiso`)**.** *Let* A *be a precategory and* $X, Y : A_0$ *objects. Then any path* $p : X =_{A_0} Y$ *yields an isomorphism*

$$\texttt{idtoiso}\ p : \mathrm{Hom}(X, Y).$$

This parallels the map `eqweqmap`, yielding an equivalence of types given an equality. In a *univalent category*, one effectively assumes the equivalent of the Univalence Axiom for this analogue of `eqweqmap`.

Definition 2.23 (`is_univalent`)**.** *Let* A *be a category. Then we say that* A *is a* univalent category *if the map* `idtoiso` *is a weak equivalence. That is to say, for any* $X, Y : A_0$*, any isomorphism* $X \cong Y$ *corresponds exactly with a path* $X =_{A_0} Y$*.*

Example 2.24 (`is_univalent_HSET`)**.** *The category* `HSET` *of sets is a univalent category.*

A univalent category is in fact the closest notion to a category in set-based mathematics. Interestingly enough, when formalizing model category theory and the small object argument, we actually *never* need the notion of a univalent category. It is interesting to see how we only need a weaker notion of categories in order to show that the results hold.

Chapter 3

Displayed Categories

One tool we will use extensively in our formalization is the theory of *displayed categories*. Displayed categories provide a simple way to construct new categories from existing ones, usually by adding some sort of data or properties to the objects and morphisms. This theory has been developed and formalized in `UniMath` quite recently by Benedikt Ahrens and Peter LeFanu Lumsdaine [11]. We will introduce the necessary theory that we will be using in our rephrasing and formalization of model category theory and the small object argument. Throughout this chapter, we will introduce the concepts following their article and their formalization. We also introduce a new construction that is not in their article or the `UniMath` library, as well as some very useful and important examples which we will use in the rest of this thesis.

3.1 Definitions

It is common practice to construct a new category $\mathcal{D}$ out of another category $\mathcal{C}$ by adding data to the objects and morphisms. This can be expressed in terms of a (forgetful) functor $F : \mathcal{D} \longrightarrow \mathcal{C}$. Instead of having a mapping from the objects and morphisms of $\mathcal{D}$ to those of $\mathcal{C}$, it may be useful to instead index them as families of objects and morphisms 'lying over' those of $\mathcal{C}$, see Figure 3.1. We define a displayed category as follows.

Definition 3.1 (`disp_cat`)**.** *A* displayed category $\mathcal{D}$ *over a category* $\mathcal{C}$ *consists of the following data:*

(i) *For each object* $X : \mathcal{C}$*, a type* $\mathcal{D}_X$ *of 'objects over* X*'.*

(ii) *For each morphism* $f : X \to Y$ *with* $X, Y : \mathcal{C}$*, and for each displayed object* $\overline{X} : \mathcal{D}_X$ *and* $\overline{Y} : \mathcal{D}_Y$ *a set of 'morphisms from* $\overline{X}$ *to* $\overline{Y}$ *over* f*', denoted* $\overline{X} \to_f \overline{Y}$*.*

(iii) *For each object* $X : \mathcal{C}$ *and each* $\overline{X} : \mathcal{D}_X$*, a morphism* $1_{\overline{X}} : \overline{X} \to_{\mathrm{id}_X} \overline{X}$*.*

(iv) *For all* $X, Y, Z : \mathcal{C}$*,* $\overline{X} : \mathcal{D}_X$*,* $\overline{Y} : \mathcal{D}_Y$*,* $\overline{Z} : \mathcal{D}_Z$ *and* $f : X \to Y$*,* $g : Y \to Z$*, a composition function*

$$\left(\overline{X} \to_f \overline{Y}\right) \times \left(\overline{Y} \to_g \overline{Z}\right) \to \left(\overline{X} \to_{f \cdot g} \overline{Z}\right),$$

denoted like the usual composition, $(\overline{f}, \overline{g}) \mapsto \overline{f} \cdot \overline{g}$*.*

such that, for suitable inputs,

(i) $\overline{f} \cdot 1_Y =_* \overline{f}$*.*

(ii) $1_X \cdot \overline{f} =_* \overline{f}$*.*

(iii) $\overline{f} \cdot (\overline{g} \cdot \overline{h}) =_* (\overline{f} \cdot \overline{g}) \cdot \overline{h}$*.*

where the equalities $=_*$ *are not actual equalities, since the equality would be ill-typed. Rather, one of the sides of the equation has to be transported along the corresponding equality in* $\mathcal{C}$ *(right identity, left identity or associativity), as explained in Chapter 2.*

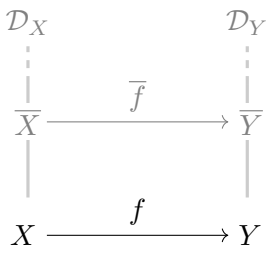


Figure 3.1: Intuitive picture of a displayed category $\mathcal{D}$ over $\mathcal{C}$. A morphism $f : X \to Y$ of $\mathcal{C}$ is drawn, together with families $\mathcal{D}_X$ and $\mathcal{D}_Y$ of displayed object and a displayed morphism $\overline{f} : \overline{X} \to_f \overline{Y}$.

This definition allows us to easily define an 'actual' category from a displayed category, simply by forming (dependent) pairs of objects and morphisms of $\mathcal{C}$ with corresponding objects and morphisms of $\mathcal{D}$. We call this category $\mathcal{D}^{\text{tot}}$, or the *total category of* $\mathcal{D}$.

Definition 3.2 (`total_category`)**.** *Given a category $\mathcal{C}$ and a displayed category $\mathcal{D}$ over $\mathcal{C}$, we define the* total category of $\mathcal{D}$*, denoted $\mathcal{D}^{tot}$ as*

- ***Objects*** *a type $\sum_{X:\mathcal{C}} \mathcal{D}_X$.*
- ***Morphisms*** *from the dependent pair $(X, \overline{X})$ to the dependent pair $(Y, \overline{Y})$ as the set $\sum_{f:X\to Y} \overline{X} \to_f \overline{Y}$.*

with the obvious unit and composition.

Remark. *It is easy to verify that the morphisms in $\mathcal{D}^{tot}$ indeed form a set, since it is a $\sum$-type of sets over a set. This makes it so $\mathcal{D}^{tot}$ is indeed a category, and not merely a precategory.*

Example 3.3. *Perhaps the most basic displayed category one could think of is a category as a displayed category over itself, where both displayed objects and morphisms are simply the unit type* `unit`*. In other words, $\mathcal{D}_X :=$ `unit`, as well as $\overline{X} \to_f \overline{Y} :=$ `unit` for any choice of (displayed) objects and morphisms. This makes it so the total category has objects $\sum_{X:\mathcal{C}}$ `unit` and morphisms $\sum_{f:X\to Y}$ `unit`, making it, in some sense, 'equal' to $\mathcal{C}$ itself.*

Example 3.4 (`arrow`)**.** *We define the 'arrow category' $\mathcal{C}^{\mathbf{2}}$ of a category $\mathcal{C}$, where the objects are morphisms in $\mathcal{C}$ and arrows are commutative diagrams, as a displayed category over the product category $\mathcal{C} \times \mathcal{C}$. We simply define it as*

- ***Displayed objects*** *over $(X, Y) : \mathcal{C} \times \mathcal{C}$ as the type $X \to Y$.*
- ***Displayed morphisms*** *between displayed objects $f : X \to Y$ and $g : A \to B$ over a morphism $(h, k) : (X, Y) \to (A, B)$ as the propositional type $f \cdot k =_{(X\to B)} h \cdot g$.*

Example 3.5 (`three`)**.** *We define the 'three category' $\mathcal{C}^{\mathbf{3}}$ of a category $\mathcal{C}$, where the objects are diagrams of shape $0 \to 1 \to 2$ and morphisms are commutative diagrams as a displayed category over $\mathcal{C}^{\mathbf{2}}$. We define it as*

- ***Displayed objects*** *over $f : X \to Y : \mathcal{C}^{\mathbf{2}}$ as the type*

$$\sum_{(E_f:\mathcal{C})} \sum_{(f_{01}:X\to E_f)} \sum_{(f_{12}:E_f\to Y)} f_{01} \cdot f_{12} =_{(X\to Y)} f.$$

- ***Displayed morphisms*** *between displayed objects (E_f, f_{01}, f_{12}) and $(E_{f'}, {f'}_{01}, f'_{12})$ over a morphism (g_{00}, g_{22}) as the type*

$$\sum_{g_{11}:E_f\to E_{f'}} \left(f_{01} \cdot g_{11} =_{(X\to E_{f'})} g_{00} \cdot f'_{01}\right) \times \left(f_{12} \cdot g_{22} =_{(E_f\to Y')} g_{11} \cdot f'_{12}\right),$$

providing a morphism g_{11} between the 'middle objects' such that the following diagram commutes

$$\begin{array}{ccc} X & \xrightarrow{g_{00}} & X' \\ {\scriptstyle f_{01}}\downarrow & & \downarrow{\scriptstyle f'_{01}} \\ E_f & \xrightarrow{g_{11}} & E_{f'} \\ {\scriptstyle f_{12}}\downarrow & & \downarrow{\scriptstyle f'_{12}} \\ Y & \xrightarrow[g_{22}]{} & Y' \end{array}$$

Let us also define equivalents to constructions like functors and natural transformations on displayed categories, in a way that they 'lift' to functors and natural transformations on the total categories. We do this by defining them on the displayed objects and morphisms, with analogous constraints.

Definition 3.6 (`disp_functor`)**.** *Given categories $\mathcal{C}, \mathcal{C}'$, displayed categories $\mathcal{D}, \mathcal{D}'$ over $\mathcal{C}, \mathcal{C}'$ respectively and a functor $F : \mathcal{C} \longrightarrow \mathcal{C}'$, a* displayed functor over F*, denoted as $G : \mathcal{D} \longrightarrow_F \mathcal{D}'$, consists of*

- *A map of displayed objects, i.e. for each $X : \mathcal{C}$, a map $G_X : \mathcal{D}_X \to \mathcal{D}'_{F(X)}$, where the object X is commonly omitted.*
- *A map of displayed morphisms, i.e. for each morphism $f : X \to Y$ in $\mathcal{C}$, and displayed objects $\overline{X}$ and $\overline{Y}$ over X and Y respectively, a map of displayed morphisms $\big(\overline{X} \to_f \overline{Y}\big) \to \big(G(\overline{X}) \to_{F(f)} G(\overline{Y})\big)$.*

such that the evident analogues of the axioms for normal functors hold. Again, one has to account for the displayed types using transports.

In a similar fashion, we define displayed natural transformations, providing a way to transform between displayed functors, similar to how one transforms between functors with normal natural transformations.

Definition 3.7 (`disp_nat_trans`)**.** *Let $F, F' : \mathcal{C} \longrightarrow \mathcal{C}'$ be functors, and $\alpha : F \Longrightarrow F'$ a natural transformation. Let G, G' be displayed functors from $\mathcal{D}$ to $\mathcal{D}'$ lying over F and F' respectively. Then a* displayed natural transformation *over α consists of a dependent function*

$$\beta : \prod_{X:\mathcal{C}} \prod_{\overline{X}:\mathcal{D}_X} G(\overline{X}) \to_{\alpha(X)} G'(\overline{X}),$$

such that for any $f : X \to Y$ and $\overline{f} : \overline{X} \to_f \overline{X}$, we have that the following diagram commutes

$$\begin{array}{ccc} G(\overline{X}) & \xrightarrow{G\overline{f}} & G(X') \\ {\scriptstyle \beta(\overline{X})}\downarrow & & \downarrow{\scriptstyle \beta(\overline{Y})} \\ G'(\overline{X}) & \xrightarrow{G'\overline{f}} & G'(\overline{Y}) \end{array}$$

These definitions allow us to rephrase a lot of category theoretical language in simpler terms, at least for formalization. For example, a common construction Garner uses in his paper [9] is defining morphisms between structures that lie over the arrow category $\mathcal{C}^{\mathbf{2}}$ in terms of morphisms in the slice category $\mathbf{CAT}/\mathcal{C}^{\mathbf{2}}$, like in the diagram below

$$\begin{array}{ccc} \mathsf{L}\text{-}\mathbf{Map} & \xrightarrow{\tau_l} & \mathsf{L}'\text{-}\mathbf{Map} \\ & {\scriptstyle U_{\mathsf{L}}}\searrow \quad \swarrow{\scriptstyle U_{\mathsf{L}'}} & \\ & \mathcal{C}^{\mathbf{2}} & \end{array}$$

For now, the exact meanings of the categories in this diagram are irrelevant, but the diagonal maps are forgetful functors. Instead, if we defined the categories as proper displayed categories, we could define this as a displayed functor $\tau_l : \mathsf{L}\text{-}\mathbf{Map} \longrightarrow_{\mathrm{id}_{\mathcal{C}^\mathbf{2}}} \mathsf{L}'\text{-}\mathbf{Map}$. Along with an easier definition, it is also much easier to reason with something like this, as equalities in $\mathcal{C}^\mathbf{2}$ that one obtains by tracing Garner's diagram need not be definitional, whereas a similar construction using displayed categories will *always* yield definitional equalities. This is desirable, as definitional equalities are much easier to work and reason with than propositional ones, as explained in Chapter 2.

3.2 Constructions

Given a category $\mathcal{C}$ and a displayed category $\mathcal{D}$ over $\mathcal{C}$, there is a canonical projection (`pr1_category` in `UniMath`), which is a functor $\pi_1^{\mathcal{D}} : \mathcal{D}^{\mathrm{tot}} \longrightarrow \mathcal{C}$ projecting a pair $(X, \overline{X}) : \sum_{X:\mathcal{C}} \mathcal{D}_X$ down to X. It turns out to be very useful to define sections of this canonical projection. This is something we will be using extensively when defining functorial factorizations. We define a type of *sections* of a displayed category to do this.

Definition 3.8 (`section_disp`)**.** *Given a category $\mathcal{C}$ and a displayed category $\mathcal{D}$ over $\mathcal{C}$, a* section *from $\mathcal{C}$ to $\mathcal{D}$ consists of a dependent function of objects $F : \prod_{X:\mathcal{C}} \mathcal{D}_X$ and a corresponding dependent function, also denoted by F, of type $\prod_{f:X\to Y} F(X) \to_f F(Y)$, such that $F(\mathrm{id}_X) = 1_{F(X)}$ and $F(f\cdot g) = F(f)\cdot F(g)$ for compatible morphisms f and g in $\mathcal{C}$.*

Remark. *The two maps in the definition of a section are essentially just two choice functions. One could also view the section as a displayed functor from the category $\mathcal{C}$ as displayed category over itself to $\mathcal{D}$. We do not define them in this way though, since the formalization would become riddled with canconical projections.*

Remark (`section_functor`)**.** *Like displayed functors, a section $\mathcal{C} \longrightarrow \mathcal{D}$ of a displayed category $\mathcal{D}$ over $\mathcal{C}$ lifts to a functor $\mathcal{C} \longrightarrow \mathcal{D}^{tot}$ to the total category.*

In formalization, this makes it so that for any section $F : \mathcal{C} \longrightarrow \mathcal{D}$, and any $X : \mathcal{C}$, the composite $F\cdot \pi_1^{\mathcal{D}}(X)$ is in fact *definitionally equal* to X. Note that we do not say that the functors themselves are definitionally equal (they are in fact only *propositionally* equal), but the images of any object or any morphism are. If we were to define displayed categories and sections in a more naive way, like the classical approach of using a forgetful functor $\mathcal{D} \longrightarrow \mathcal{C}$ to define displayed categories, there is no way to obtain this definitional equality, at least not without explicitly knowing both functors. This would make things very hard and inconvenient to reason with, especially in a formalization.

Now we will define a hand-crafted notion of natural transformations between sections. We prefer this over using natural transformation between the lifted functors, since the data for a section lies in the section itself, and definitional information is lost in the lifted functor that arises from it. If we were to use natural transformations between the lifted functors as morphisms between sections, we would not be able to turn the image of a section (through its lifted functor) back down to a section, since we have lost this necessary definitional information, witnessing that it is in fact a section, not just any functor.

Definition 3.9. `section_nat_trans_disp` *Let F and F' be sections of a displayed category $\mathcal{D}$ over $\mathcal{C}$. A* natural transformation of sections *from F to F' is a family of displayed morphisms*

$$\prod_{X:\mathcal{C}} F(X) \to_{\mathrm{id}_X} F'(X),$$

such that 'the appropriate diagrams' commute.

Remark. *In this definition, 'the appropriate diagrams' again involves some dependent equalities. To explain this a bit more, for a general morphism $f : X \to Y$ in $\mathcal{C}$, the following diagram is* not *well-typed:*

$$\begin{array}{ccc} F(X) & \xrightarrow{Ff} & F(Y) \\ \downarrow & & \downarrow \\ F'(X) & \xrightarrow[F'f]{} & F'(Y) \end{array}$$

This is because going around the left and bottom yields a displayed morphism over $\mathrm{id}_X \cdot f$, whereas going around the top and right yields a displayed morphism over $f \cdot \mathrm{id}_Y$. This is a subtle difference, and it can be solved by transporting this diagram along the appropriate equalities (relating $\mathrm{id}_X \cdot f$ and f, and $f \cdot \mathrm{id}_Y$ and f) to allow for a well-typed naturality axiom:

```
Definition section_nat_trans_disp_axioms
    {C : category}
    {D : disp_cat C}
    {F F': section_disp D}
    (nt : section_nat_trans_disp_data F F') : UU :=
  ∏ x x' (f : x --> x'),
    transportf _
      (id_right _ @ !(id_left _))
      (section_disp_on_morphisms F f ;; nt x') =
    nt x ;; section_disp_on_morphisms F' f.
```

where we transport the composite on the left-hand side along the path

```
id_right _ @ !(id_left _)
```

to obtain a well-typed equality.

3.3 The Structure Identity Principle

The HoTT book introduces the 'Structure Identity Principle' (SIP), which expresses that isomorphic structures on a precategory are identical. It turns out that we can use the data from a *standard structure* to allow for an easier way of defining displayed categories, in case the structure we wish to add to our base category allows for it. In fact, the formulation of the SIP in the HoTT book effectively states that displayed categories (produced from structure data) indeed provide a category.

Definition 3.10. *A* notion of structure (P, H) *over a precategory $\mathcal{C}$ consists of the following [14].*

- *A type family $P : \mathcal{C}_0 \to \mathcal{U}$.*
- *For each $A, B : \mathcal{C}$ and $X : P(A)$ and $Y : P(B)$ and $f : A \to B$, a proposition $H_{X,Y}(f)$.*

such that H is closed under identity and composition.

Providing this data allows us to define a displayed category on $\mathcal{C}$ without having to explicitly provide a displayed identity and composition, as this can be produced from the propositional type H [11, `disp_cat_from_SIP_data`].

Example 3.11. *In fact, the arrow category $\mathcal{C}^{\mathbf{2}}$ can be defined in this way as a displayed category over the binary product of a category $\mathcal{C}$ with itself. This is because the added data to the morphisms is a propositional type, as $\mathcal{C}$ is a category and a commutative diagram is simply an equality between morphisms.*

Example 3.12. *The three category $\mathcal{C}^{\mathbf{3}}$ can* not *be defined in this way, as the data we add to the morphisms is both an extra morphism (between the 'middle objects'), and the commutativity constraint of the two relevant diagrams. This type is not propositional in general.*

3.4 Examples

We have seen two important examples of displayed categories in this chapter: the arrow category $\mathcal{C}^{\mathbf{2}}$ and the three category $\mathcal{C}^{\mathbf{3}}$. These two categories will play an important role in this thesis. This section clarifies some notation that we will be using on these categories, and contains some basic results about them.

3.4.1 The Arrow Category

Given $f : X \to Y$ as object in the arrow category, we label the objects X and Y in a by $\operatorname{dom} f$ and $\operatorname{cod} f$ respectively. Given a morphism $\gamma : f \to g$ in the arrow category, we will label the *morphism on the domains* as γ_{00} and the *morphism on the codomains* as γ_{11}. As a diagram, this looks like

$$\begin{array}{ccc} X & \xrightarrow{\gamma_{00}} & A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ Y & \xrightarrow[\gamma_{11}]{} & B \end{array}.$$

From the definition of the total category, morphisms between arrows $f : X \to Y$ and $g : A \to B$ are a type

$$f \to g := \sum_{(\gamma_{00},\gamma_{11}):(X,Y)\to(A,B)} f \cdot \gamma_{11} = \gamma_{00} \cdot g.$$

Now, since we assume $\mathcal{C}$ to be a category, the fiber type is propositional. The `subtypePath` lemma (Lemma 2.9) then tells us that

Lemma 3.13 (`arrow_mor_eq`, `arrow_mor00_eq`, `arrow_mor11_eq`)**.** *Given two morphisms $\gamma, \gamma' : f \to g$ of $\mathcal{C}^{\mathbf{2}}$, we get a term $\gamma =_{f\to g} \gamma'$ if and only if we have terms*

$$\gamma_{00} =_{\operatorname{dom} f \to \operatorname{dom} g} {\gamma'}_{00}$$

and

$$\gamma_{11} =_{\operatorname{cod} f \to \operatorname{cod} g} {\gamma'}_{11}.$$

One other common property we will use later on in this thesis is that

Lemma 3.14 (`arrow_colims`)**.** *Suppose $\mathcal{C}$ is a cocomplete category, then the arrow category $\mathcal{C}^{\mathbf{2}}$ is cocomplete.*

The colimits are defined as one would expect. The colimit of a diagram in the arrow category is the canonical arrow between the colimits of the projected diagrams on the domain and codomain respectively. Just to clarify this idea, one could think of the colimit of a diagram $\{ f_v : X_v \to Y_v \}_v$in the arrow category as the following diagram

$$\begin{array}{ccccccccc} \dots & \cdots\!\!\rightarrow & X_v & \longrightarrow & X_{v+} & \longrightarrow & X_{v++} & \cdots\!\!\rightarrow \dots \cdots\!\!\rightarrow & \operatorname{colim} X_v \\ & & {\scriptstyle f_v}\downarrow & & {\scriptstyle f_{v+}}\downarrow & & \downarrow{\scriptstyle f_{v++}} & & \downarrow{\scriptstyle \operatorname{colim} f_v} \\ \dots & \cdots\!\!\rightarrow & Y_v & \longrightarrow & Y_{v+} & \longrightarrow & Y_{v++} & \cdots\!\!\rightarrow \dots \cdots\!\!\rightarrow & \operatorname{colim} Y_v \end{array}$$

By the *diagram projected on the domain* (`project_diagram00`) we mean the diagram $\{ X_v \}_v$ in $\mathcal{C}$, and the *diagram projected on the codomain* (`project_diagram11`) is the diagram $\{ Y_v \}_v$ in $\mathcal{C}$. One can also show the following.

Lemma 3.15 (`project_colimcocone00`, `project_colimcocone11`)**.** *Suppose $\{ f_v : X_v \to Y_v \}$ is a diagram in $\mathcal{C}^{\mathbf{2}}$, and $f : X_\infty \to Y_\infty$ is a colimit for this diagram. Then X_∞ is a colimit for the diagram projected on the domains, and similarly, Y_∞ is a colimit for the diagram projected on the codomains.*

3.4.2 The Three Category

In the three category $\mathcal{C}^{\mathbf{3}}$, we commonly denote an object that lies over an arrow $f : X \to Y$ as

$$X \xrightarrow{f_{01}} E_f \xrightarrow{f_{12}} Y.$$

A morphism $\gamma : f \to g$, for objects $f : X \to E_f \to Y$ and $g : A \to E_g \to B$ is denoted as

$$\begin{array}{ccc} X & \xrightarrow{\gamma_{00}} & A \\ {\scriptstyle f_{01}}\downarrow & & \downarrow{\scriptstyle g_{01}} \\ E_f & \xrightarrow{\gamma_{11}} & E_g \\ {\scriptstyle f_{12}}\downarrow & & \downarrow{\scriptstyle g_{12}} \\ Y & \xrightarrow[\gamma_{22}]{} & B \end{array}.$$

We denote the canonical projection $\mathcal{C}^{\mathbf{3}} \longrightarrow \mathcal{C}^{\mathbf{2}}$ by

$$d_1 := \pi_1^{\mathcal{C}^{\mathbf{3}}} : \mathcal{C}^{\mathbf{3}} \longrightarrow \mathcal{C}^{\mathbf{2}}.$$

We define two more 'face maps' d_0, d_2 (practically 'forgetting' about the object in the subscript) as

$$d_0 : \left(X_0 \xrightarrow{f_{01}} X_1 \xrightarrow{f_{12}} X_2\right) \mapsto X_1 \xrightarrow{f_{12}} X_2$$
$$d_2 : \left(X_0 \xrightarrow{f_{01}} X_1 \xrightarrow{f_{12}} X_2\right) \mapsto X_0 \xrightarrow{f_{01}} X_1.$$

One can show the following, similar to the arrow category.

Lemma 3.16 (`three_colims`)**.** *Suppose $\mathcal{C}$ is a cocomplete category, then the three category $\mathcal{C}^{\mathbf{3}}$ is cocomplete.*

Chapter 4

Model Categories

As briefly motivated in the introduction, model categories provide a language and tools to study homotopy theory in a more global and generic way than the traditional study of topological spaces. A model structure on a category $\mathcal{C}$ consists of three related classes of morphisms $(\mathcal{W}, \mathcal{K}, \mathcal{F})$, called the *weak equivalences*, the *cofibrations* and the *fibrations*. These classes of maps are related through two *weak factorization systems*. Weak factorization systems are in turn comprised of two interacting classes of maps: $\mathcal{L}$ and $\mathcal{R}$. These two classes are related through a dual lifting property, such that any map in the category can be factored as a map in $\mathcal{L}$ and a map in $\mathcal{R}$.

These weak factorization systems (WFS) are called *weak*, because there is no additional structure on them. Neither the lifts nor factorizations need to have any structure in any way, nor does the factorization have to be unique. This leads to some issues from a categorical point of view. Grandis and Tholen introduced the notion of *natural weak factorization system* to resolve these issues [8]. In this chapter, we figure out that there are some additional constructive issues. It turns out that some of the properties one wishes a WFS to have, depend on the axiom of choice. Natural weak factorization systems will satisfy analogues to these properties, without the need for the axiom of choice.

This chapter will build up some definitions, needed to define the notion of a WFS and of a model category. We go over some of the most important properties of WFSs, and provide important examples. We give some new insight into constructive issues that arise when trying to show some of their desired properties, while motivating that these issues cannot be circumvented by changing the way one defines a WFS. We then introduce the notion of a model category, and give a basic introduction to the *homotopy category* construction, motivating the usefulness of model category theory.

In this chapter, and in the rest of this thesis, we will always assume that the category $\mathcal{C}$ we are working with is a category, not a precategory or a univalent category. This means that the only assumption we make on the category is that the type of morphisms between any pair of objects is a set.

First of all, let us formally define the notion of a *morphism class*.

Definition 4.1 (`morphism_class`)**.** *A class of morphism is a collection of subsets of* $(X \to Y)$ *for all objects* $X, Y : \mathcal{C}$*. We write* $f \in \mathcal{S}$ *to denote that a morphism* $f : X \to Y$ *is contained in a morphism class* $\mathcal{S}$*.*

```
Definition morphism_class (C : category) : UU :=
  ∏ (X Y : C), hsubtype (X --> Y).
```

Here, `hsubtype` is defined as

$$\texttt{hsubtype} := \prod_{\texttt{X : UU}} \texttt{X -> hProp}.$$

The term `hsubtype (X --> Y)` indicates for each morphism of type `X --> Y` whether it is or is not included in the morphism class.

Example 4.2. *A basic example is the morphism class containing all morphisms of $\mathcal{C}$, we will call this* `univ`*. Another example is the morphism class containing all isomorphisms of $\mathcal{C}$, we will call this* `isos`*.*

4.1 Preliminaries

We will need some preliminary constructions on morphism classes before we continue on to weak factorization systems. First of all, let us define retracts.

4.1.1 Retracts

Definition 4.3 (`retract`, `morphism_class_retract_closed`)**.** *A class $\mathcal{S}$ of maps in a category $\mathcal{C}$ is closed under retracts, if given a commutative diagram*

$$\begin{array}{ccccc} X' & \xrightarrow{i_X} & X & \xrightarrow{r_X} & X' \\ \downarrow{\scriptstyle f'} & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle f'} \\ Y' & \xrightarrow{i_Y} & Y & \xrightarrow{r_Y} & Y' \end{array} \qquad \begin{array}{l} i_X \cdot r_X = \mathrm{id} \\ \\ i_Y \cdot r_Y = \mathrm{id} \end{array}$$

with $f \in \mathcal{S}$, it follows that $f' \in \mathcal{S}$. We say that f' is a retract *of f.*

```
Definition is_retract {x y x' y' : C} (f : x --> y) (f' : x' --> y')
    (ix : x' --> x) (rx : x --> x') (iy : y' --> y) (ry : y --> y') : UU :=
  (ix · rx = identity x') × (iy · ry = identity y')
  × (ix · f = f' · iy) × (rx · f' = f · ry).

Definition retract {x y x' y' : C} (f : x --> y) (f' : x' --> y') : UU :=
  ∑ (ix : x' --> x) (rx : x --> x') (iy : y' --> y) (ry : y --> y'),
    is_retract f f' ix rx iy ry.

Definition morphism_class_retract_closed
    {C : category} (S : morphism_class C) : UU :=
  ∏ x y (f : x --> y) x' y' (f' : x' --> y'),
    (S _ _ f') × (retract f' f) -> (S _ _ f).
```

It may be useful to think of the diagram as a composition of arrows in the arrow category $\mathcal{C}^{\mathbf{2}}$, where the diagram simply says that $(i_X, i_Y) \cdot (r_X, r_Y) = \mathrm{id}_{f'}$ in the arrow category $\mathcal{C}^{\mathbf{2}}$.

Given a morphism class, we define its *closure under retracts.*

Definition 4.4 (`morphism_class_retract_closure`)**.** *Given a morphism class $\mathcal{S}$ of maps in $\mathcal{C}$, we define its* closure under retracts *as the class of all morphisms f' in $\mathcal{C}$ such that there exists an $f \in \mathcal{S}$ such that f' is a retract of f. We denote this class by $\mathcal{S}^{cl}$.*

```
Definition morphism_class_retract_closure
    {C : category} (S : morphism_class C) : morphism_class C :=
  λ x y (f : x --> y), ∃ x' y' (f' : x' --> y'), (S _ _ f') × (retract f' f).
```

In fact, an alternative definition to a morphism class being closed under retracts is simply that it is equal to its closure under retracts.

Lemma 4.5 (`morphism_class_retract_closed_iff_eq_cl`)**.** *A morphism class $\mathcal{S}$ is closed under retracts if and only if it is equal to its retract closure.*

Retracts have some basic properties, without elaborating on them too much, we list a few. A retract of an isomorphism is again an isomorphism (`retract_is_iso`). Retracts are preserved under the action of functors (`functor_on_retract`). A retract from a morphism f' to f yields a retract from f'^{opp} to f^{opp} in the opposite category (`opp_retract`). Finally, for any morphism f the type `retract`f f is inhabited at least by a diagram where the horizontal morphisms are all identity (`retract_self`).

4.1.2 Lifting Problems

Weak factorization systems describe how two morphism classes interact with each other, in terms of a dual lifting property, as well as a factorization on the whole category. To define this dual lifting property, we first introduce the notion of *lifting problems*.

Definition 4.6. *A lifting problem in a category $\mathcal{C}$ is a morphism in the arrow category $\mathcal{C}^{\mathbf{2}}$. More precisely, for two morphisms f, g in $\mathcal{C}$, an* (f, g)-lifting problem *is a diagram*

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

We can also denote a lifting problem as the corresponding morphism $(h, k) : f \to g$ in the arrow category $\mathcal{C}^{\mathbf{2}}$.

We would like these lifting 'problems' to have 'solutions', which we call *fillers*.

Definition 4.7 (`filler`)**.** *Given morphisms f, g in $\mathcal{C}$, an (f, g)-lifting problem has a filler l if the dashed arrow exists.*

$$\begin{array}{ccc} X & \xrightarrow{h} & A \\ {\scriptstyle f}\downarrow & \overset{l}{\nearrow} & \downarrow{\scriptstyle g} \\ Y & \xrightarrow[k]{} & B \end{array}$$

```
Definition filler {x y a b : C} {f : x --> y} {g : a --> b}
    {h : x --> a} {k : y --> b} (H : h · g = f · k) :=
  ∑ l : y --> a, (f · l = h) × (l · g = k).
```

The morphism classes in a WFS are related to each other through the *lifting property*.

Definition 4.8 (`lp`, `elp`)**.** *Given morphisms f, g in $\mathcal{C}$, we say that that (f, g) has the* lifting property *if each (f, g)-lifing problem has a (not necessarily unique) filler.*

```
Definition lp {x y a b : C} (f : x --> y) (g : a --> b) : hProp :=
  ∀ (h : x --> a) (k : y --> b) (H : h · g = f · k), ‖filler H‖.
```

Now, we use the truncation here because we want to make sure that `lp` is actually a property, as we want to use it to define morphism classes. We could also have defined it without, in which case it is no longer a property. Even with the truncation, we are able to prove lemmas using the universal property of the propositional truncation, Lemma 2.14. However, defining the lifting property without truncation allows us to actually 'access' the filler for every lifting problem in *any* context, not just when we are trying to prove something proprositional. This is something that will be useful later on. We define it as the *existential lifting property*.

```
Definition elp {x y a b : C} (f : x --> y) (g : a --> b) : UU :=
  ∏ (h : x --> a) (k : y --> b) (H : h · g = f · k), filler H.
```

This may seem like a subtle difference, but it makes a big difference in formalizing the theory. This will be made clear later on in this chapter.

An interesting and useful fact is that retracts preserve the lifting property in the following way.

Lemma 4.9 (`elp_of_retracts`, `lp_of_retracts`)**.** *Let f and g be two morphisms in $\mathcal{C}$ and let f' be a retract of f and g' a retract of g. If (f, g) has the (existential) lifting property, then so does (f', g').*

One can show this for both the lifting property *and* the *existential* lifting property, simply by pasting the retract diagram next to any lifting problem. The lifting property allows us to define certain constructions on morphism classes, defining the relation between the morphism classes in a WFS.

Definition 4.10 (`rlp`)**.** *For a class of maps $\mathcal{L}$, we say that g satisfies the right lifting property (RLP) with respect to $\mathcal{L}$ if (f, g) has the lifting property for all $f \in \mathcal{L}$. We let $\mathcal{L}^{\square}$ denote the class of all such maps g.*

```
Definition rlp (L : morphism_class C) : (morphism_class C) :=
    λ {a b : C} (g : a --> b), ∀ (x y : C) (f : x --> y), ((L _ _) f ⇒ lp f g).
```

Dually, we define the left lifting property.

Definition 4.11 (`llp`)**.** *For a class of maps $\mathcal{R}$, we say that f satisfies the left lifting property (LLP) with respect to $\mathcal{R}$ if (f, g) has the lifting property for all $g \in \mathcal{R}$. We let ${}^{\square}\mathcal{R}$ denote the class of all such maps f.*

```
Definition llp (R : morphism_class C) : (morphism_class C) :=
    λ {x y : C} (f : x --> y), ∀ (a b : C) (g : a --> b), ((R _ _) g ⇒ lp f g).
```

4.2 Weak Factorization Systems

We are now equipped to define weak factorization systems. We first define the factorization axiom that describes one of the ways the two classes in a weak factorization system interact.

Definition 4.12 (`wfs_fact_ax`)**.** *We say that a pair of morphism classes $(\mathcal{L}, \mathcal{R})$* factors *$\mathcal{C}$ if every morphism $f : X \to Y$ factors as a composite*

$$X \xrightarrow{\lambda} E_f \xrightarrow{\rho} Y$$

with $E_f : \mathcal{C}$, $\lambda \in \mathcal{L}$ and $\rho \in \mathcal{R}$.

```
Definition wfs_fact_ax {C : category} (L R : morphism_class C) :=
  ∏ x y (f : x --> y),
    ∃ ef (l : x --> ef) (r : ef --> y),
      (L _ _) l × (R _ _) r × l · r = f.
```

Definition 4.13 (`wfs`)**.** *A weak factorization system (WFS), is an ordered pair $(\mathcal{L}, \mathcal{R})$ of classes of maps in $\mathcal{C}$ that factors $\mathcal{C}$ and satisfies*

$$\mathcal{L} = {}^{\square}\mathcal{R} \quad \textit{and} \quad \mathcal{R} = \mathcal{L}^{\square}.$$

```
Definition is_wfs {C : category} (L R : morphism_class C) :=
  (L = llp R) × (R = rlp L) × (wfs_fact_ax L R).

Definition wfs (C : category) :=
  ∑ (L R : morphism_class C), is_wfs L R.
```

Remark. *This definition of WFS shows why we cannot use the* existential *lifting property instead, as the equalities $\mathcal{L} = {}^{\square}\mathcal{R}$ and $\mathcal{R} = \mathcal{L}^{\square}$ would be ill-typed. A corresponding notion of ${}^{\square}\mathcal{R}$ or $\mathcal{L}^{\square}$ would not simply be a morphism class, but rather a morphism class* with extra data*, containing information about the lifts in any appropriate lifting problem.*

Example 4.14 (`wfs_isos_univ`)**.** *The morphism class consisting of all isomorphisms* `isos`*, and the morphism class containing all morphisms in $\mathcal{C}$,* `univ`*, form a weak factorization system* (`isos`, `univ`)*. The choice for the factorization is obvious. Dually,* (`univ`, `isos`) *also forms a WFS.*

Example 4.15. *One can define the notion of* strong factorization system *as a WFS for which the lifts between $\mathcal{L}$- and $\mathcal{R}$-maps are unique. Any strong factorization system is a weak factorization system.*

Example 4.16. *The pair (Complemented Mono, Split Epi) forms a WFS on* **SET** *[9], factoring a morphism $f : X \to Y$ as*

$$X \xrightarrow{\mathtt{in}^{\sqcup}_X} X \sqcup Y \xrightarrow{f \sqcup \mathrm{id}_Y} Y.$$

Example 4.17. *The more general (Mono, Epi) is a WFS on* **SET** *if and only if the axiom of choice holds [16,* `Weak Factorization System on Set`*].*

Example 4.18. *Perhaps the most important examples of weak factorization systems are those on* **TOP**. *Let $\mathcal{W}$ denote the class of weak homotopy equivalences in* **TOP**. *Let $\mathcal{F}$ denote the class of Serre fibrations, and $\mathcal{K}$ the class of retracts of relative CW complexes. In other words, $\mathcal{K}$ consists of maps $r : Y \to X$ where Y is a relative CW complex on X, such that $i \cdot r = \mathrm{id}_X$ for $i : X \to Y$ the canonical inclusion. Then both $(\mathcal{W} \cap \mathcal{F}, \mathcal{K})$ and $(\mathcal{F}, \mathcal{W} \cap \mathcal{K})$ form weak factorization systems on* **TOP** *[2].*

Example 4.19. *Again in* **TOP**, *let $\mathcal{W}$ denote the class of homotopy equivalences, and let $\mathcal{F}$ now denote the class of Hurewicz fibrations. Let $\mathcal{K} = (\mathcal{W} \cap \mathcal{F})^{\square}$, the* closed Hurewicz cofibrations. *Then the pairs $(\mathcal{W} \cap \mathcal{F}, \mathcal{K})$ and $(\mathcal{F}, \mathcal{W} \cap \mathcal{K})$ form weak factorization systems [20]. In these examples, the lifting property corresponds with the Homotopy Lifting Property and the Homotopy Extension Property in topology.*

Example 4.20. *Other important examples of weak factorization systems are those on* **SSET**. *Let $\mathcal{W}$ again denote the class of weak homotopy equivalences, now in* **SSET**. *Let $\mathcal{F}$ denote the class of* Kan fibrations, *and $\mathcal{K}$ the class of monomorphisms $f : X \to Y$ which are levelwise injections, i.e. $f_n : X_n \to Y_n$ is injective, then the pairs $(\mathcal{W} \cap \mathcal{F}, \mathcal{K})$ and $(\mathcal{F}, \mathcal{W} \cap \mathcal{K})$ form weak factorization systems [2].*

One important fact about WFSs is that one does not have to show that a morphism f has the right lifting property with respect to any $\mathcal{R}$-map in order to show that it is an $\mathcal{L}$-map. It is in fact equivalent to show that (f, ρ) has the lifting property with respect to some given $\mathcal{R}$-map ρ that factors f. This is a property that shows some parallels between WFSs and NWFSs which we define in the next chapter. Dually, the following holds.

Lemma 4.21 (`llp_iff_lift_with_R`)**.** *Let $(\mathcal{L}, \mathcal{R})$ be a WFS on $\mathcal{C}$ and let f be a morphism in $\mathcal{C}$. Let λ be an $\mathcal{L}$-map and ρ be an $\mathcal{R}$-map such that $f = \lambda \cdot \rho$. Then f is an $\mathcal{R}$-map if and only if (λ, f) has the lifting property.*

Proof. The forward direction is obvious. For the converse, let g be any $\mathcal{L}$-map, and consider a lifting problem

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

using the factorization of f, we rewrite this as

$$\begin{array}{ccccc} A & \xrightarrow{h} & X & \xrightarrow{\lambda} & E_f \\ {\scriptstyle g}\downarrow & \overset{l_{g,\rho}}{\nearrow} & & & \downarrow{\scriptstyle \rho} \\ B & & \xrightarrow[k]{} & & Y \end{array}$$

where we find a lift $l_{g,\rho}$ since g is an $\mathcal{L}$-map and ρ an $\mathcal{R}$-map. By assumption, there is a lift

$$\begin{array}{ccc} X & = & X \\ {\scriptstyle\lambda}\downarrow & \overset{l_{\lambda,f}}{\nearrow} & \downarrow{\scriptstyle f} \\ E_f & \xrightarrow[\rho]{} & Y \end{array}$$

The composite $l_{g,\rho} \cdot l_{\lambda,f}$ is a lift for the (g, f) lifting problem. □

We go over some more interesting properties of WFSs. They are summarized in the notion of *left- or right saturated* morphism classes. We claim that the class $\mathcal{L}$ of left maps is *left saturated* and the class $\mathcal{R}$ of right maps is *right saturated* [3, Prop 14.1.8].

Definition 4.22. *Let $\mathcal{L}$ be a class of maps in a category $\mathcal{C}$. We say that $\mathcal{L}$ is left saturated if the following closure properties hold.*

(i) *$\mathcal{L}$ contains all isomorphisms of $\mathcal{C}$.*

(ii) *$\mathcal{L}$ is closed under retracts.*

(iii) *Any pushout of a map in $\mathcal{L}$ is in $\mathcal{L}$. That is, if the following diagram is a pushout and f is in $\mathcal{L}$, then so is p_1.*

$$\begin{array}{ccc} X & \longrightarrow & Z \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle p_1} \\ Y & \longrightarrow & P \end{array}$$

(iv) *Any coproduct of maps in $\mathcal{L}$ is in $\mathcal{L}$.*

(v) *$\mathcal{L}$ is closed under transfinite composition. In other words, if we have a chain*

$$X_0 \to X_1 \to X_2 \dots$$

for which every $X_v \to X_{v+1}$ is in $\mathcal{L}$, then the composite $X_0 \to \operatorname{colim} X_v$ is in $\mathcal{L}$.

The notion of a right saturated class of morphisms in $\mathcal{C}$ is dual. In this case, pushouts and coproducts should be replaced with pullbacks and products.

Remark. *A transfinite composition is actually defined on* any *limit ordinal, not just for the first limit ordinal ω. In our definition of left saturated, we choose to only ask for the transfinite composition of chains, so diagrams corresponding with the first limit ordinal ω. We do this because, even though some work has been done very recently [21], the theory of ordinals has not been developed far enough in HoTT and* `UniMath` *to be able to formalize a more general definition. In the rest of this thesis, we will also consider transfinite composites to mean transfinite composites for ω, for precisely this reason.*

Showing these properties for a WFS may not be very hard on paper, but requires some work in `UniMath`. We briefly go over the statements and possibly some proofs and issues we run into with the way WFSs are defined.

To show that $\mathcal{L}$ (respectively $\mathcal{R}$) contains all isomorphisms, it is useful to note the following.

Lemma 4.23 (`llp_anti`)**.** *For two classes $\mathcal{J} \subseteq \mathcal{S}$ in $\mathcal{C}$, we have that $\mathcal{S}^{\square} \subseteq \mathcal{J}^{\square}$. Similarly, we also have ${}^{\square}\mathcal{S} \subseteq {}^{\square}\mathcal{J}$.*

Proof. The proof is simple, simply note that if $f \in {}^{\square}\mathcal{S}$, then f has the left lifting property with maps in $\mathcal{S}$, and since $\mathcal{J} \subseteq \mathcal{S}$, it has the left lifting property with all maps in $\mathcal{J}$. □

The previous lemma, combined with the example of the (`isos`, `univ`) WFS, pretty much directly imply the following lemma.

Lemma 4.24 (`wfs_L_contains_isos`)**.** *Given a WFS* $(\mathcal{L}, \mathcal{R})$*,* $\mathtt{isos} \subseteq \mathcal{L}$*, and dually,* $\mathtt{isos} \subseteq \mathcal{R}$ *as well.*

Proof. This follows simply from the fact that

$$\mathcal{R} \subseteq \mathtt{univ}$$

and so

$$\mathtt{isos} = {}^{\square}\mathtt{univ} \subseteq {}^{\square}\mathcal{R} = \mathcal{L}.$$

and dually for $\mathcal{R}$. □

Lemma 4.9 directly implies the next point.

Lemma 4.25 (`wfs_L_retract`)**.** *A WFS* $(\mathcal{L}, \mathcal{R})$ *is closed under retracts, meaning that if* $f \in \mathcal{L}$*, and* f' *is a retract of* f*, then* $f' \in \mathcal{L}$ *as well, and similar for* $\mathcal{R}$*.*

Next, we want to show that a WFS is closed under pushouts. Instead, we show that a WFS is closed under pullbacks, and since a WFS gives a canonical WFS in the opposite category, it follows that dually, a WFS is closed under pushouts. This is where the power of the Coq proof assistant really becomes clear: the proof checker accepted

```
apply (wfs_closed_pullbacks (opp_wfs _))
```

as a complete proof for showing closedness under pushouts.

Lemma 4.26 (`wfs_closed_pullbacks`, `wfs_closed_pushouts`)**.** *A WFS* $(\mathcal{L}, \mathcal{R})$ *is closed under pullbacks, meaning that if we have a pullback diagram*

$$\begin{array}{ccc} P & \xrightarrow{p_1} & X \\ {\scriptstyle p_2}\downarrow & & \downarrow{\scriptstyle f} \\ Z & \xrightarrow[p]{} & Y \end{array}$$

where f *is in* $\mathcal{R}$*, then* p_2 *is in* $\mathcal{R}$*.*

Proof. Given a lifting problem

$$\begin{array}{ccc} A & \longrightarrow & P \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle p_2} \\ B & \longrightarrow & Z \end{array}$$

with $g \in \mathcal{L}$, we paste the pullback diagram onto the above diagram to obtain the following.

$$\begin{array}{ccccc} A & \longrightarrow & P & \longrightarrow & X \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle p_2} & & \downarrow{\scriptstyle f} \\ B & \longrightarrow & Z & \longrightarrow & Y \end{array}$$

This yields a lift $B \to X$. The universal property of the pullback then gives a morphism $B \to P$, which forms a lift in the (g, p_2) lifting problem. □

Then finally, we want to show that a WFS is closed under coproducts and transfinite composition. These points are where we run into issues with the way we need to define the lifting property. The propositional truncation makes it so we are not able to reason about all lifts for a whole range of lifting problems at once, forcing us to use the axiom of choice to show these two properties.

Lemma 4.27 (`wfs_closed_coproducts`)**.** *A WFS $(\mathcal{L}, \mathcal{R})$ is closed under coproducts, meaning that for any two families of objects $\{ X_i \}_{i:I}, \{ Y_i \}_{i:I}$ with some type I and any family of maps $\{ f_i \}_{i:I}$ such that $f_i \in \mathcal{L}$ for all $i : I$, the coproduct $f : \bigsqcup_{i:I} X_i \to \bigsqcup_{i:I} Y_i$ of the f_i is also in $\mathcal{L}$.*

Proof. Firstly, consider a lifting problem

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

with $g \in \mathcal{R}$. Using the properties of the coproduct, this gives diagrams for the individual f_i, where we obtain the existence of a lift l_i through the lifting properties of the individual f_i.

$$\begin{array}{ccc} X_i & \xrightarrow{h} & A \\ {\scriptstyle f_i}\downarrow & \overset{l_i}{\nearrow} & \downarrow{\scriptstyle g} \\ Y_i & \xrightarrow[k]{} & B \end{array}$$

Now this is where the propositional truncation of the lifting property becomes readily apparent. *Using the axiom of choice*, we obtain a morphism

$$\bigsqcup l_i : \bigsqcup Y_i \to A$$

such that the diagram of the coproducts commutes [16, `Weak Factorization System`]. □

Lemma 4.28 (`wfs_closed_transfinite_composition`)**.** *Suppose $(\mathcal{L}, \mathcal{R})$ is a WFS and let $d := \{ X_v \}_{v:\mathbb{N}}$ be a chain in $\mathcal{C}$, for which every*

$$X_v \to X_{v+1}$$

is in $\mathcal{L}$, then the canonical map

$$\mathtt{in}_0^{\rightarrow} : X_0 \hookrightarrow \operatorname{colim} X_v$$

is in $\mathcal{L}$.

Proof. Suppose $g \in \mathcal{R}$ for some $g : A \to B$, and consider a lifting problem

$$\begin{array}{ccc} X_0 & \xrightarrow{h} & A \\ {\scriptstyle \mathtt{in}_0^{\rightarrow}}\downarrow & & \downarrow{\scriptstyle g} \\ \operatorname{colim} X_v & \xrightarrow[k]{} & B \end{array}$$

The idea is to inductively define maps $X_v \to A$ using the lifting property. For X_0, the map $X_0 \to A$ is clear: the map h. Then suppose we have a map $h_v : X_v \to A$. We get a lifting problem

$$\begin{array}{ccccc} X_v & & \xrightarrow{\qquad h_v \qquad} & & A \\ \downarrow & & & & \downarrow{\scriptstyle g} \\ X_{v+1} & \xrightarrow{\mathtt{in}_{v+1}^{\rightarrow}} & \operatorname{colim} X_v & \xrightarrow[k]{} & B \end{array}$$

where the left map is in $\mathcal{L}$ and the right map is in $\mathcal{R}$. This gives us a filler $h_{v+1} : X_{v+1} \to A$. *Using the axiom of choice*, this allows us to construct a cocone on d with vertex A, giving us the morphism $\operatorname{colim} X_v \to A$ we need to fill our original lifting problem. □

The Axiom of Choice in these proofs

As emphasized in the previous two proofs, we need the axiom of choice to finish the argument. The reason for this is that we know the mere existence of a lift in individual diagrams, indexed by an indexing type I, but we need to put all the lifts together to get the mere existence of a lift in a 'combined' diagram. For example, in the proof of the closedness under coproducts, we have a hypothesis of type

$$\prod_{i:I} \left\| \sum_{l_i : Y_i \to A} f_i \cdot l_i = h \times l_i \cdot g = k \right\|$$

telling us that we know of the *mere existence* of a lift for every individual diagram. We want to show that there is a 'combined lift', i.e. a term of type

$$\left\| \sum_{l : \prod_{i:I} Y_i \to A} \prod_{i:I} f_i \cdot l(i) = h \times l(i) \cdot g \right\|,$$

asking for the mere existence of a lift in the 'total diagram'. This is precisely the statement of the axiom of choice in HoTT, which says that for any set X and any family of types $L : X \to \mathcal{U}$ such that $L(x)$ is a set for all $x : X$, and any family of predicates $P : \prod_{x:X} L(x) \to \mathtt{hProp}$, we have [14]

$$\left(\prod_{x:X} \left\| \sum_{l_x : L(x)} P(x, l_x) \right\| \right) \to \left\| \sum_{l : \prod_{x:X} L(x)} \prod_{x:X} P(x, l(x)) \right\|.$$

In our example, X was the indexing set I, L mapped the lifting problem corresponding to index $i : I$ to the lift $l_i : Y_i \to A$ and P was the proposition that the i-th diagram commutes with the given lift. Indeed, this is a mere proposition, since we assumed our category to be a category (i.e. where all homsets are small) and not a precategory. We *would* be able to show that a WFS is closed under coproducts assuming the type theoretical version of the axiom of choice, or at the very least assuming that our indexing type I is a 'choice base', which effectively assumes the axiom of choice only for our indexing type I.

One might think that we could perhaps change our definition of a WFS to use the *existential* lifting property, which arguably would fix the problem. The issue is that the existential lifting property is actually not propositional, so the axiom

$$\mathcal{L} = {}^{\square}\mathcal{R}$$

in a WFS would not make sense anymore. After all, in this equality, $\mathcal{L}$ is a morphism class, whereas ${}^{\square}\mathcal{R}$ is now a morphism class *with extra data*. The term ${}^{\square}\mathcal{R}$ now holds data about all lifts for any lifting problem with a right map. This is no longer 'just a morphism class'. There would be no nice way of defining such a thing in a sensible way, unless we simply use a morphism class and pass along a choice function that gives a lift for any lifting problem with an $\mathcal{R}$-map. This would not accomplish much though, since it would only move the 'issue' of the axiom of choice into the assumptions of the lemma anyway, and we might as well require our indexing type I to be a choice base.

4.3 Model Categories

With the definition of WFSs, we can now define the notion of a model structure. As mentioned before, a model structure describes the behavior of three related classes of morphisms $(\mathcal{W}, \mathcal{K}, \mathcal{F})$, called the *weak equivalences*, the *cofibrations* and the *fibrations*. As the names

suggest, these classes correspond with their namesakes in the category of topological spaces. After defining what a model structure is, we give some important examples including some on the category of topological spaces. We then give a brief introduction to the homotopy category, which is perhaps the most important tool arising from model category theory.

Definition 4.29 (`is_model_category`)**.** *A model structure on $\mathcal{C}$ consists of classes $(\mathcal{W}, \mathcal{K}, \mathcal{F})$ of morphisms of $\mathcal{C}$, where $\mathcal{W}$ are the* weak equivalences*, $\mathcal{K}$ the* cofibrations *and $\mathcal{F}$ the* fibrations*, such that*

(i) $\mathcal{W}$ has the two out of three property.
(ii) $(\mathcal{K}, \mathcal{F} \cap \mathcal{W})$ is a WFS.
(iii) $(\mathcal{K} \cap \mathcal{W}, \mathcal{F})$ is a WFS.

We split the definition for the two out of three property into 3 separate axioms, giving us a type

```
Definition weq_comp_ax {C : category} (W : morphism_class C) : UU :=
    ∀ x y z (f : x --> y) (g : y --> z),
      (W _ _) f ⇒ (W _ _) g ⇒ (W _ _) (f · g).

Definition weq_cancel_left_ax {C : category} (W : morphism_class C) : UU :=
    ∀ (x y z : C) (f : x --> y) (g : y --> z),
      (W _ _) f ⇒ (W _ _) (f · g) ⇒ (W _ _) g.

Definition weq_cancel_right_ax {C : category} (W : morphism_class C) : UU :=
    ∀ (x y z : C) (f : x --> y) (g : y --> z),
      (W _ _) g ⇒ (W _ _) (f · g) ⇒ (W _ _) f.

Definition is_weak_equivalences {C : category} (W : morphism_class C) : UU :=
    weq_comp_ax W × weq_cancel_left_ax W × weq_cancel_right_ax W.

Definition weak_equivalences (C : category) : UU :=
    ∑ (W : morphism_class C), is_weak_equivalences W.
```

The type theoretic definition of a model structure simply becomes

```
Definition is_model_category {C : category} (W K F : morphism_class C) :=
    is_weak_equivalences W × is_wfs K (F ∩ W) × is_wfs (K ∩ W) F.
```

Example 4.30. *Every category $\mathcal{C}$ that is both complete and cocomplete admits the* trivial model structure*, where the weak equivalences are the class* `isos` *of isomorphisms and both the fibrations and the cofibrations are the class* `univ` *of all maps [16,* ***`trivial model structure`****].*

Example 4.31. *Perhaps the most well-known example of a model structure is on the category* **TOP** *of topological spaces, where the fibrations are the Serre fibrations, the cofibrations are retracts of relative CW-complexes and weak equivalences are the* weak homotopy equivalences *[2]. This model structure is referred to as the* classical model structure on **TOP***, and was hinted at in previous examples.*

Example 4.32. *The other examples of WFSs on* **TOP** *also form a model structure: with homotopy equivalences as weak equivalences, Hurewicz fibrations as fibrations and the* closed Hurewicz cofibrations *as cofibrations. This model structure is sometimes referred to as the* Hurewicz *or* Strøm model structure *[20].*

Example 4.33. *In the category of* **SSET***, there is a model structure with weak homotopy equivalences as weak equivalences, the class of* Kan fibrations *as fibrations, and with cofibrations the class of monomorphisms $f : X \to Y$ which are levelwise injections [2]. This model structure is known as the* Quillen model structure *on* **SSET***, and can in fact be used to present type theory itself [15].*

Example 4.34. *A very modern example where model structures are used is in the study of* $(\infty, 1)$-categories*. These can be seen as a more modern generalization of model categories. Regardless of the way* $(\infty, 1)$*-categories are represented, there is always a model structure on them [7].*

There are many more examples of model structures, such as model structures on R-Mod, the category of R-Modules for a Frobenius ring R [2], or model structures on the category of R chain complexes, where R is any ring [3].

To finish off this chapter, we give a very brief introduction to the construction of the *homotopy category* from a model structure. This category allows one to work with homotopy invariant concepts, and use certain important homotopy theoretical results and constructions in a more generic and global setting, serving as an important tool in homotopy theory.

4.3.1 The Homotopy Category

Perhaps the most important property of model categories is that they allow one to define 'well-behaved homotopy categories'. The idea behind homotopy categories is that the weak equivalences in the category become actual isomorphisms, as well as being able to consider maps *up to homotopy*. This is something we commonly want in topology for example, as many constructions are preserved by weak equivalences and homotopies.

For a brief introduction of homotopy categories of model categories, we follow [3]. For this section, we assume that $\mathcal{C}$ is a complete and cocomplete model category with $(\mathcal{W}, \mathcal{K}, \mathcal{F})$ the classes of weak equivalences, cofibrations and fibrations. We denote the initial object of $\mathcal{C}$ by $\emptyset$ and the terminal object by $*$.

Firstly, we define morphisms in $\mathcal{W} \cap \mathcal{F}$ to be *acyclic fibrations*, and morphisms in $\mathcal{W} \cap \mathcal{K}$ to be *acyclic cofibrations*. We introduce the notion of (co)fibrant objects.

Definition 4.35. *Suppose* $X : \mathcal{C}$*. Then* X *is* cofibrant*, if and only if the unique morphism* $\emptyset \to X$ *form the is a cofibration. Any morphism* $q : QX \to X$ *which is an acyclic fibration where* QX *is cofibrant is called a* cofibrant replacement of X*. Dually, we define* X *to be* fibrant *if and only if the unique morphism* $X \to *$ *is a fibration, and any morphism* $r : X \to RX$ *is called a* fibrant replacement of X *if and only if* r *is an acyclic cofibration and* RX *is fibrant.* X *is* bifibrant *if it is both fibrant and cofibrant.*

Remark. *Note that we can always obtain (co)fibrant replacements for* X *by factoring either* $(\emptyset \to X)$ *or* $X \to *$ *with the appropriate WFS in the model structure.*

Remark. *If the factorization is* functorial*, which we will define in the next chapter, there is a canonical (functorial) choice for the (co)fibrant replacements* (QX) *and* RX *of an object* $X : \mathcal{C}$*. We in fact find two bifibrant replacements:* QRX *and* RQX*.*

We can define general notions of cylinders and path objects in a model category, similar to cylinders and paths in **TOP**.

Definition 4.36. *A cylinder object* $\operatorname{Cyl} X$ *for an object* $X : \mathcal{C}$ *is an object together with maps* $i_0, i_1 : X \to \operatorname{Cyl} X$ *and* $p : \operatorname{Cyl} X \to X$ *such that* $i_0 \cdot p = \mathrm{id}_X = i_1 \cdot p$ *and* p *is a weak equivalence. By the two out of three property, then* i_0 *and* i_1 *are also weak equivalences.*

Example 4.37. *Indeed, in* **TOP** *with the Hurewicz model structure, any* $X \times [0, 1]$ *is a cylinder object.*

Cylinder objects allow us to define a general notion of (left) homotopy:

Definition 4.38. *A* left homotopy *between maps* $f, g : X \to Y$ *is a map* $h : \operatorname{Cyl} X \to Y$ *such that* $i_0 \cdot h = f$ *and* $i_1 \cdot h = g$*. We say that* f *and* g *are* left homotopic*.*

Indeed, this notion of left homotopy corresponds with that in **TOP** with the Hurewicz model structure. Path objects and right homotopy are a dual notion. One can show that a map is a weak equivalence if it is either left or right homotopic to a weak equivalence.

Definition 4.39. *Let X be cofibrant and Y be fibrant. Then two maps $f, g : X \to Y$ are said to be* homotopic *if and only if they are left or right homotopic. In this case, left and right homotopic is equivalent. We denote $\pi(X, Y)$ for the set of homotopy classes of maps $X \to Y$.*

The model structure allows us to define the following notion of the homotopy category.

Definition 4.40. *The homotopy category* $\mathrm{Ho}\,\mathcal{C}$ *of a category $\mathcal{C}$ is a category with the same objects as $\mathcal{C}$, with morphism sets* $\mathrm{Hom}(X, Y) := \pi(RQX, RQY)$.

It turns out that the isomorphisms in this category are precisely the image of the weak equivalences $\mathcal{W}$ under the RQ functor. It is also true that the homotopy category constructed on a model category is in fact independent of the choice of (co)fibrant replacements, and even independent even of the classes of fibrations and cofibrations [7].

This construction yields various interesting results from homotopy theory, but in a more general context. One example is the well-known Whitehead theorem.

Lemma 4.41 (Whitehead)**.** *The following dual versions of the Whitehead theorem hold:*

1. *A map $p : Z \to Y$ between fibrant objects is a weak equivalence if and only if $p_* : \pi(X, Z) \to \pi(X, Y)$ is a bijection for all cofibrant objects X.*
2. *A map $i : W \to X$ between cofibrant objects is a weak equivalence if and only if $i^* : \pi(X, Y) \to \pi(W, Y)$ is a bijection for all fibrant objects Y.*

The homotopy category obtained from a model structure also provides the necessary tools to study homotopy (co)limits.

The general conclusion of this theory is that model categories indeed provide a more general and global way of doing homotopy theory. Instead of working locally, perhaps considering only a single space, or a specific set of spaces, they allow one to study the homotopy theoretical properties in a more global setting, considering the whole category.

Chapter 5

Natural Weak Factorization Systems

Natural weak factorization systems (NWFSs) are a sort of 'algebraic refinement' of WFSs. In the theory of WFSs, being an $\mathcal{L}$-map or $\mathcal{R}$-map was a *property* of morphisms. An NWFS is based on a functorial factorization, giving a canonical, well-behaved choice for the factorizations. We also impose additional structure, making it so that being an $\mathcal{L}$- or $\mathcal{R}$-map is now no longer a property, but a (co)algebraic *structure* on the morphisms. This may seem like a subtle difference, but it has major consequences. For example, we are able to 'fix' the issues with the axiom of choice that we ran into when proving certain closedness properties of WFSs. Still, we are able to show that it has nearly the same closedness properties that we want a WFS to have. We will also show that an NWFS gives us a WFS in a canonical way.

The main thing we are interested in though, is Garner's refinement of Quillen's *small object argument*, allowing us to construct *cofibrantly generated NWFSs*. This is a construction that allows one to define NWFSs in a category given a sufficiently well-behaved morphism class. Garner's construction fixes some of the problematic aspects he saw in Quillen's theory, like the way NWFSs 'fix' the problems in WFSs. This chapter builds up the necessary theory to understand this construction.

We will follow two articles by Garner on cofibrantly generated natural weak factorizations and the small object argument [9] [22]. Since these articles were published, a lot of work has been done on the theory of displayed categories [11]. Displayed categories form a very natural framework for developing the theory of functorial factorizations and natural weak factorization systems, so we will be redefining some constructions in terms of displayed categories. We will explain why this is the most natural, and possibly the only workable way to define these notions in formalization.

5.1 Functorial Factorizations

We have seen before how $\mathcal{C}^{\mathbf{3}}$ is defined as a displayed category over $\mathcal{C}^{\mathbf{2}}$. This allows us to define a functorial factorization as a section of this displayed category.

Definition 5.1 (`functorial_factorization`)**.** *A functorial factorization F over a category $\mathcal{C}$ is a section of the canonical projection*

$$d_1 : \mathcal{C}^{\mathbf{3}} \longrightarrow \mathcal{C}^{\mathbf{2}}.$$

```
Definition functorial_factorization (C : category) := section_disp (three_disp C).
```

We will commonly denote the image of a morphism $f : X \to Y$ under a functorial factorization as a composite, consisting of two morphisms λ_f and ρ_f, factoring through a 'middle object' E_f

$$X \xrightarrow{\lambda_f} E_f \xrightarrow{\rho_f} Y.$$

Interchangably, we may denote the morphisms as f_{01} and f_{12} respectively. We will use the notation that we have introduced for the three category when we talk about a functorial factorization applied to a morphism $\gamma : f \to g$ in the arrow category $\mathcal{C}^{\mathbf{2}}$:

$$\begin{array}{ccc} X & \xrightarrow{F(\gamma)_{00}} & A \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_g} \\ E_f & \xrightarrow{F(\gamma)_{11}} & E_g \\ {\scriptstyle \rho_f}\downarrow & & \downarrow{\scriptstyle \rho_g} \\ Y & \xrightarrow[F(\gamma)_{22}]{} & B \end{array} \quad \left(\begin{array}{ccc} X & \xrightarrow{\gamma_{00}} & A \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_g} \\ E_f & \xrightarrow{F(\gamma)_{11}} & E_g \\ {\scriptstyle \rho_f}\downarrow & & \downarrow{\scriptstyle \rho_g} \\ Y & \xrightarrow[\gamma_{11}]{} & B \end{array} \right)$$

And when we are considering multiple functorial factorizations, the $\lambda_{(-)}$, $\rho_{(-)}$ and $E_{(-)}$ notations will be made distinguishable using superscripts (like $\lambda'_{(-)}$ for the left map corresponding to a functorial factorization F').

Besides the canonical projection d_1, recall the two other projections $d_0, d_2 : \mathcal{C}^{\mathbf{3}} \longrightarrow \mathcal{C}^{\mathbf{2}}$, which are defined in Chapter 3 as

$$d_0 : \left(X_0 \xrightarrow{f_{01}} X_1 \xrightarrow{f_{12}} X_2\right) \mapsto X_1 \xrightarrow{f_{12}} X_2$$
$$d_2 : \left(X_0 \xrightarrow{f_{01}} X_1 \xrightarrow{f_{12}} X_2\right) \mapsto X_0 \xrightarrow{f_{01}} X_1.$$

Postcomposing a functorial factorization F with these projections d_0 or d_2 picks out the right and left map respectively. We obtain functors $R, L : \mathcal{C}^{\mathbf{2}} \longrightarrow \mathcal{C}^{\mathbf{2}}$, sending a morphism f to its right map ρ_f or left map λ_f respectively.

```
Definition fact_R {C : category} (F : functorial_factorization C) :
    arrow C ⟶ arrow C :=
  F • face_map_0.

Definition fact_L {C : category} (F : functorial_factorization C) :
    arrow C ⟶ arrow C :=
  F • face_map_2.
```

Defining a functorial factorization as a section in this way, the left and right functors L and R are automatically compatible. That is to say, for any morphism $f : X \to Y$, we have

$$L(f) \cdot R(f) =_{X \to Y} f$$

This equality is well-typed because the domains and codomains of $L(f)$ and $R(f)$ are what they should be, in a *definitional* way. One might try to define a functorial factorization in a more naive way, say something like

```
Definition functorial_factorization' (C : category) :=
  ∑ F : (arrow C ⟶ three C),
    F • face_map_1 = functor_identity (arrow C).
```

This is a definition which is more in line with a set theoretical approach: I have an object, which satisfies this proposition. Whereas in HoTT, we would much rather want to define these constraints to be baked in directly, as explained in Chapter 2. Of course, defining functorial factorizations as sections achieves precisely this.

Constructively, from a HoTT point of view, the set theoretical approach does not make much sense at all, since the requirement

```
F • face_map_1 = functor_identity (arrow C)
```

is an equality of functors. The first problem is that this relation is not propositional, complexifying comparisons between terms of this notion of functorial factorizations. However, there is a bigger problem with this definition. With this naive definition, though the composite $L(f)\cdot R(f)$ is defined, it may not have the same domain and codomain as f. We merely know that their domains and codomains are *propositionally* equal by the requirement on the functorial factorization. This makes it so the equality $L(f)\cdot R(f) = f$ is *ill-typed*. We can find terms

$$p : \mathrm{dom}(L(f)) =_{\mathcal{C}} \mathrm{dom}\, f \qquad \text{and} \qquad q : \mathrm{cod}(R(f)) =_{\mathcal{C}} \mathrm{cod}\, f$$

and use the fact that propositional equalities give isomorphisms through `idtoiso` to get a well-typed equality

$$f =_{X\to Y} (\texttt{idtoiso}\ p)^{-1} \cdot L(f)\cdot R(f)\cdot \texttt{idtoiso}\ q.$$

We might still be able to build a theory on this, though the whole theory would become messy and riddled with these sorts of isomorphisms.

Example 5.2 (`cop_functorial_factorization`)**.** *The factorization facotring $f : X \to Y$ as*

$$X \xrightarrow{\mathtt{in}^{\sqcup}_X} X \sqcup Y \xrightarrow{f\sqcup \mathrm{id}_Y} Y.$$

from the previous chapter is in fact a functorial factorization.

In fact, this forms a functorial factorization in any category with binary coproducts, not just **SET**.

We define two natural transformations $\Phi : L \Longrightarrow \mathrm{id}_{\mathcal{C}^{\mathbf{2}}}$ and $\Lambda : \mathrm{id}_{\mathcal{C}^{\mathbf{2}}} \Longrightarrow R$ with components

$$\Phi_f := \begin{array}{ccc} X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle f} \\ Ef & \xrightarrow[\rho_f]{} & Y \end{array} \qquad \text{and} \qquad \Lambda_f := \begin{array}{ccc} X & \xrightarrow{\lambda_f} & Ef \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \rho_f} \\ Y & = & Y \end{array}$$

These natural transformations turn L and R into *pointed endofunctors*:

Definition 5.3. *A* pointed endofunctor (T, η) *on a category $\mathcal{C}$ is a functor $T : \mathcal{C} \longrightarrow \mathcal{C}$ together with a natural transformation $\eta : \mathrm{id}_{\mathcal{C}} \Longrightarrow T$.*

Remark. *A pointed endofunctor is like a monad without its multiplication. It is a pointed object in the category of endofunctors.*

To define a natural weak factorization system, we require these pointed endofunctors to be extended to a monad / comonad pair. With this (co)monad pair, we define classes of left and right maps as their (co)algebras. The (co)monad (co)multiplication will guarantee us that for any morphism f, the morphisms (λ_f) and ρ_f are indeed left and right maps respectively, i.e. that they are (co)algebras.

5.2 Natural Weak Factorization Systems

Definition 5.4 (`nwfs`)**.** *A* Natural Weak Factorization System *(NWFS) on a category $\mathcal{C}$ is given by a functorial factorizaton $F : \mathcal{C}^{\mathbf{2}} \longrightarrow \mathcal{C}^{\mathbf{3}}$, together with an extension of the corresponding pointed endofunctor (R, Λ) to a monad $\mathsf{R} = (R, \Lambda, \Pi)$ and the extension of the corresponding copointed endofunctor (L, Φ) to a comonad $\mathsf{L} = (L, \Phi, \Sigma)$. Such a natural weak factorization system (L, R) is said to* lie over *F.*

Later on, it becomes useful to split this definition into two halves: LNWFS and RNWFS, the former containing only data concerning the left comonad, and the latter only containing data concerning the right monad.

Definition 5.5 (`lnwfs_over`)**.** *A* left half of an NWFS *(LNWFS) on a category $\mathcal{C}$ is given by a functorial factorization F, together with an extension of the corresponding copointed endofunctor (L, Φ) to a comonad $\mathsf{L} = (L, \Phi, \Sigma)$.*

```
Definition lnwfs_over {C : category} (F : functorial_factorization C) :=
    ∑ (Σ : (fact_L F) ⟹ (fact_L F) • (fact_L F)), Monad_laws (L_monad_data F Σ).
```

Similarly, we define the *right half of an NWFS* (RNWFS).

```
Definition rnwfs_over {C : category} (F : functorial_factorization C) :=
    ∑ (Π : (fact_R F) • (fact_R F) ⟹ (fact_R F)), Monad_laws (R_monad_data F Π).
```

together, we use these to form the definition of an NWFS:

```
Definition nwfs_over {C : category} (F : functorial_factorization C) :=
    (lnwfs_over F) × (rnwfs_over F).

Definition nwfs (C : category) :=
    ∑ (F : functorial_factorization C), nwfs_over F.
```

Example 5.6. *All WFSs introduced on* **TOP** *in the previous chapter, from the model structures with fibrations being either Serre fibrations or the Hurewicz fibrations, are in fact natural weak factorization systems [9].*

Example 5.7. *The WFSs introduced on* **SSET** *in the previous chapter, from the* Quillen model structure *on* **SSET**, *are in fact natural weak factorization systems [9].*

Example 5.8. *Any strong factorization system gives rise to a functorial factorization. The left and right functor of this factorization can always be extended to an idempotent comonad and an idempotent monad respectively, giving rise to a natural weak factorization system [9].*

Example 5.9 (`cop_nwfs_over`)**.** *The functorial factorization in* **SET** *factoring*

$$X \xrightarrow{\mathbf{in}^{\sqcup}_X} X \sqcup Y \xrightarrow{f \sqcup \mathrm{id}_Y} Y$$

admits an NWFS structure. Note that in this case, λ_{λ_f} is the map

$$X \xrightarrow{\mathbf{in}^{\sqcup}_X} X \sqcup (X \sqcup Y).$$

The map

$$X \sqcup Y \xrightarrow{\mathrm{id}_X \sqcup \mathbf{in}^{\sqcup}_Y} X \sqcup (X \sqcup Y)$$

is the map on codomains $\Sigma_{f\,11}$ for the component of the comultiplication $\Sigma_f : \lambda_f \to \lambda_{\lambda_f}$, and something similar can be done for ρ_f.

In fact, this functorial factorization admits an NWFS in any category with binary coproducts, not just **SET**.

Similar to a plain WFS, an NWFS has left- and right maps. These are defined to be precisely the (co)algebras to the L) and R (co)monad. We denote the (displayed) categories of left- and right maps of an NWFS as (L-**Map**) and R-**Map** respectively.

```
Definition nwfs_L_maps {C : category} (n : nwfs C) :=
    MonadAlg_disp (nwfs_L_monad n).
Definition nwfs_R_maps {C : category} (n : nwfs C) :=
    MonadAlg_disp (nwfs_R_monad n).
```

5.3 Properties of NWFSs

Further solidifying the categorical coherence of NWFSs versus plain WFSs, we form the entities we have defined so far into categories. We then show that the (co)multiplications (Σ) and Π which define the NWFS structure on a functorial factorization have to be of specific shapes. These results come in useful in our formalization. Finally, to relate NWFSs back to plain WFSs, we show that an NWFS allows one to construct a WFS. Firstly though, let us recall some theory on monads.

5.3.1 Monads

As a short interlude, we recall some theory about monads. We will write out some of the laws for the case we are most interested in: endofunctors on the arrow category $\mathcal{C}^{\mathbf{2}}$. A more extensive account on monads can be found in many books on category theory, such as [23].

Definition 5.10 (Monad)**.** *A* monad (T, η, μ) *in a category* $\mathcal{C}$ *consists of an endofunctor* $T : \mathcal{C} \longrightarrow \mathcal{C}$*, together with natural transformations* $\eta : \mathrm{id}_{\mathcal{C}} \Longrightarrow T$ *and* $\mu : T^2 \Longrightarrow T$*, called the* unit *and* multiplication*, such that the following diagrams commute:*

$$\begin{array}{ccccc} \mathrm{id}_{\mathcal{C}} \cdot T & \xrightarrow{\eta T} & T^2 & \xleftarrow{T\eta} & T \cdot \mathrm{id}_{\mathcal{C}} \\ & \searrow^{\cong} & \downarrow{\scriptstyle \mu} & \swarrow_{\cong} & \\ & & T & & \end{array} \qquad \begin{array}{ccc} T^3 & \xrightarrow{T\mu} & T^2 \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow[\mu]{} & T \end{array}$$

where ηT *denotes the natural transformation with components* $(\eta T)_X := \eta_{TX}$ *and* $T\eta$ *that with components* $(T\eta)_X := T(\eta_X)$*. This concept is known as* whiskering*.*

The first diagram contains the 'left unit law' and the 'right unit law' and the second diagram contains the 'associativity law'. A comonad is a monad in the opposite category. In the upcoming chapters, we will be looking at comonads a lot more than monads. Just for illustration, let us draw out the diagrams for (the duals of) these three laws for a functorial factorization F for which the left functor has been extended to a comonad (L, Φ, Σ) in the arrow category $\mathcal{C}^{\mathbf{2}}$. We draw the diagrams for the components at an arrow $f : X \to Y$. The 'left unit law' says that

$$\begin{array}{ccccc} X & \xrightarrow{\Sigma_{f\,00}} & X & \xrightarrow{\Phi_{\lambda_f\,00}} & X \\ {\scriptstyle \lambda_f}\downarrow & & {\scriptstyle \lambda_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle \lambda_f} \\ E_f & \xrightarrow[\Sigma_{f\,11}]{} & E_{\lambda_f} & \xrightarrow[\Phi_{\lambda_f\,11}]{} & E_f \end{array} = \mathrm{id}_{\lambda_f}$$

The 'right unit law' says that

$$\begin{array}{ccccc} X & \xrightarrow{\Sigma_{f\,00}} & X & \xrightarrow{F(\Phi_f)_{00}} & X \\ {\scriptstyle \lambda_f}\downarrow & & {\scriptstyle \lambda_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle \lambda_f} \\ E_f & \xrightarrow[\Sigma_{f\,11}]{} & E_{\lambda_f} & \xrightarrow[F(\Phi_f)_{11}]{} & E_f \\ & & {\scriptstyle \rho_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle \rho_f} \\ & & E_f & \xrightarrow[F(\Phi_f)_{22}]{} & Y \end{array} = \mathrm{id}_{\lambda_f}$$

where the equality is only on the composition of the solid horizontal arrows, as morphisms in $\mathcal{C}^{\mathbf{2}}$. The 'associativity law' says that

$$\begin{array}{ccccc} X & \xrightarrow{\Sigma_{f\,00}} & X & \xrightarrow{F(\Sigma_f)_{00}} & X \\ {\scriptstyle\lambda_f}\downarrow & & {\scriptstyle\lambda_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle\lambda_{\lambda_{\lambda_f}}} \\ E_f & \xrightarrow[\Sigma_{f\,11}]{} & E_{\lambda_f} & \xrightarrow[F(\Sigma_f)_{11}]{} & E_{\lambda_{\lambda_f}} \\ & & {\scriptstyle\rho_{\lambda_f}}\vdots & & \vdots{\scriptstyle\rho_{\lambda_{\lambda_f}}} \\ & & E_f & \xrightarrow[F(\Sigma_f)_{22}]{} & E_{\lambda_f} \end{array} \quad = \quad \begin{array}{ccccc} X & \xrightarrow{\Sigma_{f\,00}} & X & \xrightarrow{\Sigma_{\lambda_{f\,00}}} & X \\ {\scriptstyle\lambda_f}\downarrow & & {\scriptstyle\lambda_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle\lambda_{\lambda_{\lambda_f}}} \\ E_f & \xrightarrow[\Sigma_{f\,11}]{} & E_{\lambda_f} & \xrightarrow[\Sigma_{\lambda_{f\,11}}]{} & E_{\lambda_{\lambda_f}} \end{array}$$

where again the equality is only on the composition of the solid horizontal arrows, as morphisms in $\mathcal{C}^{\mathbf{2}}$. We define morphisms of (co)monads as well.

Definition 5.11 (Monad_Mor)**.** *A morphism between monads (T, η, μ) and (T', η', μ') is a natural transformation $\tau : T \Longrightarrow T'$ such that at any $X : \mathcal{C}$, τ 'preserves the unit'*

$$\eta_X \cdot \tau_X = \eta'_X.$$

and τ 'preserves the multiplication',

$$\mu_X \cdot \tau_X = \tau_{TX} \cdot T'(\tau_X) \cdot \mu'_X.$$

The definition for a comonad morphism is of course dual. Again, let us draw out the diagrams for this in the case of functorial factorizations F and F', for which the left functors were extended to comonads (L, Φ, Σ) and (L', Φ', Σ') respectively. The unit preservation law says that, for any $f : X \to Y$ in $\mathcal{C}^{\mathbf{2}}$, we have

$$\begin{array}{ccccc} X & \xrightarrow{\tau_{f\,00}} & X & \xrightarrow{\Phi_{f\,00}} & X \\ {\scriptstyle\lambda'_f}\downarrow & & {\scriptstyle\lambda_f}\downarrow & & \downarrow{\scriptstyle f} \\ E'_f & \xrightarrow[\tau_{f\,11}]{} & E_f & \xrightarrow[\Phi_{f\,11}]{} & Y \end{array} \quad = \quad \begin{array}{ccc} X & \xrightarrow{\Phi'_{f\,00}} & X \\ {\scriptstyle\lambda'_f}\downarrow & & \downarrow{\scriptstyle f} \\ E'_f & \xrightarrow[\Phi'_{f\,11}]{} & Y \end{array}$$

and the multiplication preservation law says that

$$\begin{array}{ccccc} X & \xrightarrow{\tau_{f\,00}} & X & \xrightarrow{\Sigma_{f\,00}} & X \\ {\scriptstyle\lambda'_f}\downarrow & & {\scriptstyle\lambda_f}\downarrow & & \downarrow{\scriptstyle\lambda_{\lambda_f}} \\ E'_f & \xrightarrow[\tau_{f\,11}]{} & E_f & \xrightarrow[\Sigma_{f\,11}]{} & E_{\lambda_f} \end{array} \quad = \quad \begin{array}{ccccccc} X & \xrightarrow{\Sigma'_{f\,00}} & X & \xrightarrow{F'(\tau_f)_{00}} & X & \xrightarrow{\tau_{\lambda_{f\,00}}} & X \\ {\scriptstyle\lambda'_f}\downarrow & & \downarrow{\scriptstyle\lambda'_{\lambda'_f}} & & \downarrow{\scriptstyle\lambda'_{\lambda_f}} & & \downarrow{\scriptstyle\lambda_{\lambda_f}} \\ E'_f & \xrightarrow[\Sigma'_{f\,11}]{} & E'_{\lambda'_f} & \xrightarrow[F'(\tau_f)_{11}]{} & E'_{\lambda_f} & \xrightarrow[\tau_{\lambda_{f\,11}}]{} & E_{\lambda_f} \\ & & {\scriptstyle\rho'_{\lambda'_f}}\vdots & & \vdots{\scriptstyle\rho'_{\lambda_f}} & & \\ & & E'_f & \xrightarrow[F'(\tau_f)_{22}]{} & E_f & & \end{array}$$

Again, the equality is only on the compositions of the solid arrows, as morphisms of $\mathcal{C}^{\mathbf{2}}$. Some information can be derived from these diagrams pretty easily for (L)NWFSs. Lastly though, before we do that, we recall the definition of a (co)monad (co)algebra, which we used to define the (L-**Map**) and R-**Map** categories.

Definition 5.12 (MonadAlg_disp)**.** *Given a monad (T, η, μ) on $\mathcal{C}$, a T-algebra is an object X of $\mathcal{C}$ together with an arrow $\alpha : TX \to X$ for which the following diagrams commute*

$$\begin{array}{ccc} T^2X & \xrightarrow{T\alpha} & TX \\ {\scriptstyle\mu_X}\downarrow & & \downarrow{\scriptstyle\alpha} \\ TX & \xrightarrow[\alpha]{} & X \end{array} \qquad \begin{array}{ccc} X & \xrightarrow{\eta_X} & TX \\ & \searrow & \downarrow{\scriptstyle\alpha} \\ & & X \end{array}$$

Monad algebras form a displayed category over $\mathcal{C}$.

Yet again, the definition for a coalgebra of a comonad is dual, also forming a displayed category over $\mathcal{C}$. Let us draw out the unit axiom for an extended left functor (L, Φ, Σ) of a functorial factorization F, witnessing f as a retract of λ_f:

$$\begin{array}{ccccc} X & \xrightarrow{\alpha_{00}} & X & \xrightarrow{\Phi_{f\,00}} & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \lambda_f} & & \downarrow{\scriptstyle f} \\ Y & \xrightarrow[\alpha_{11}]{} & E_f & \xrightarrow[\Phi_{f\,11}]{} & Y \end{array} = \mathrm{id}_f \,.$$

Example 5.13 (`cop_nwfs_r_map_iff_split_epi`)**.** *For the factorization on* **SET** *which factors $f : X \to Y$ through $X \sqcup Y$, an* R-**Map** *structure on $g : A \to B$ corresponds with a splitting for g. That is, the existence of a morphism $g^* : B \to A$ such that $g^* \cdot g = \mathrm{id}_B$. Indeed, the unit axiom of the algebra map α says that*

$$\begin{array}{ccccc} A & \xrightarrow{\lambda_g} & A \sqcup B & \xrightarrow{\alpha_{00}} & A \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle \rho_g} & & \downarrow{\scriptstyle g} \\ B & = & B & = & B \end{array}$$

is identity on g, implying that

$$\alpha_{00}\Big|_A \cdot g = \mathrm{id}_B \,.$$

An L-**Map** *structure on $f : X \to Y$ exists just when f is a complemented monomorphism [9]. Indeed, we get the (Complemented Mono, Split Epi) (N)WFS [24].*

Example 5.14. *For a strong factorization system $(\mathcal{L}, \mathcal{R})$, the* L-**Map**s *and* R-**Map**s *from the corresponding NWFS reduce back to $\mathcal{L}$-maps and $\mathcal{R}$-maps [9].*

5.3.2 Categorical Properties

In order to form the entities we have defined so far into categories, we define morphisms them.

Definition 5.15 (`fact_mor`)**.** *A morphism of functorial factorizations $\tau : F \to F'$ between functorial factorizations F, F' over $\mathcal{C}$ is a natural transformation between sections.*

```
Definition fact_mor {C : category} (F F' : functorial_factorization C) :=
    section_nat_trans_disp F F'.
```

Recall that this means that all the components $\tau_f : F(f) \to_{\mathrm{id}_{\mathcal{C}^2}} F'(f)$ of τ lie over the identity morphism.

This allows us to define a category $\mathbf{Ff}_{\mathcal{C}}$ of functorial factorizations over $\mathcal{C}$ (`Ff`). Since we defined NWFSs in terms of functorial factorizations 'with added structure', we can easily define the category of NWFSs on $\mathcal{C}$ as a displayed category over $\mathbf{Ff}_{\mathcal{C}}$. We again split this construction into LNWFSs and RNWFSs, giving us two separate displayed categories $\mathbf{LNWFS}_{\mathcal{C}}$ (`LNWFS`) and $\mathbf{RNWFS}_{\mathcal{C}}$ (`RNWFS`) over $\mathbf{Ff}_{\mathcal{C}}$. Together they yield a displayed category $\mathbf{NWFS}_{\mathcal{C}}$ (`NWFS`) over $\mathbf{Ff}_{\mathcal{C}}$.

In order to do this, we need some added structure on the morphisms in $\mathbf{Ff}_{\mathcal{C}}$. Suppose F and F' are two functorial factorizations underlying NWFSs (L, R) and $(\mathsf{L}', \mathsf{R}')$ respectively. Whiskering a morphism $\tau : F \to F'$ with the other face maps d_0 and d_2 gives natural transformations $\tau_l : L \Longrightarrow L'$ and $\tau_r : R \Longrightarrow R'$ respectively.

```
Definition lnwfs_mor {C : category} {F F' : functorial_factorization C}
    (n : lnwfs_over F) (n' : lnwfs_over F')
    (τ : fact_mor F F') : (lnwfs_L_monad n') ⟹ (lnwfs_L_monad n) :=
  post_whisker (op_nt τ) (functor_opp face_map_2).
Definition rnwfs_mor {C : category} {F F' : functorial_factorization C}
    (n : rnwfs_over F) (n' : rnwfs_over F')
    (τ : fact_mor F F') : (rnwfs_R_monad n) ⟹ (rnwfs_R_monad n') :=
  post_whisker τ face_map_0.
```

Definition 5.16 (`lnwfs_mor_axioms`)**.** *We say that a morphism $\tau : F \to F'$ of functorial factorizations, where F and F' underly LNWFSs, is a* morphism of LNWFSs *if the induced natural transformation τ_l is a comonad morphism.*

```
Definition lnwfs_mor_axioms {C : category} {F F' : functorial_factorization C}
    (n : lnwfs_over F) (n' : lnwfs_over F')
    (τ : fact_mor F F') :=
  Monad_Mor_laws (lnwfs_mor n n' τ).
```

Similarly, we define

Definition 5.17 (`rnwfs_mor_axioms`)**.** *We say that a morphism $\tau : F \to F'$ of functorial factorizations, where F and F' underly RNWFSs, is a* morphism of RNWFSs *if the induced natural transformation τ_r is a comonad morphism.*

```
Definition rnwfs_mor_axioms {C : category} {F F' : functorial_factorization C}
    (n : rnwfs_over F) (n' : rnwfs_over F')
    (τ : fact_mor F F') :=
  Monad_Mor_laws (rnwfs_mor n n' τ).
```

Together, this gives us

Definition 5.18 (`nwfs_mor_axioms`)**.** *We say that a morphism $\tau : F \to F'$ of functorial factorizations, where F and F' underly NWFSs, is a* morphism of NWFSs *if τ is both a morphism of LNWFSs and of RNWFSs.*

```
Definition nwfs_mor_axioms {C : category} (n n' : nwfs C) (τ : fact_mor n n') :=
    lnwfs_mor_axioms n n' τ × rnwfs_mor_axioms n n' τ.
```

Now, since all of the `..._mor_axioms` types are propositional, we can use the Structure Identity Principle to easily define $\mathbf{LNWFS}_{\mathcal{C}}$, $\mathbf{RNWFS}_{\mathcal{C}}$ and $\mathbf{NWFS}_{\mathcal{C}}$, the categories of left halves, right halves, and full natural weak factorization systems on $\mathcal{C}$, as displayed categories over $\mathbf{Ff}_{\mathcal{C}}$.

5.3.3 Properties of the (Co)multiplication

The multiplication and comultiplication that define an NWFS structure on a functorial factorization can be shown to be of specific shapes, and have certain useful properties in proofs later on. Grandis and Tholen elaborated a bit on this in their paper [8]. These shapes follow pretty much directly from the (co)monad axioms, in combination with the known shape of the unit Λ and the counit Φ for any given functorial factorization. In all of these lemmas, we assume we are given an NWFS (L, R) with monad multiplication Π and comonad comultiplication Σ and we consider the components at some morphism $f : X \to Y$.

Lemma 5.19 (`nwfs_Σ_top_map_id`, `nwfs_Π_bottom_map_id`)**.** *The multiplication and comultiplication are of the following shapes:*

$$\Sigma_f = \begin{array}{ccc} X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_f & \xrightarrow[\sigma_f]{} & E_{\lambda_f} \end{array} \qquad \Pi_f = \begin{array}{ccc} E_{\rho_f} & \xrightarrow{\pi_f} & E_f \\ {\scriptstyle \rho_{\rho_f}}\downarrow & & \downarrow{\scriptstyle \rho_f} \\ Y & = & Y \end{array}$$

Furthermore, we can show some relations on the morphisms σ_f and π_f [8, 2.2].

Lemma 5.20. *Let $F : \mathcal{C}^{\mathbf{2}} \longrightarrow \mathcal{C}^{\mathbf{3}}$ be the underlying functorial factorization of (L, R). The maps σ_f and π_f have the following properties:*

$$\sigma_f \cdot \rho_{\lambda_f} = \mathrm{id}_{E_f} \qquad \lambda_{\rho_f} \cdot \pi_f = \mathrm{id}_{E_f}$$
$$\sigma_f \cdot \sigma_{\lambda_f} = \sigma_f \cdot F(\Sigma_f)_{11} \qquad \pi_{\rho_f} \cdot \pi_f = F(\Pi_f)_{11} \cdot \pi_f$$

Now we can in fact show similar lemmas when we are given only an LNWFS or an RNWFS (`lnwfs_Σ_top_map_id`, `lnwfs_Σ_bottom_map_inv`), since we do not use any information about the other half.

5.3.4 An NWFS is a WFS

One of the most important properties of an NWFS is that it in fact gives us a WFS. We follow the proof from [9, Section 2.15]. First of all, let $f : X \to Y$ be a morphism, and consider potential (co)algebra maps $(\alpha_\lambda : f \to \lambda_f)$ and $\alpha_\rho : \rho_f \to f$. The (co)algebra axioms force these maps to be of a particular shape. By the coalgebra axioms, we have

$$\alpha_\lambda \cdot \Phi_f = \begin{array}{ccccc} X & \longrightarrow & X & = & X \\ {\scriptstyle f}\downarrow & & {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle f} \\ Y & \xrightarrow[s]{} & Ef & \xrightarrow[\rho_f]{} & Y \end{array} = \begin{array}{ccc} X & = & X \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ Y & = & Y \end{array}$$

and so we see that the map on domains of α_λ must be identity. Similarly, the map on codomains of α_ρ is identity. We also obtain a section s of ρ_f, and similarly, a retraction p of λ_f from the monad algebra axioms.

```
Lemma R_map_section {C : category} {n : nwfs C} {a b : C} {f : a --> b}
    (hf : nwfs_L_maps n f) :
  ∑ s, f · s = arrow_mor (fact_L n f) ×
     s · arrow_mor (fact_R n f) = identity _.

Lemma L_map_retraction {C : category} {n : nwfs C} {c d : C} {g : c --> d}
    (hg : nwfs_R_maps n g) :
  ∑ p, p · g = arrow_mor (fact_R n g) ×
     arrow_mor (fact_L n g) · p = identity _.
```

Using this, we can construct a (specific) lift given a lifting problem (h, k) between L-map f and an R-map g.

Lemma 5.21 (`L_map_R_map_elp`)**.** *Let (L, R) be an NWFS over a functorial factorization F, then there exists a lift for every (f, g)-lifting problem (h, k) where f is an* L**-Map** *and g an* R**-Map**.

Proof. Let $f : X \to Y$ and $g : A \to B$. Consider the diagram obtained by applying the functorial factorization F to (h, k) and attaching the (co)monad algebra morphisms.

$$\begin{array}{ccccccc} X & = & X & \xrightarrow{h} & A & & \\ {\scriptstyle f}\downarrow & & {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_g} & & \\ Y & \xrightarrow[s]{} & E_f & \xrightarrow{F(h,k)_{11}} & E_g & \xrightarrow{p} & A \\ & & {\scriptstyle \rho_f}\downarrow & & \downarrow{\scriptstyle \rho_g} & & \downarrow{\scriptstyle g} \\ & & Y & \xrightarrow[k]{} & B & = & B \end{array}$$

The lift can be simply read off as $s \cdot F(h, f)_{11} \cdot p$. □

It is important to note that we in fact prove the *existential* lifting property, meaning that we obtain the actual lift, and not just the mere existence of one. This is an important difference with plain WFSs, where we only know of the mere existence of a lift. The lemma we just proved shows that the existence of a lift between an L-**Map** and an R-**Map** corresponds precisely with the existence of their (co)algebra maps. After all, the existence of a coalgebra map is basically an algebraic way of saying that '(f, ρ_f) has the lifting property' and dually for the existence of an algebra map. In this way, the (co)algebra maps and the lifting property correspond precisely with Lemma 4.21 in the theory of plain WFSs, providing a lift of an L-**Map** f against its right map ρ_f and a lift of an R-**Map** against its left map λ_f.

We will show that an NWFS actually gives us a plain WFS in a canonical way. First note that λ_f can be given an L-**Map** structure for any morphism f. This structure comes directly from the comultiplication of the NWFS. The coalgebra laws follow directly from the lemmas in the previous section.

Lemma 5.22 (`nwfs_Lf_is_L_map`)**.** *Let* (L, R) *be an NWFS. Then for any morphism* f*, the corresponding morphism* λ_f *can be given an* L-**Map** *structure.*

Similarly, one can show that ρ_f can be given an R-**Map** structure. The previous two lemmas pretty much tell us how to construct a WFS from an NWFS.

Theorem 5.23 (`nwfs_is_wfs`)**.** *Given an NWFS* (L, R)*, the pair* $(\mathsf{L}\text{-}\mathbf{Map}^{cl}, \mathsf{R}\text{-}\mathbf{Map}^{cl})$ *is a WFS.*

Proof. We have to show a couple of things. Firstly, we show that $\mathsf{L}\text{-}\mathbf{Map}^{\mathrm{cl}} \subseteq {}^{\square}\left(\mathsf{R}\text{-}\mathbf{Map}^{\mathrm{cl}}\right)$. In fact, we can show something stronger, which is that $\mathsf{L}\text{-}\mathbf{Map} \subseteq {}^{\square}\mathsf{R}\text{-}\mathbf{Map}$. This follows directly from Lemma 5.21. It follows that $\mathsf{L}\text{-}\mathbf{Map}^{\mathrm{cl}} \subseteq {}^{\square}\left(\mathsf{R}\text{-}\mathbf{Map}^{\mathrm{cl}}\right)$, since retracts preserve the lifting property, from Lemma 4.9.

Next we show that ${}^{\square}\left(\mathsf{R}\text{-}\mathbf{Map}^{\mathrm{cl}}\right) \subseteq \mathsf{L}\text{-}\mathbf{Map}^{\mathrm{cl}}$. In this case, we can not (in general) prove a stronger statement, as is explained in the remark after the proof. Now let $f \in {}^{\square}\left(\mathsf{R}\text{-}\mathbf{Map}^{\mathrm{cl}}\right)$. Note that since f has the left lifting property with respect to any morphism in $\mathsf{R}\text{-}\mathbf{Map}^{\mathrm{cl}}$, it has the left lifting property with respect to any map in R-**Map** as well, and in particular with ρ_f, since ρ_f is an R-map. We obtain a filler l for the following lifting problem.

$$\begin{array}{ccc} X & \xrightarrow{\lambda_f} & E_f \\ {\scriptstyle f}\downarrow & \overset{l}{\nearrow} & \downarrow{\scriptstyle \rho_f} \\ Y & = & Y \end{array}$$

Reshaping this diagram a bit, we see that f is a retract of λ_f, which has an L-**Map** structure, as shown in Lemma 5.22.

$$\begin{array}{ccccc} X & = & X & = & X \\ {\scriptstyle f}\downarrow & & {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle f} \\ Y & \xrightarrow{l} & E_f & \xrightarrow{\rho_f} & Y \end{array}$$

Dually, one can show that $\mathsf{R}\text{-}\mathbf{Map}^{\mathrm{cl}} = \left(\mathsf{L}\text{-}\mathbf{Map}^{\mathrm{cl}}\right)^{\square}$. The factorization axiom for the WFS follows precisely from the facts that λ_f is an L-map, and that ρ_f is an R-map. □

Remark. *It is not true in general that* ${}^{\square}\mathsf{R}$-**Map** $\subseteq \mathsf{L}$-**Map**. *Suppose* $f \in {}^{\square}\mathsf{R}$-**Map**, *then* f *has the lifting property with respect to any* $g \in \mathsf{R}$-**Map**. *In order to show that* f *is an* L-**Map**, *we need to find the* L-**Map***-structure map (the coalgebra morphism). The only tools we have to construct this map are the lifts with respect to the* R-**Map**s, *but these lifts (as given from the assumption) do not necessarily have to be natural, or correspond with a coalgebra*

structure in any way. In his article, Garner merely mentions that all the laws for a WFS are satisfied 'except possibly for closure under retracts' [9, Remark 2.17]. However, for a general case like this in a formalization, we will always have to assume that we have to take the retract closures.

And so we see that an NWFS indeed gives us a WFS. But NWFSs are better behaved in many ways. They have algebraic structure and provide a canonical and natural choice for fillers in lifting problems. As we will see, they allow for a much more elegant version of the *small object argument*, used to generate WFSs. Before that, we show that they 'fix' the problems we had with plain WFSs. We no longer need the axiom of choice to show equivalents of the lemmas we were not able to before.

Lemma 5.24 (`nwfs_closed_coproducts`)**.** *Given families of objects* $\{ X_i \}_{i:I}, \{ Y_i \}_{i:I}$ *for some type* I*. Suppose we have a family of maps* $\{ f_i \}_{i:I}$ *such that* f_i *is an* L**-Map** *for all* $i : I$*. Then the coproduct* $f : \bigsqcup_{i:I} X_i \to \bigsqcup_{i:I} Y_i$ *of the* f_i *also has an* L**-Map** *structure.*

The proof for this lemma is very similar to the proof we *tried* to use for plain WFSs (in Lemma 4.27). We get coalgebra maps for every individual f_i, and we can put this together to obtain coalgebra data for f.

We can also show the following.

Lemma 5.25 (`nwfs_closed_transfinite_composition`)**.** *Let* $d := \{ X_v \}_{v:\mathbb{N}}$ *be a chain in* $\mathcal{C}$*, for which every*

$$f_v : X_v \to X_{v+1}$$

has an L**-Map** *structure, then the canonical map*

$$\mathtt{in}_0^{\to} : X_0 \hookrightarrow \operatorname{colim} X_v$$

is a retract of an L**-Map**.

Proof. The strategy for this proof is similar to the analogue for WFSs. We want to show that we have a map α_∞ that fills the following diagram:

$$\begin{array}{ccc} X_0 & = & X_0 \\ {\scriptstyle \mathtt{in}_0^{\to}}\downarrow & & \downarrow{\scriptstyle \lambda_{\mathtt{in}_0^{\to}}} \\ X_\infty & \xrightarrow[\alpha_\infty]{} & E_{\mathtt{in}_0^{\to}} \end{array}$$

The map α_∞ that we construct indeed shows that $\mathtt{in}_0^{\to}$ is a retract of $\lambda_{\mathtt{in}_0^{\to}}$. We claim that for any $v : \mathbb{N}$, we have a map α_v such that

$$\begin{array}{ccc} X_v & \xrightarrow{\alpha_v} & E_{\mathtt{in}_0^{\to}} \\ {\scriptstyle f_v}\downarrow & & \downarrow{\scriptstyle \rho_{\mathtt{in}_0^{\to}}} \\ X_{v+1} & \xrightarrow[\mathtt{in}_{v+1}^{\to}]{} & X_\infty \end{array}$$

We set $\alpha_0 := \lambda_{\mathtt{in}_0^{\to}}$. Then suppose we have a map α_v that fills the above diagram. For $v+1$, we obtain a map α_{v+1} by finding a lift from f_v against $\rho_{\mathtt{in}_0^{\to}}$, using the algebra map obtained from the multiplication Π for the latter. We get a diagram

$$\begin{array}{ccccccc} X_v & = & X_v & \xrightarrow{\alpha_v} & E_{\mathtt{in}_0^{\to}} & & \\ {\scriptstyle f_v}\downarrow & & {\scriptstyle \lambda_{f_v}}\downarrow & & \downarrow{\scriptstyle \lambda_{\rho_{\mathtt{in}_0^{\to}}}} & & \\ X_{v+1} & \xrightarrow[\alpha_{f_v}]{} & E_v & \xrightarrow{F(\sigma_v, \mathtt{in}_{v+1}^{\to})_{11}} & E_{\rho_{\mathtt{in}_0^{\to}}} & \xrightarrow{\pi_{\rho_{\mathtt{in}_0^{\to}}}} & E_{\mathtt{in}_0^{\to}} \\ & & {\scriptstyle \rho_{f_v}}\downarrow & & \downarrow{\scriptstyle \rho_{\rho_{\mathtt{in}_0^{\to}}}} & & \downarrow{\scriptstyle \rho_{\mathtt{in}_0^{\to}}} \\ & & X_{v+1} & \xrightarrow[\mathtt{in}_{v+1}^{\to}]{} & X_\infty & = & X_\infty \end{array}$$

where we can read off $\alpha_{v+1} : X_{v+1} \to E_{\mathtt{in}_0^{\to}}$. Indeed, these morphisms define α_∞, and one can show that $\mathtt{in}_0^{\to}$ is a retract of $\lambda_{\mathtt{in}_0^{\to}}$, which is an L-**Map**. □

Remark. *We would much rather show that* $\mathtt{in}_0^{\to}$ *is an* L-**Map** *itself. We are almost able to do so, as the retract with* $\lambda_{\mathtt{in}_0^{\to}}$ *is in fact the same as the unit axiom of an* L-**Map**. *Sadly, the multiplication axiom requires some specific interaction between the comultiplication* Σ *and the multiplication* Π *of our NWFS. A common assumption is* distributivity *[9], which may be sufficient.*

Remark. *It is important to note that these two proofs do* not *show that the WFS one obtains from an NWFS is closed under coproducts and transfinite compositions. After all, we would run into the same issue as before: we would only know the mere existence of a lift in the 'individual diagrams' in both proofs. We would still not be able to 'put them together', yielding a lift for the full problem. These lemmas* do *show that we can assume such statements to hold when we build a theory on the more refined notion of an NWFS.*

Remark. *These algebraic analogues to weak factorization systems can be used to define an algebraic analogue to model structures: algebraic model structures [10]. Like for weak factorization systems, these solve some of the issues one runs into with plain model structures, such as the functoriality of NWFSs providing a canonical choice of lifts.*

Chapter 6

The Classical Small Object Argument

Quillen's (classical) small object argument [1] is a powerful technique that allows one to construct weak factorization systems given a sufficiently well-behaved class of morphisms. It can even be used to construct model structures on categories, known as cofibrantly generated model structures. Garner improved on this construction to allow one to generate NWFSs instead, in an algebraically coherent way. His construction also resolves some problematic aspects he saw in the classical small object argument, being that it has no universal property, it does not converge and it does not seem to be related to other transfinite constructions in categorical algebra [9].

In this chapter, we introduce the classical small object argument, following Hovey's book on model categories [2]. When defining the argument, we generally keep the examples of topological spaces (**TOP**) or simplicial sets (**SSET**) in mind, to gain some intuition behind the construction. We also point out a constructive issue in this construction. In the next chapter, we will discuss Garner's 'algebraization' of this argument, elaborating on his articles [9][22] and drawing parallels with the classical counterpart discussed in this chapter.

There is no formalization of the theory in this chapter, so we will be using notation closer to that of set-theory based mathematics, in order to stay closer to the original theory.

6.1 The Small Object Argument

We define what exactly a 'small object' is.

Definition 6.1. *Suppose $\mathcal{C}$ is a category with all small colimits, and J is a class of morphisms of $\mathcal{C}$. Let A be an object of $\mathcal{C}$ and κ a cardinal. We say that A is* κ-small relative to J *if, for all κ-filtered ordinals γ and γ-sequences*

$$X_0 \to X_1 \to \ldots \to X_\alpha \to \ldots$$

where each map $X_\alpha \to X_{\alpha+1}$ is in J for $\alpha + 1 < \gamma$, the map of sets

$$\mathrm{colim}_{\alpha<\gamma}(A \to X_\alpha) \to (A \to \mathrm{colim}_{\alpha<\gamma} X_\alpha)$$

is an isomorphism. We say that A is small relative to J *if it is κ-small relative to J for some cardinal κ. We say that A is* small *if it is small relative to all morphisms in $\mathcal{C}$.*

Remark. *As mentioned before, the theory of ordinals in HoTT has not been developed very far, even with recent developments [21]. For this reason, we will only consider transfinite composites for chains, so for the first limit ordinal ω (i.e. γ is ω, and κ is some finite value). Since this chapter will not be formalized, we leave the definition as it is in Hovey's book [2],*

but one may keep in mind that the limit ordinal is ω, the sequence is a chain and smallness means small relative to ω. These assumptions will still be sufficient for the construction to work in important examples such as topological spaces and simplicial sets [2].

Intuitively, this means that an object is small relative to J if a map to a long enough composition of maps in J factors through some stage of this composition [2].

Example 6.2. *Any set is small. We show that if A is a set, then A is $|A|$-small. Suppose γ is an $|A|$-filtered ordinal, and X is a γ-sequence of sets. For any map $f : A \to \operatorname{colim}_{\alpha<\gamma} X_\alpha$, we find for each $a \in A$ an index α_a such that $f(a)$ is in the image of X_{α_a}. Then let $\beta = \sup_{a\in A} \alpha_a$. Since γ is $|A|$-filtered, $\beta < \gamma$, and so f factors through some map $A \to X_\beta$.*

Before we get into the small object argument construction, we introduce some terminology and some technical Lemmas.

Definition 6.3. *Given a class of morphisms J in a category $\mathcal{C}$, we call a map a* J-cofibration *if it is in ${}^{\square}\left(J^{\square}\right)$. Dually, we call it a* J-fibration *if it is in $\left({}^{\square}J\right)^{\square}$.*

Like the names suggest, these classes will end up being the (co)fibrations in our weak factorization system. We specify a special subclass of ${}^{\square}\left(J^{\square}\right)$.

Definition 6.4. *Let J be a class of morphisms in a category $\mathcal{C}$. A* relative J-cell complex *is a transfinite composition of pushouts of elements of J. We denote this class by J-cell. That is, if $g : A \to B$ is in J-cell, then there is an ordinal γ and a γ-sequence $X_{(-)} : \gamma \longrightarrow \mathcal{C}$ such that g is the composition of $X_{(-)}$, and such that for each α with $\alpha+1 < \gamma$, there is a pushout square*

$$\begin{array}{ccc} A_\alpha & \longrightarrow & X_\alpha \\ {\scriptstyle g_\alpha}\downarrow & \ulcorner & \downarrow \\ B_\alpha & \longrightarrow & X_{\alpha+1} \end{array}$$

such that g_α is in J. We say that $A : \mathcal{C}$ is a J-cell complex if the map $0 \to A$ is a relative J-cell complex for an initial object 0.

Example 6.5. *In the case of the categories* **TOP** *and* **SSET** *of topological spaces, the class J will be the class of boundary inclusions of spheres into the corresponding disks*

$$J := \{\, j_n : S^n \hookrightarrow D_{n+1} \mid n = -1, 0, 1, \dots \}$$

Note that in this case, these pushout squares correspond precisely to the gluing of cells. This means that maps in J-cell correspond precisely with relative CW complexes on the domain, and J-cell complexes are precisely the absolute CW-complexes, since we take S^{-1} to be the empty space, which is the initial object of **TOP** *and* **SSET**.

It can be shown that all isomorphisms are in J-cell, since we can take them to be the composition of the 1-sequence on their domain. After all, for any other colimit on this sequence, the isomorphism itself gives us a (unique) ismorphism to this colimit.

Remark. *From the definition of J-cell, it need not be a subclass of ${}^{\square}\left(J^{\square}\right)$. This fact follows from the fact that ${}^{\square}\left(J^{\square}\right)$ is closed under pushouts of coproducts and transfinite compositions.* However, that proof required the axiom of choice. *This is a constructive issue in the small object argument that is easily overlooked.*

We need some results about J-cell. More detailed proofs of these Lemmas can be found in [2, Section 2.1.2].

Lemma 6.6. *Suppose $\mathcal{C}$ is a category with all small colimits, and J is a class of morphisms of $\mathcal{C}$. Then J-cell is closed under transfinite compositions.*

Proof Idea. Let $X_{(-)}$ be a γ-sequence of relative J-cell complexes. Then every map $X_\alpha \to X_{\alpha+1}$ is a relative J complex, i.e. the composition of an ordinal sequence $Y_\alpha : \gamma_\alpha \longrightarrow \mathcal{C}$. We define a new ordinal, consisting of pairs of ordinals $\Gamma := \sum_{\alpha:\gamma} \gamma_\beta$ and define a new order on Γ. This gives a new sequence $Z : \Gamma \longrightarrow \mathcal{C}$ such that every map $Z_\alpha \to Z_{\alpha+1}$ is either a map $Y_\gamma \to Y_{\gamma+1}$ or an isomorphism. It is possible to 'collapse' these isomorphisms by taking an equivalence class on the ordinal Γ, giving again a new ordinal sequence, where every map is in fact a pushout of a map of J. By definition, this composition is then again in J-cell. □

In the category of topological spaces, this corresponds with the fact that a transfinite composition of relative CW complexes still yields a relative CW complex.

Lemma 6.7. *Suppose $\mathcal{C}$ is a category with all small colimits, and J is a class of morphisms of $\mathcal{C}$. Then any pushout of (set-indexed) coproducts of maps of J is in J-cell.*

Proof Idea. Suppose K is a set and $g_k : A_k \to B_k$ is a map of J for all k in K. Suppose g is the pushout in the diagram

$$\begin{array}{ccc} \bigsqcup_{k\in K} A_k & \longrightarrow & A \\ {\scriptstyle \bigsqcup_{k\in K} g_k}\downarrow & & \downarrow{\scriptstyle g} \\ \bigsqcup_{k\in K} B_k & \longrightarrow & B \end{array}$$

The idea is that we convert this coproduct of cell attachments into a transfinite composition of sequential cell attachments of the individual g_k, which ends up giving the same pushout. □

As was hinted at in the proof idea, in the category of topological spaces or simplicial sets, this lemma shows that instead of gluing cells individually, we can also glue many cells at once.

With these technical Lemmas, we are equipped to prove that the small object argument construction works.

Theorem 6.8 (Small object argument)**.** *Suppose $\mathcal{C}$ is a category containing all small colimits, and J is a class of morphisms in $\mathcal{C}$. Suppose the domains of all the maps in J are small relative to J-cell. Then there is a functorial factorization (L, R) on $\mathcal{C}$ such that, for all morphisms f in $\mathcal{C}$, $L(f)$ is in J-cell and $R(f)$ is in $J^{\square}$.*

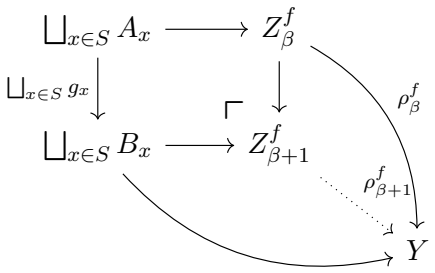
Proof. Choose a cardinal κ such that every domain of J is κ-small relative to J-cell, and let γ be a κ-filtered ordinal. Given $f : X \to Y$ we will define a functorial γ-sequence $Z^f_{(-)} : \gamma \longrightarrow \mathcal{C}$ such that $Z^f_0 = X$ and a natural transformation $Z^f_{(-)} \overset{\rho^f_{(-)}}{\Longrightarrow} Y$ factoring f. Each map $Z^f_\alpha \to Z^f_{\alpha+1}$ will be a pushout of maps in J. Then $L(f)$ will be the composition of Z^f, and $R(f)$ will be the map $E_f := \operatorname{colim} Z^f \to Y$, induced by ρ^f. It follows from Lemma 6.6 and Lemma 6.7 that $L(f)$ is in J-cell.

We define Z^f_α and $\rho^f_\alpha : Z^f_\alpha \to Y$ using transfinite induction. Firstly, we set $Z^f_0 = X$ and $\rho^f_0 = f$. Now assume we have defined Z^f_α and ρ^f_α for all $\alpha < \beta$ for some limit ordinal β. We define $Z^f_\beta = \operatorname{colim}_{\alpha<\beta} Z^f_\alpha$ and ρ^f_β to be the map induced by the ρ^f_α.

Having defined Z^f_β and ρ^f_β, we define $Z^f_{\beta+1}$ and $\rho^f_{\beta+1}$. This is called the 'limit ordinal step'. Let S_f be the set of all commutative squares (i.e. lifting problems)

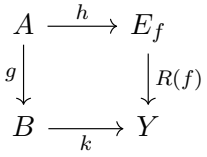
$$\begin{array}{ccc} A & \longrightarrow & Z^f_\beta \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle \rho^f_\beta} \\ B & \longrightarrow & Y \end{array}$$

with g in J. For any lifting problem x in S_f, denote by $g_x : A_x \to B_x$ the corresponding map of J. We take $Z^f_{\beta+1}$ to be the pushout in the following diagram



and set $\rho^f_{\beta+1}$ to be the unique map given from the universal property of the pushout.

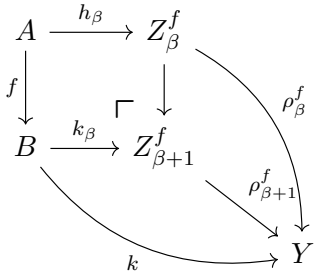
It remains to show that $R(f) = \operatorname{colim} \rho^f_\beta : E_f \to Y$ has the right lifting property with respect to J. To show this, we use the smallness property of the domains of the maps in J. Suppose we are given a commutative square



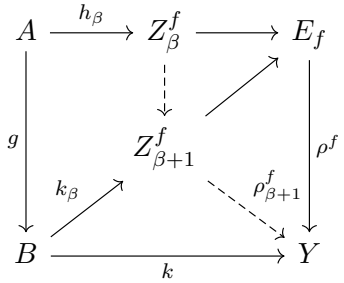
where g is in J. From the smallness of A, it follows that there is a $\beta < \gamma$ such that h is the composite

$$A \xrightarrow{h_\beta} Z^f_\beta \to E_f.$$

If we consider the pushout diagram corresponding to this step in the transfinite sequence, we get a map $k_\beta : B \to Z^f_{\beta+1}$ by construction that makes this diagram commute.



Factoring ρ^f_β and $\rho^f_{\beta+1}$ through the colimit, we get that the following diagram commutes:



The diagonal map here is precisely the lift we are looking for. □

Example 6.9. *In the category* **TOP** *of topological spaces and* **SSET** *of simplicial sets, this construction boils down to the following: Every step in the transfinite sequence, we glue all*

possible cells to the current stage. This means that every step of the way, we glue duplicate cells, as we can glue all the cells that we have glued before, as well as any new ones that attach to the cells from the previous step.

This is what Garner means with the fact that this construction 'does not converge'. The idea is that we just stop whenever we have gone far enough. This again implies some sort of choice, as well as possibly requiring us to work with massive ordinals.

Unpacking the requirement of the smallness of the domains of maps in J with respect to J-cell in **TOP**, we simply mean that including a sphere into a CW-complex factors through some stage in the construction of the CW-complex. A simple argument could be that any compact subset of a CW-complex intersects only finitely many cells. We can in fact prove something stronger though. The following Lemma allows us to use the small object argument in the category of topological spaces, with J the class of sphere inclusions [2, Lemma 2.4.1].

Lemma 6.10. *All topological spaces are small with respect to inclusions.*

Proof. Suppose $X_{(-)}$ is a γ-sequence of topological spaces for some ordinal γ. Then every map $X_\alpha \to X_{\alpha+1}$ is an inclusion, and by transfinite induction, so is any map $X_\alpha \to X_\beta$ for limit ordinals $\beta > \alpha$. So the map $X_\alpha \to \operatorname{colim} X$ is an inclusion. So if we can factor any map $A \to \operatorname{colim} X$ from a space A through a map of *sets* $A \to X_\alpha$, this map is automatically continuous. This follows from the fact that any set is small, as shown in Example 6.2. □

Even stronger though, is the following lemma [2, Proposition 2.4.2], which allows us to use the small object argument with a truncation at the first limit ordinal ω.

Lemma 6.11. *Compact topological spaces are finite with respect to closed T^1 inclusions.*

In his algebraization of the small object argument, Garner introduces two smallness requirements. The first one is a much simpler, but much stronger one. This smallness requirement in fact does not hold in **TOP**, as it requires smallness relative to all morphisms in the category, not just relative to those in J. This is not a problem in the category **SSET** of simplicial sets, as we have the following result [2, Lemma 3.1.2].

Lemma 6.12. *All finite simplicial sets are finite in* **SSET**.

6.2 Cofibrantly Generated Model Structures

With the small object argument, we can generate model structures on a category $\mathcal{C}$. Consider the following theorem [2, Theorem 2.1.9].

Theorem 6.13. *Suppose $\mathcal{C}$ is a category with all small limits and colimits. Suppose $\mathcal{W}$ is a class of morphisms in $\mathcal{C}$, and I and J are morphism classes of $\mathcal{C}$. Then there is a cofibrantly generated model structure on $\mathcal{C}$ with $\mathcal{W}$ as the class of weak equivalences, $I^\square$ as the class of fibrations, and $J^\square = \mathcal{W} \cap (I^\square)$ if and only if the following conditions are satisfied.*

1. *The class $\mathcal{W}$ has the two out of three property and is closed under retracts.*
2. *The domains of I are small relative to I-cell.*
3. *The domains of J are small relative to J-cell.*
4. *I-cell $\subseteq \mathcal{W} \cap {}^\square\left(J^\square\right)$.*
5. *$J^\square \subseteq \mathcal{W} \cap (I^\square)$.*
6. *Either $\mathcal{W} \cap {}^\square\left(J^\square\right) \subseteq {}^\square\left(I^\square\right)$ or $\mathcal{W} \cap (I^\square) \subseteq J^\square$.*

Effectively, we use two classes I and J to generate two weak factorization systems. The other requirements make sure the weak factorization systems interact properly.

Example 6.14. *The* classical model structure *on* **TOP**, *where the weak equivalences are the weak homotopy equivalences, fibrations are Serre fibrations and cofibrations are retracts of relative CW-complexes, is generated By*

$$J := \{\, j_n : S^n \hookrightarrow D_{n+1} \mid n = -1, 0, 1, \dots \}$$

and

$$I := \{\, i_n : D^n \hookrightarrow D_n \times [0,1] \mid n = 0, 1, \dots \}$$

where i_n maps D_n to $D_n \times \{0\}$. Indeed, $I^{\square}$ is the set of Serre fibrations by definition.

Example 6.15. *In* **SSET**, *the class of canonical boundary inclusions*

$$J := \{\, j_n : \partial\Delta[n] \hookrightarrow \Delta[n] \mid n \geq 0 \,\}$$

and the class of canonical inclusions

$$I := \{\, i_{r,n} : \Lambda^r[n] \hookrightarrow \Delta[n] \mid n > 0, 0 \leq r \leq n \,\}$$

generate the Quillen model structure *on* **SSET**. *The fibrations are the Kan fibrations, and the cofibrations are the monomorphisms $f : X \to Y$ which are levelwise injections.*

Hovey describes even more examples of cofibrantly generated model structures, such as that on the category R-Mod of modules for a Frobenius ring R, or on the category of R chain complexes for a ring R [2].

Chapter 7

The Algebraic Small Object Argument

Now that we have seen the classical small object argument, we can move on to the most interesting part of this thesis: generating natural weak factorization systems with Garner's algebraic small object argument. The construction is inductive like its counterpart. The general idea is that for a class J of morphisms in our category $\mathcal{C}$, we want to construct some LNWFS. This LNWFS will have certain properties that will allow us to apply a transfinite construction, generalized from one by Kelly [25], giving us a full NWFS.

The initial LNWFS we construct will correspond to the first step in the classical small object argument. Similar to that construction the 'left part' will already obey the properties we want it to, but the 'right part' still needs to be fixed. The transfinite construction then 'fixes' the right half. In this way, we are able to generate an NWFS (L, R) from only a morphism class J. There will be many similarities between the constructions, but also some obvious differences. The construction also resolves the issues with the axiom of choice that we encountered before.

In fact, the NWFS that is generated this way will be *cofibrantly generated*. That is to say, all the morphisms in J are L-**Map**s, and the class of R-**Map**s corresponds with an algebraized notion of $J^{\square}$ [9]. Unfortunately, this is out of scope for this thesis, and we will be focussing on the existence of an NWFS.

We will be following Garner's articles [9] [22], rephrasing the theory using univalent foundations, filling in details where he left them out and providing intuition using important examples in **TOP** and **SSET**, as well as a computable example in **SET**. We also redefine part of the construction to be more direct and intuitive.

In the rest of this chapter, we will always assume that $\mathcal{C}$ is the category we are working in, and J is a given class of morphisms, which are using to generate our NWFS. Some examples in this chapter will provide intuition about the construction in the case of topological spaces (**TOP**) or simplicial sets (**SSET**). Just like the previous chapter, the class J in these cases consists of boundary inclusions of disks

$$J = \{\, j_n : S^n \hookrightarrow D_{n+1} \mid n = -1, 0, 1, \dots \,\}.$$

Another, simpler example that we consider in examples is that of the factorization on **SET**, factoring $f : X \to Y$ through $X \sqcup Y$. It turns out that the corresponding NWFS is obtained by choosing J to be the set containing the single morphism from the empty set to the one point set [9]

$$J = \{\, \emptyset \to * \,\}.$$

7.1 The One-Step Comonad

As mentioned, we want to construct an NWFS from and appropriate morphism class J. This is an iterative process, which is very similar to the classical small object argument. The first step is to produce the *one-step comonad*. This is an object of $\mathbf{LNWFS}_{\mathcal{C}}$, which we will then use to construct a full NWFS. The one-step comonad corresponds the very first step of the classical small object argument.

Firstly though, at this point the morphism class J is not a category in our formalization, but just a class of morphisms. Classically, we could choose to turn J into a category by simply taking it to be the discrete subcategory of $\mathcal{C}^{\mathbf{2}}$ containing precisely the maps of J as objects. In our case, it is easier to define the full subcategory of J as a displayed category over $\mathcal{C}^{\mathbf{2}}$ where the objects over $g : \mathcal{C}^{\mathbf{2}}$ are the propositional type $g \in J$.

```
Definition morcls_disp (J : morphism_class C) : disp_cat (arrow C) :=
  disp_full_sub (arrow C) (λ g, J _ _ g).
```

Giving us a total category J^{tot} where the objects are precisely $\sum_{g:\mathcal{C}^{\mathbf{2}}} g \in J$. We use this to define the type of *lifting problems with respect to* J.

Definition 7.1 (`morcls_lp`)**.** *Given a morphism class J and a morphism f in $\mathcal{C}$, we denote the type of (g, f)-lifting problems as g ranges over J by S_f.*

This means that any $x : S_f$ is a lifting problem of the form

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle f} \\ B & \longrightarrow & Y \end{array}$$

with $g \in J$. We denote by g_x the map in J corresponding to a lifting problem $x : S_f$, and by h_x and k_x the top and bottom map respectively.

```
Definition morcls_lp (J : morphism_class C) (f : arrow C) : UU :=
  ∑ (g : total_category (morcls_disp J)), (pr1 g) --> f.
```

Consider the diagram corresponding to the first step of the classical small object argument:

$$\begin{array}{ccc} \coprod_{x:S_f} A_x & \xrightarrow{\coprod_{x:S_f} h_x} & X \\ {\scriptstyle \coprod_{x:S_f} g_x}\downarrow & & \downarrow{\scriptstyle f} \\ \coprod_{x:S_f} B_x & \xrightarrow[\coprod_{x:S_f} k_x]{} & Y \end{array}$$

Like before, we take a pushout to obtain the following diagram.

$$\begin{array}{ccccc} \coprod_{x:S_f} A_x & \xrightarrow{\coprod_{x:S_f} h_x} & X & = & X \\ {\scriptstyle \coprod_{x:S_f} g_x}\downarrow & & {\scriptstyle \lambda^1_f}\downarrow & & \downarrow \\ \coprod_{x:S_f} B_x & \longrightarrow & E^1_f & \xrightarrow[\rho^1_f]{} & Y \end{array}$$

Now we write $R^1 : \mathcal{C}^{\mathbf{2}} \longrightarrow \mathcal{C}^{\mathbf{2}}$ for the functor sending f to ρ^1_f. We also define Λ^1 for the natural transformation $\mathrm{id}_{\mathcal{C}^{\mathbf{2}}} \Longrightarrow R^1$, with components at $f : X \to Y$

$$\begin{array}{ccc} X & \xrightarrow{\lambda^1_f} & E^1_f \\ {\scriptstyle \Lambda^1_f\ f}\downarrow & & \downarrow{\scriptstyle \rho^1_f} \\ Y & = & Y \end{array}$$

In a similar fashion, we define a pointed endofunctor (L^1, Φ^1), sending f to λ^1_f with the appropriate counit Φ^1. Our first goal will be to show that the copointed endofunctor (L^1, Φ^1) extends to a comonad, giving us an object of $\mathbf{LNWFS}_{\mathcal{C}}$. The pointed endofunctor (R^1, Λ^1) does *not*, in general, extend to a monad though, explaining why we do not yet obtain an object of $\mathbf{NWFS}_{\mathcal{C}}$.

To relate this back to the classical small object argument, the fact that (L^1, Φ^1) extends to a comonad corresponds with the fact that at every step in the transfinite construction of the classical small object argument, the left map is in J-cell. After all, the comultiplication witnesses that λ^1_f lifts against $\rho^1_{\lambda^1_f}$ for any morphism f. The absence of a monad multiplication means that ρ^1_f need not be an R^1-**Map**, similar to how the right map we obtain after just any step in the classical small object argument need not be a fibration.

Firstly though, we show that the objects we introduced are indeed functors and natural transformations. The first step is to turn the assignation $L^1 : f \mapsto \lambda^1_f$ into the object part of a functor. We need to define how the functor acts on morphisms.

In order to do this, we first define a functor

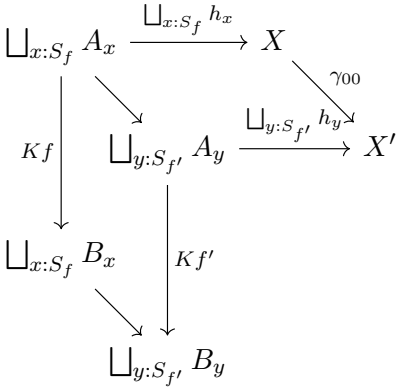
$$K : \mathcal{C}^{\mathbf{2}} \longrightarrow \mathcal{C}^{\mathbf{2}} : f \mapsto \coprod_{x:S_f} g_x.$$

We need to define what this functor does on morphisms. Note that given a morphism $\gamma : f \to f'$ in $\mathcal{C}^{\mathbf{2}}$, we can simply postcompose any lifting problem $x : S_f$ with γ to obtain a lifting problem in $S_{f'}$. We get a map $S_\gamma : S_f \to S_{f'}$, which in turn allows us to define K on γ as the inclusion of the coproduct

$$K\gamma := \coprod_{x:S_f} \mathtt{in}^{\coprod}_{S_\gamma(x)} : \coprod_{x:S_f} g_x \to \coprod_{y:S_{f'}} g_y,$$

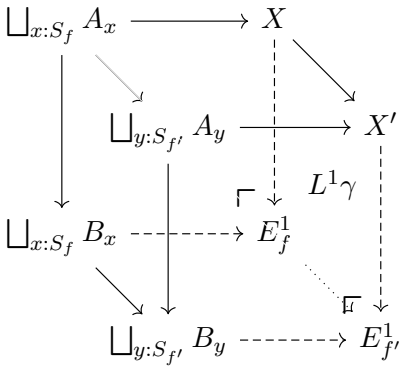
where $\mathtt{in}^{\coprod}_{(-)}$ denotes the inclusion into the coproduct of the object at the given index. Verifying that this is indeed a functor is fairly straightforward (`morcls_coprod_functor`).

The maps $\coprod_{x:S_f} h_x$ and $\coprod_{x:S_f} k_x$ become the components for a natural transformation $\phi : K \Longrightarrow \mathrm{id}_{\mathcal{C}^{\mathbf{2}}}$. We consider part of the naturality square of ϕ (which is a cube since it is a square in the arrow category)



where the left face is $K\gamma$. We push out the front face to obtain $Kf' \to \lambda^1_{f'}$, and the rear face yields $Kf \to \lambda^1_f$. We define $L^1\gamma$ to be the right-hand face, with the map on codomains induced by the universal property of the pushout. In the diagram below, the pushed out arrows are dashed, and the dotted arrow, the map $\left(L^1\gamma\right)_{11}$, is the map obtained using the

universal property of the pushout.



We can show that this construction is functorial.

Lemma 7.2 (`one_step_comonad_functor`)**.** *L^1 as defined above is a functor.*

This follows from the properties of pushouts. After all, identity must be the unique map making this construction commute when $\gamma = \mathrm{id}_f$ and the composition of the pushout maps must be the same as the unique map making the composition of two cubes commute. Indeed, we also have the following.

Lemma 7.3 (`one_step_monad_functor`)**.** *R^1 as defined above is a functor.*

Lemma 7.4 (`one_step_factorization`)**.** *L^1 and R^1 form a functorial factorization F^1, called the* one-step factorization*.*

Lemma 7.5 (`one_step_monad_unit`, `one_step_comonad_counit`)**.** *The objects Λ^1 and Φ^1 are in fact natural transformations.*

Example 7.6. *It may be useful to think about this 'commuting cube construction' intuitively. Thinking about it in terms of topological spaces, we are simply mapping the cells that are glued to one relative CW-complex to those in another, in the way defined by the map $S_\gamma : S_f \to S_{f'}$.*

Note that this construction immediately gives us a natural transformation

$$\xi : K \Longrightarrow L^1$$

where the components are the pushouts

$$\xi_f := \begin{array}{ccc} \bigsqcup_{x:S_f} A_x & \xrightarrow{\bigsqcup_{x:S_f} h_x} & X \\ {\scriptstyle \bigsqcup_{x:S_f} g_x}\downarrow & & \downarrow{\scriptstyle \lambda^1_f} \\ \bigsqcup_{x:S_f} B_x & \xrightarrow[\bigsqcup_{x:S_f} k_x]{} & E^1_f \end{array}$$

We get a copointed endofunctor (L^1, Φ^1), but we want to extend it to a comonad. To find the multiplication, we use the same construction as for $L^1\gamma$, with $f' = \lambda^1_f$ and a different map of lifting problems $S_f \to S_{\lambda^1_f}$. We get this map directly from the construction of λ^1_f, since for a lifting problem $x : (g_x \to f)$ in S_f with $g_x \in J$, we compose the inclusion of g_x into Kf with ξ_f to obtain a morphism

$$\begin{array}{ccccc} A_x & \xhookrightarrow{\mathtt{in}^{\sqcup}_x} & \bigsqcup_{x:S_f} A_x & \longrightarrow & X \\ {\scriptstyle g_x}\downarrow & & {\scriptstyle \bigsqcup_{x:S_f} g_x}\downarrow & & \downarrow{\scriptstyle \lambda^1_f} \\ B_x & \xhookrightarrow[\mathtt{in}^{\sqcup}_x]{} & \bigsqcup_{x:S_f} B_x & \longrightarrow & E^1_f \end{array}$$

Since the rest of the construction is the same as before, functoriality follows. It turns out that, indeed

Lemma 7.7 (`one_step_comonad`, `one_step_comonad_as_LNWFS`)**.** *The comultiplication Σ^1 obtained from this construction indeed extends (L^1, Φ^1) to a comonad, giving us an object of $\mathbf{LNWFS}_{\mathcal{C}}$, lying over the one-step factorization F^1.*

Example 7.8. *Before we continue, let us think about the comultiplication intuitively. In the* **TOP** *and* **SSET** *examples, the maps $g_x : A_x \to B_x$ are boundary inclusions of spheres into disks, making the pushout of the coproduct the same as gluing all possible cells to the domain X of f. The map from S_f to $S_{\lambda^1_f}$ effectively tells us that if we can factor f through a cell attachment on X, then we can factor λ^1_f through the same cell attachment. This seems sensible, as λ^1_f is defined precisely as this cell attachment. The comultiplication map $\lambda^1_f \to \lambda^1_{\lambda^1_f}$, together with the comonad axioms then tell us that this factoring of λ^1_f adds no new information, as everything can be collapsed back into the codomain of λ^1_f.*

Example 7.9. *In the category* **SET***, where we chose $J = \{ \emptyset \to * \}$, the one-step factorization is in fact already the factorization [22, Examples 34]*

$$X \xrightarrow{f} Y \mapsto X \xrightarrow{\mathtt{in}^{\sqcup}_X} X \sqcup Y \xrightarrow{f \sqcup \mathrm{id}_Y} Y.$$

For this choice of J and any $f : X \to Y$, the possible lifting problems can be indexed by Y, as there is precisely one map $* \to Y$ for every $y : Y$. This yields the following pushout:

$$\begin{array}{ccccc} \emptyset & \longrightarrow & X & = & X \\ \downarrow & & \downarrow{\scriptstyle \lambda^1_f} & & \downarrow{\scriptstyle f} \\ \bigsqcup_{y:Y} * & \xrightarrow[\sqcup_{y:Y} \mathrm{const}_y]{} & E^1_f & \xrightarrow[\rho^1_f]{} & Y \end{array}$$

Indeed, since $\bigsqcup y : Y * \cong Y$, we get that $E^1_f = X \sqcup Y$, and the factorizations indeed coincide.

7.2 The Iterative Step

Now that we have found an object in $\mathbf{LNWFS}_{\mathcal{C}}$, we want to 'fix' the right part using a transfinite construction. We can use a transfinite construction defined on general *monoidal categories* to find a monoid in the category. We will define a monoidal structure on $\mathbf{Ff}_{\mathcal{C}}$, which can be extended to $\mathbf{LNWFS}_{\mathcal{C}}$. We define it in such a way, that a monoid F in this monoidal structure corresponds with an object of $\mathbf{RNWFS}_{\mathcal{C}}$ over F, or, when lifted to $\mathbf{LNWFS}_{\mathcal{C}}$, a monoid L over $F : \mathbf{Ff}_{\mathcal{C}}$ corresponds to an object of $\mathbf{NWFS}_{\mathcal{C}}$ over F. In other words, the monoid structure gives us the extension of the right functor of the underlying functorial factorization to a monad. Afterwards, we define the transfinite construction on a general monoidal category $\mathcal{V}$, using some *smallness assumptions*, which we will show for $\mathbf{LNWFS}_{\mathcal{C}}$ in the subsequent sections, completing the argument.

Firstly though, we briefly introduce the theory of monoidal categories, following the definitions found in `UniMath` [12] [13]. These definitions are similar to those found in more classical literature [23], but are adapted to be more fit for formalization. This notion is less strict in the sense that equalities in the traditional definition are replaced by (natural) isomorphisms.

In their articles, Ahrens, Matthes and Wullaert describe the need for this adapted definition, partly relating to a common theme in this thesis: propositional versus definitional equalities. For example, in the category of endofunctors on a category $\mathcal{C}$, one might like to use that $\mathrm{id}_{\mathcal{C}} \cdot F = F$ for any endofunctor F. These objects are *not* convertible in formalization though, making this equality propositional and not definitional. The adapted notion of monoidal categories works around this issue, providing a simple and performant way to work

with monoidal categories in formalization. It uses isomorphisms in places where existing definitions use equalities, making it a less strict notion than its counterpart. This more modern notion of a monoidal structure will prove to be sufficient for Garner's algebraic small object argument.

7.2.1 Monoidal Categories

A monoidal structure on a category defines a genercal notion of a product on the category, like the direct product $\times$, the direct sum $\oplus$ or the tensor product $\otimes$. In Ahrens, Matthes and Wullaert's adapted notion of monoidal categories, this product comes in the form of a *whiskered bifunctor*.

Definition 7.10 (bifunctor)**.** *A* whiskered bifunctor *on a category $\mathcal{V}$ is a bifunctor $\otimes : \mathcal{V} \times \mathcal{V} \longrightarrow \mathcal{V}$, such that $A \otimes (-)$ and $(-) \otimes X$ are endofunctors for any $A, X : \mathcal{V}$, such that*

$$(A \otimes f) \cdot (g \otimes Y) = (g \otimes X) \cdot (B \otimes f)$$

for any $A, B, X, Y : \mathcal{V}$, $f : X \to Y$ and $g : A \to B$. We refer to this identity as the commutativity of whiskering.

Remark. *The endofunctor requirement on $A \otimes (-)$ and $(-) \otimes X$ means they provide the following* whiskering data*:*

- *For any $A : \mathcal{V}$ and $f : X \to Y$ in $\mathcal{V}$, a morphism*

  $$A \otimes f : A \otimes X \to A \otimes Y.$$

  This is called leftwhiskering*.*
- *For any $g : A \to B$ in $\mathcal{V}$ and $X : \mathcal{V}$, a morphism*

  $$g \otimes X : A \otimes X \to B \otimes X.$$

  This is called rightwhiskering*.*

with the following compatibility with identity and composition in $\mathcal{V}$:

$$\begin{aligned} A \otimes \mathrm{id}_X &= \mathrm{id}_{A \otimes X} & A \otimes (f \cdot f') &= (A \otimes f) \cdot (A \otimes f') \\ \mathrm{id}_A \otimes X &= \mathrm{id}_{A \otimes X} & (g \cdot g') \otimes X &= (g \otimes X) \cdot (g' \otimes X) \end{aligned}$$

for objects A, X and compatible morphisms f, f', g, g'.

Example 7.11 (monendocat_tensor)**.** *Functor composition in endofunctor categories can be extended to form a whiskered bifunctor. Whiskering for this whiskered bifunctor corresponds with the notion of whiskering briefly mentioned in Definition 5.10.*

Definition 7.12 (monoidal)**.** *A* monoidal structure *on a category $\mathcal{V}$ consists of a quintuple $(\otimes, I, \mathtt{lu}, \mathtt{ru}, \alpha)$ consisting of the following data.*

- *A whiskered bifunctor $\otimes : \mathcal{V} \times \mathcal{V} \longrightarrow \mathcal{V}$, referred to as the* tensor product*.*
- *A* unit *$I : \mathcal{V}$.*
- *A* left unitor `lu`*, which is a natural isomorphism $I \otimes (-) \Longrightarrow \mathrm{id}_{\mathcal{V}}$, i.e. a family of isomorphisms*

  $$\mathtt{lu}\ X : I \otimes X \to X$$

  for $X : \mathcal{V}$, such that

  $$(I \otimes f) \cdot \mathtt{lu}\ Y = (\mathtt{lu}\ X) \cdot f$$

  for any $f : X \to Y$. We denote the inverses by `luinv`*. That is to say,*

  $$\mathtt{luinv}\ X : X \to I \otimes X$$

  is an inverse to `lu` X*.*

- *A* right unitor `ru`*, which is a natural ismorphism* $(-) \otimes I \Longrightarrow \mathrm{id}_{\mathcal{V}}$*. We denote the inverse by* `ruinv`*.*
- *An* associator α*, which is a natural isomorphism* $((-)\otimes(-))\otimes(-) \Longrightarrow (-)\otimes((-)\otimes(-))$*. We denote the inverse by* α`inv`*.*

such that the triangle identity *holds*

$$\begin{array}{ccc} (X \otimes I) \otimes Y & \xrightarrow{\quad\alpha\quad} & X \otimes (I \otimes Y) \\ & {\scriptstyle (\mathtt{ru}\ X)\otimes Y} \searrow \quad \swarrow {\scriptstyle X\otimes(\mathtt{lu}\ Y)} & \\ & X \otimes Y & \end{array}$$

and the pentagon identity *holds*

$$\begin{array}{ccccc} ((W \otimes X) \otimes Y) \otimes Z & \xrightarrow{\alpha\otimes Z} & (W \otimes (X \otimes Y)) \otimes Z & \xrightarrow{\quad\alpha\quad} & W \otimes ((X \otimes Y) \otimes Z) \\ & {\scriptstyle \alpha} \searrow & & \swarrow {\scriptstyle W\otimes\alpha} & \\ & (W \otimes X) \otimes (Y \otimes Z) & \xrightarrow[\alpha]{\qquad} & W \otimes (X \otimes (Y \otimes Z)) & \end{array}$$

for appropriate instances of α*. We say that* $\mathcal{V}$ *is a* monoidal category*.*

Example 7.13 (monendocat_monoidal)**.** *One can define a monoidal structure on any endofunctor category, where the tensor product is functor composition, together with appropriate whiskering data.*

Finally, we introduce the notion of *monoid* in this adapted version of monoidal categories.

Definition 7.14 (monoid)**.** *A* monoid *in a monoidal category* $\mathcal{V}$ *is an object* $M : \mathcal{V}$ *together with a map* $\mu : M \otimes M \to M$*, reffered to as the* multiplication *and a map* $\eta : I \to M$*, referred to as the* unit*, such that the following diagrams commute*

$$\begin{array}{ccccc} I \otimes M & \xrightarrow{\eta\otimes M} & M \otimes M & \xleftarrow{M\otimes\eta} & M \otimes I \\ & {\scriptstyle \mathtt{lu}\ M} \searrow & \downarrow{\scriptstyle \mu} & \swarrow {\scriptstyle \mathtt{ru}\ M} & \\ & & M & & \end{array} \qquad \begin{array}{ccccc} (M \otimes M) \otimes M & \xrightarrow{\alpha} & M \otimes (M \otimes M) & \xrightarrow{M\otimes\mu} & M \otimes M \\ {\scriptstyle \mu\otimes M}\downarrow & & & & \downarrow{\scriptstyle \mu} \\ M \otimes M & & \xrightarrow[\mu]{\qquad\qquad} & & M \end{array}$$

Example 7.15 (monoid_to_monad_CAT, monad_to_monoid_CAT)**.** *Indeed, "a monad is just a monoid in the category of endofunctors" [23].*

Remark. *As will be remarked a few times, we often leave out unitors and associators for readability, though they must be present in a formal proof. Because of their properties, they are usually simple, yet cumbersome to deal with. We will henceforth denote a monoidal structure only by its multiplication and unit, i.e. as* $(\otimes, I)$*.*

Dually, one may define the notion of a *comonoidal structure* as a monoidal structure in the opposite category.

7.2.2 The Monoidal Structure on $\mathbf{Ff}_{\mathcal{C}}$

To reiterate, the next step is to define a monoidal structure on $\mathbf{Ff}_{\mathcal{C}}$, such that a monoid corresponds with an object in $\mathbf{RNWFS}_{\mathcal{C}}$. We will lift this monoidal structure to one on $\mathbf{LNWFS}_{\mathcal{C}}$, so that a monoid in $\mathbf{LNWFS}_{\mathcal{C}}$ corresponds with an object of $\mathbf{NWFS}_{\mathcal{C}}$. We define the unit of the monoidal structure on $\mathbf{Ff}_{\mathcal{C}}$ to be the functorial factorization I (`Ff_lcomp_unit`) sending

$$X \xrightarrow{f} Y \quad \mapsto \quad X \xrightarrow{\mathrm{id}_X} X \xrightarrow{f} Y.$$

For two functorial factorizations F, F', we define their tensor product $F' \otimes F$ to be

$$X \xrightarrow{f} Y \quad \mapsto \quad X \xrightarrow{\lambda_f \cdot \lambda'_{\rho_f}} K'_{\rho_f} \xrightarrow{\rho'_{\rho_f}} Y.$$

To phrase it differently, $F' \otimes F$ factors f with F, and then factors the resulting right map again using F'.

Lemma 7.16 (`Ff_monoidal`)**.** *The pair $(\otimes, I)$ defines a monoidal structure on $\mathbf{Ff}_\mathcal{C}$.*

It is easy to define (though a lot of work to formalize) the required natural isomorphisms that turn $\mathbf{Ff}_\mathcal{C}$ into a monoidal category (the left and right unitors `Ff_l_id_left`, `Ff_l_id_right` and the associator `Ff_l_assoc` and their inverses). This is because for all these transformations, the relevant data, being the middle morphism of the components of the natural transformations, is always identity. Since we usually do not care about what *exactly* the middle object is, let us use denote it with a $\square$, simply to make it clear which object exactly is the middle object. For example, consider the right unitor $F \otimes I \to F$:

$$\begin{array}{ccc}
X & = & X \\
{\scriptstyle \mathrm{id}_X}\downarrow & & \downarrow{\scriptstyle \lambda_f} \\
{\scriptstyle \lambda_f}\downarrow & & \downarrow \\
\square & = & \square \\
{\scriptstyle \rho_f}\downarrow & & \downarrow{\scriptstyle \rho_f} \\
Y & = & Y
\end{array}$$

We also define the whiskering of morphisms (`Ff_l_leftwhisker`, `Ff_l_rightwhisker`). Given functorial factorizations F, G, G' and a morphism $\tau : G \to G'$, we define $\tau \otimes F : G \otimes F \to G' \otimes F$ to be

$$\begin{array}{ccc}
X & = & X \\
{\scriptstyle \lambda^F_f}\downarrow & = & \downarrow{\scriptstyle \lambda^F_f} \\
{\scriptstyle \lambda_{\rho^F_f}}\downarrow & & \downarrow{\scriptstyle \lambda'_{\rho^F_f}} \\
\square & \xrightarrow{\tau_{\rho^F_f}} & \square' \\
{\scriptstyle \rho_{\rho^F_f}}\downarrow & & \downarrow{\scriptstyle \rho'_{\rho^F_f}} \\
Y & = & Y
\end{array}$$

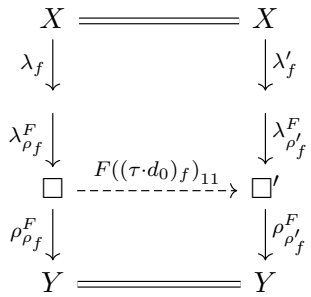
Leftwhiskering, so $F \otimes \tau : F \otimes G \to F \otimes G'$ is a bit more complex. Consider $F((\tau_r)_f)$, i.e. F applied to the component of the induced natural transformation $\tau_r : R_G \Longrightarrow R'_G$ at f

$$(\tau_r)_f := \begin{array}{ccc}
E_f & \xrightarrow{(\tau_f)_{11}} & E'_f \\
{\scriptstyle \rho_f}\downarrow & & \downarrow{\scriptstyle \rho'_f} \\
Y & = & Y
\end{array}$$

giving us

$$\begin{array}{ccc}
E_f & \longrightarrow & E'_f \\
{\scriptstyle \lambda^F_{\rho_f}}\downarrow & & \downarrow{\scriptstyle \lambda^F_{\rho'_f}} \\
\square & \xrightarrow{F((\tau \cdot d_0)_f)_{11}} & \square' \\
{\scriptstyle \rho^F_{\rho_f}}\downarrow & & \downarrow{\scriptstyle \rho^F_{\rho'_f}} \\
Y & = & Y
\end{array}$$

which is precisely the morphism we need, to give us

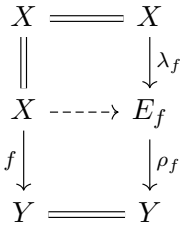


Lemma 7.17 (`Ff_monoidal`)**.** *These morphisms define a monoidal structure* $(\otimes, I)$ *on* $\mathbf{Ff}_\mathcal{C}$*.*

As defined in Definition 7.14, a monoid in $\mathbf{Ff}_\mathcal{C}$ is an object F with a unit $\eta : I \to F$ and a multiplication $\mu : F \otimes F \to F$, such that the appropriate diagrams commute. In fact, those diagrams look the same as those that should commute for a monad. After all, "a monad is just a monoid in the category of endofunctors" [23]. One important thing to note is that we should verify that the unit for the monad and the monoid correspond. This follows from the fact that I is initial.

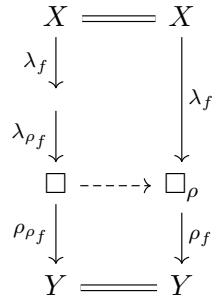
Lemma 7.18. `Ff_l_point`, `Ff_point_unique` *The monoidal unit* I *is initial in* $\mathbf{Ff}_\mathcal{C}$*.*

Proof. We can simply look at what the diagram for a morphism $I \to F$ should look like for any F in $\mathbf{Ff}_\mathcal{C}$:



By commutativity of the top square, there is precisely one map to make this diagram commute. □

The diagram for the monoid multiplication μ looks like



The bottom square combined with the monoid axioms that μ satisfies allow us to extend the right functor of F to a monad. And so, we conclude the following:

Lemma 7.19 (`Ff_monoid_is_RNWFS`)**.** *A monoid* F *in* $\mathbf{Ff}_\mathcal{C}$ *corresponds with an object of* $\mathbf{RNWFS}_\mathcal{C}$*, lying over* F*.*

In fact, the converse also holds [9], but this is not relevant for us.

Example 7.20. *Before we show that this monoidal structure lifts to one in* $\mathbf{LNWFS}_{\mathcal{C}}$*, let us think about what this tensor product does a bit more intuitively. We again consider the example of topological spaces, and think about left tensoring with* L^1*, which sends a map* $f : X \to Y$ *to a factorization*

$$X \xrightarrow{\lambda^1_f} E^1_f \xrightarrow{\rho^1_f} Y.$$

This is an operation that effectively glues (all possible) cells to X *to obtain a relative CW-complex* E^1_f*, and then tells us how to factor* f *through it.*

Tensoring L^1 *with itself factors the right map with* L^1 *again, glueing more cells to the middle object* E^1_f*, and factoring the right map through this new relative CW-complex.*

$$X \xrightarrow{\lambda^1_f} E^1_f \xrightarrow{\lambda^1_{\rho^1_f}} E^1_{\rho^1_f} \xrightarrow{\rho^1_{\rho^1_f}} Y.$$

Of course, since E^1_f *already had all possible cells on* X *glued to it, these cells are now in* $E^1_{\rho^1_f}$ *twice. The transfinite construction we will define aims to correct for this, by collapsing those cells into each other with coequalizers.*

Example 7.21. *Let us also consider what tensoring does for the one-step factorization*

$$X \xrightarrow{f} Y \mapsto X \xrightarrow{\mathtt{in}^{\sqcup}_X} X \sqcup Y \xrightarrow{f \sqcup \mathrm{id}_Y} Y$$

in **SET**. *Since the middle object is of such a simple structure, we can write out that* $L^1 \otimes L^1$ *gives us the factorization*

$$X \xrightarrow{f} Y \mapsto X \xrightarrow{\mathtt{in}^{\sqcup}_X} X \sqcup Y \xrightarrow{\mathtt{in}^{\sqcup}_{X \sqcup Y}} (X \sqcup Y) \sqcup Y \xrightarrow{(f \sqcup \mathrm{id}_Y) \sqcup \mathrm{id}_Y} Y.$$

Indeed, the tensor product effectively adds no new data. As will be clear from the application of the tensor product in the transfinite construction, this is to be expected, as the one-step factorization already admits an NWFS (or monoid) structure.

7.2.3 The Monoidal Structure on $\mathbf{LNWFS}_{\mathcal{C}}$

We want to extend this monoidal structure to $\mathbf{LNWFS}_{\mathcal{C}}$, making it so that a monoid in $\mathbf{LNWFS}_{\mathcal{C}}$ gives us a full NWFS. After all, an NWFS is just a LNWFS and a RNWFS over the same functorial factorization. There is already some theory in `UniMath` that allows us to define a 'displayed monoidal structure' on $\mathbf{LNWFS}_{\mathcal{C}}$ [12], saving us some work. After all, we want the monoidal unit to lie over I, and given LNWFSs L and L' over F and F' respectively, we want $L' \otimes L$ to lie over $F' \otimes F$.

Lemma 7.22 (`LNWFS_tot_monoidal`)**.** *Given* $L, L' : \mathbf{LNWFS}_{\mathcal{C}}$ *over* $F, F' : \mathbf{Ff}_{\mathcal{C}}$ *respectively, there is an LNWFS structure on* $F \otimes F'$*. In addition, there is also an LNWFS structure on* I*, such that* $(\otimes, I)$ *lifts to a monoidal structure on* $\mathbf{LNWFS}_{\mathcal{C}}$*.*

Using the displayed monoidal category construction we are left to show the following three things.

- There is a comonad structure on the left functor of I (`LNWFS_tot_lcomp_unit`). This is simple, since the left functor of I simply maps an arrow f to identity on its domain. In the comonad structure, all relevant maps are also identity.
- The relevant monoidal operations preserve comonad structures (i.e. the left and right unitors, the associator and left- and rightwhiskering). Though simple on paper, this requires some work in a formalization (`LNWFS_tot_l_id_left`, `LNWFS_tot_l_id_right`,

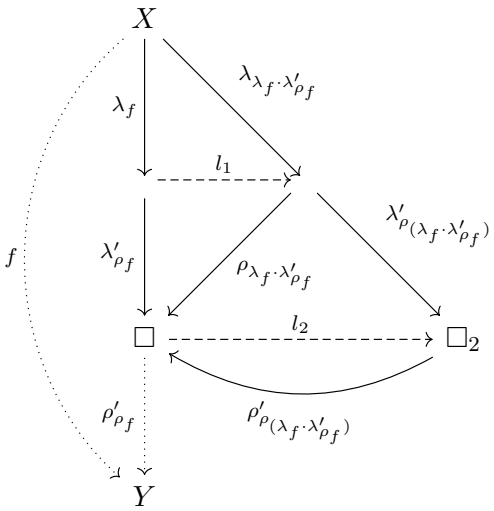
`LNWFS_tot_l_assoc`, `LNWFS_tot_l_rightwhisker`, `LNWFS_tot_l_leftwhisker`). For the unitors and associator, the relevant morphisms (so the middle morphisms of the natural transformations) are all identity. Showing that left- and rightwhiskering with an LNWFS preserves the comonad morphism axioms can be done using naturality.

- There is a comonad structure on the left functor of the tensor product $F' \otimes F$ for any $F, F' : \mathbf{Ff}_{\mathcal{C}}$. We elaborate on this point.

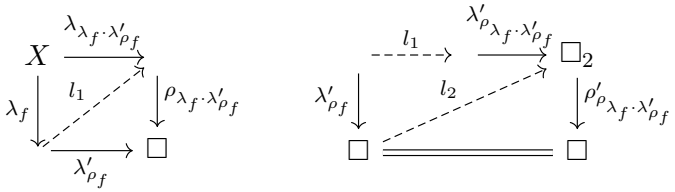
Let us denote the comultiplication on $L_{F' \otimes F}$ by $\Sigma^{\otimes}$. Firstly, let us consider the diagram we are trying to fill

$$\Sigma^{\otimes}_f = \quad \begin{array}{ccc} & L_{F' \otimes F} f & \qquad L^2_{F' \otimes F} f \\ X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda_f \cdot \lambda'_{\rho_f}}} \\ {\scriptstyle \lambda'_{\rho_f}}\downarrow & & \downarrow{\scriptstyle \lambda'_{\rho_{\lambda_f \cdot \lambda'_{\rho_f}}}} \\ \square & \xrightarrow[\sigma^{\otimes}_f]{} & \square_2 \end{array}$$

It is important to note here, that we are composing the *left functor of* $F' \otimes F$ with itself, and we are *not* tensoring $F' \otimes F$ with itself, and then taking the left functor. In order to better understand what is going on, it might be useful to think of the diagram in a different way, where we apply the construction used to define the tensor product $F' \otimes F$ twice, once to f, and once to the resulting left map. To reiterate this process for an arrow f, we first factor f with F, and then factor the right map with F', where the resulting left functor sends f to the composite of the two left maps. The resulting diagram becomes

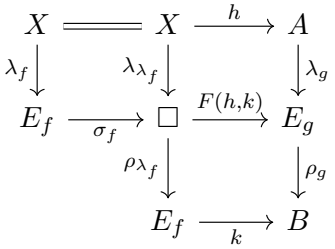


Now, the dotted lines do not really matter, we just left them there for completeness, but the dashed lines are very important. Note that the domain and codomain of l_2 are the ones we are looking for in $\sigma^{\otimes}_f$. We can think of l_1 and l_2 as fillers of the following lifting problems:

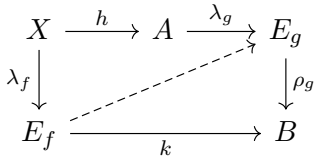


And so, in order to define l_1 and l_2, we can try to fill these lifting problems. Obviously, we are considering LNWFSs, so we do not have all the data for a full NWFS. We do not

'know what a right map' is. Furthermore, we only really know that maps of the form $\lambda_{(...)}$ or $\lambda'_{(...)}$ are left maps for F or F' respectively, recalling Lemma 5.22. We adapt the diagram we drew to find a lift between an **L-Map** and an **R-Map** in Lemma 5.21 so that it only contains things we 'know', given arrows f and g, for the LNWFS L. A similar diagram may be drawn for L'.



We can read off that the middle arrow still gives a lift for any lifting problem of the form



or the analogue for L'. Both of our lifting problems are of this form. Now note that all the operations done to obtain the lifts l_1 and l_2 are natural, allowing one to show that our candidate comultiplication is indeed a natural transformation. In fact, we can show the following.

Lemma 7.23 (`LNWFS_lcomp_comul_axioms`, `LNWFS_lcomp_comul_monad_laws`)**.** *The 'candidate comultiplication' on $L_{F'\otimes F}$ defined above is a natural transformation, and gives rise to a comonad structure on $L_{F'\otimes F}$.*

And so, we have lifted the monoidal structure on $\mathbf{Ff}_{\mathcal{C}}$ to one on $\mathbf{LNWFS}_{\mathcal{C}}$. Just to give an idea of how much work a formalization may be compared to the arguments on paper: Garner leaves out the fact that the monoidal structure on $\mathbf{Ff}_{\mathcal{C}}$ lifts to one on $\mathbf{LNWFS}_{\mathcal{C}}$, though it took almost 1000 lines of formalization to show this.

Given a monoid L in $\mathbf{LNWFS}_{\mathcal{C}}$ over a functorial factorization F, we obtain a monoid structure on F in $\mathbf{Ff}_{\mathcal{C}}$ by forgetting the $\mathbf{LNWFS}_{\mathcal{C}}$ data. The monoid in $\mathbf{Ff}_{\mathcal{C}}$ will still yield an object of $\mathbf{RNWFS}_{\mathcal{C}}$ lying over F. Together with the original monoid L over F, we obtain the two monad structures we need to define an NWFS over F. In short, we have the following:

Lemma 7.24 (`LNWFS_tot_monoid_is_NWFS`)**.** *Let L be a monoid in $\mathbf{LNWFS}_{\mathcal{C}}$ over an object F in $\mathbf{Ff}_{\mathcal{C}}$. Then F is a monoid in $\mathbf{Ff}_{\mathcal{C}}$, and L gives an object of $\mathbf{NWFS}_{\mathcal{C}}$ lying over F.*

And so, in order to generate an NWFS, our next goal becomes to find a monoid in $\mathbf{LNWFS}_{\mathcal{C}}$. We do this using a transfinite construction, which Garner generalized from an article by Kelly [25]. We go over the construction in detail, providing intuitive examples and redefining how the monoid is obtained. We show that the construction holds, assuming certain 'smallness requirements'. We show that these smallness requirements hold in our situation after introducing the construction.

7.2.4 The Transfinite Sequence

As mentioned, Garner generalized a transfinite construction by Kelly [25] to allow one to generate a monoid in a monoidal category on a specific object, given certain smallness requirements on this object. The construction defines a sequence on arbitrary ordinals, converging at some limit ordinal. Since the theory of ordinals has not been developed very much

in `UniMath`, we will only be limiting ourselves to the case where the sequence converges at ω, i.e. the first limit ordinal. This should still be sufficient for the construction to work for interesting examples, such as that of topological spaces or simplicial sets [2].

For this section, we assume $\mathcal{V}$ to be a monoidal category that has all connected colimits and T to be a pointed object in $\mathcal{V}$, with point $t : I \to T$.

```
Definition pointed : UU :=
    ∑ T, I_{V} --> T.
```

Througout the construction, we assume certain properties on $\mathcal{V}$ or T. These are the assumptions we need to show on $\mathbf{LNWFS}_{\mathcal{C}}$ and L^1 in order for this construction to work in our situation. The assumptions are the following.

V1 The monoidal category $\mathcal{V}$ has colimits of chains and coequalizers. This is a slightly weaker assumption than being cocomplete, which would of course also be sufficient.

V2 The rightwhiskering functor $(-) \otimes A$ preserves all colimits in $\mathcal{V}$ for any $A : \mathcal{V}$. In particular, it preserves colimits of chains and coequalizers. In other words, $\mathcal{V}$ is *right-closed*.

V3 The leftwhiskering functor $T \otimes (-)$ preserves colimits of chains in $\mathcal{V}$.

For any $A : \mathcal{V}$, we will define a transfinite sequence that will give us the *free T-algebra on A*, given that T satisfies the *smallness assumption* **V3** and that $\mathcal{V}$ satisfies assumptions **V1** and **V2**.

Definition 7.25 (`Mon_alg_data`)**.** *Given a pointed object (T, t) of $\mathcal{V}$, a* T-algebra *is a pair $(A, a : T \otimes A \to A)$ satisfying*

$$(t \otimes A) \cdot a = \mathrm{id}_A \,.$$

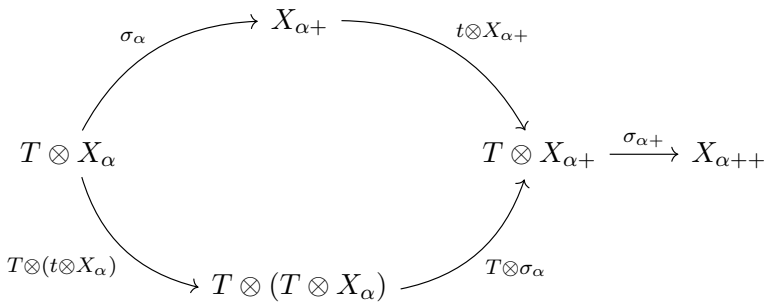
We say that a pair (A, a) is a T-algebra on A.

```
Definition Mon_alg_data (T : pointed) (A : C) : UU :=
    ∑ (a : T ⊗_{V} A --> A),
      (luinv_{V} _) · ((pointed_pt T) ⊗^{V}_{r} A) · a  = identity _.
```

We will not look into what it means exactly for a T-algebra to be free, since all we care about is constructing a monoid.

When applying the construction we are about to define, we will choose $\mathcal{V}$ to be $\mathbf{LNWFS}_{\mathcal{C}}$, and T to be L^1 with the obvious, unique point. It is easier to define this sequence on an abstract monoidal category, especially in formalization, but it may be useful to keep this example in mind.

Given $X_0 := A, X_1 := T \otimes A : \mathcal{V}$ and $\sigma_0 := \mathrm{id}_{T \otimes A} : T \otimes X_0 \to X_1$, we define a transfinite sequence, also called the *free T-algebra sequence for A*, inductively. For a successor ordinal $\alpha+ := \alpha + 1$ we define $X_{\alpha++}$ and $\sigma_{\alpha+} : T \otimes X_{\alpha+} \to X_{\alpha++}$ as the following coequalizer:



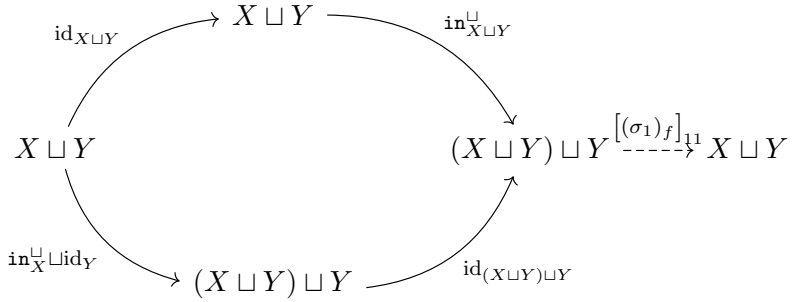
and for any step α, we define $x_\alpha : X_\alpha \to X_{\alpha+}$ to be $(t \otimes X) \cdot \sigma_\alpha$. Notice that the bottom arrow of the coequalizer diagram is just $T \otimes x_\alpha$.

Remark. *Whenever we write something like* $X_\alpha \xrightarrow{t\otimes X_\alpha} T \otimes X_\alpha$*, we actually leave out some morphisms. After all, the morphism* $t \otimes X_\alpha$ *is actually of type* $I \otimes X_\alpha \to T \otimes X_\alpha$*. In our formalization, we actually define this map as* $(\texttt{luinv}_{\mathcal{V}}\ X_\alpha)\cdot(t \otimes X_\alpha)$*.*

Remark. *The transfinite sequence can be defined on any* A *in* $\mathcal{V}$*, but generally we only care about the case where* $A = I$*. Garner defines the sequence more generally to construct an adjunction yielding a monoid. The way this adjunction is obtained turned out to be hard to formalize. We take a more direct approach and construct the monoid directly from the transfinite sequence on* I*. Therefore, the reader may keep in mind that we will be using* $A = I$ *for the rest of this section, unless specified. In the case where* $A = I$*, we call the sequence the* free monoid sequence*, as it will allow us to construct a monoid.*

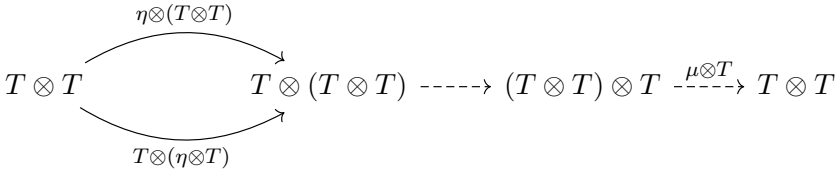
Example 7.26. *Before we continue, let us first ponder a bit on what this diagram means. Suppose* $\mathcal{V}$ *is the category of topological spaces, and our object* T *is the one-step comonad* L^1*. Then* X_α *is the result of a repeated process of gluing cells to the domain of* ρ^A_f*, where duplicate cells are removed. The maps* σ_α *are continuous maps that collapse any new, duplicate cells to the ones that were attached before. Inductively, we can show this intuition to be consistent. Indeed, in the first step,* $\sigma_0 : T \otimes A \to T \otimes A$ *does not collapse anything, since there are no duplicate cells to collapse yet. For any successive step,* $x_\alpha := (t \otimes X)\cdot\sigma_\alpha$ *defines how we glue cells to* X_α*, and then collapse any cells that we have already attached before, including* X_α *into* $X_{\alpha+}$*. In this way, the bottom arrow of the coequalizer basically includes '*X_α *with cells attached' into '*$X_{\alpha+}$ *with cells attached', whereas the top morphism collapses the 'old cells' into* $X_{\alpha+}$ *and then includes it into '*$X_{\alpha+}$ *with cells attached'. The coequalizer arrow* $\sigma_{\alpha+}$ *then associates the two ways of adding cells to* X_α *twice, giving us* $X_{\alpha++}$*, which will have no duplicate cells.*

Example 7.27. *Let us also consider the pushout in the case of* $J = \{\, \emptyset \to * \,\}$ *in* **SET** *for the transfinite sequence on* I*. We consider what happens to the middle object of the factorizations applied to* $f : X \to Y$*. Recall that for* $X_0 := I$*, the middle object is just* X*, and for* $X_1 := L^1 \otimes I$*, the middle object is* $X \sqcup Y$*. The first map* σ_0 *acts as identity. The diagram on the middle objects in the coequalizer of the first step in the sequence becomes*



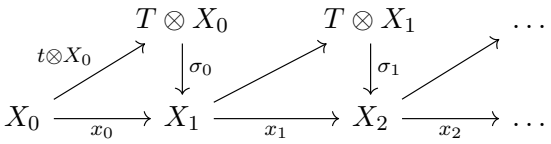
where the pushout σ_1 *just collapeses the two copies of* Y *into each other. Indeed, this sequence becomes a constant sequence of* L^1 *(up to isomorphism) right after the first step.*

Example 7.28. *In fact, one can show a more general statement. Consider the transfinite sequence on* $I : \mathcal{V}$*, and suppose that* T *is already a monoid* (T, η, μ)*, as is the case for* L^1 *from our example in* **SET**. *Assume that the point is the unit* $\eta : I \to T$*. In this case, let us 'follow the* T*s' in the coequalizer diagram of the first step:*



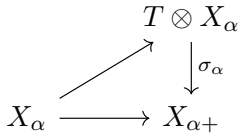
we see that the 'left T' gets mapped to either the leftmost or the middle T in the codomain of the arrows in the coequalizer. One can show, using the monoid axioms and the coequalizer properties, that the coequalizer simply becomes $T \otimes T$, and the coequalizer morphism (the dashed arrows) is the associator of $\mathcal{V}$ composed with $\mu \otimes T$, where $\mu : T \otimes T \to T$ is the monoid multiplication. Again, like the example in **SET**, *this sequence becomes a constant sequence after the 0th step.*

The full sequence looks something like



The sequence is defined inductively, but we need the previous *two* objects and the previous morphism to define what the next morphism and object are. In order for us to define this sequence properly in our formalization, with nice definitional equalities, we introduce some helper type, dubbed 'pair diagrams'.

Definition 7.29 (`pair_diagram`)**.** *A 'pair diagram', corresponding to step α in the sequence, is a diagram of the form*

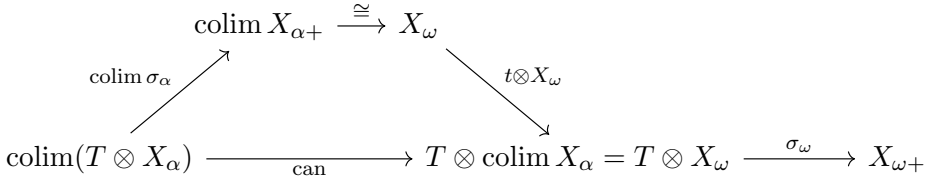


where the only real data are the objects X_α and $X_{\alpha+}$ as well as the morphism $\sigma_\alpha : T \otimes X_\alpha \to X_{\alpha+}$. The diagonal arrow is defined as $t \otimes X_\alpha$, and the horizontal arrow as $(t \otimes X_\alpha) \cdot \sigma_\alpha$. We refer to X_α as the left object *and $X_{\alpha+}$ as the* right object.

```
Local Definition pair_diagram : UU :=
    ∑ (x0 x1 : C), (T ⊗_{V} x0 --> x1).
```

And indeed, one can define the $(\alpha+1)$-th pair diagram using only the α-th pair diagram. The inductive definition allows us to make sure left object of the $(\alpha+1)$-th pair diagram is in fact *definitionally equal* to the right object of the α-th pair diagram. We can then easily define the objects of the chain by taking the left object for each pair diagram, and the horizontal arrows as the morphisms between them.

Garner and Kelly also define a 'limit ordinal step' in the sequence in their articles, similar to the classical small object argument. We follow their construction, but since we will only be considering chains, we will only define it for the first limit ordinal ω. The definition is the similar for any other limit ordinal. Assumption **V1** allows us to define $X_\omega := \operatorname{colim} X_\alpha$, and a map $\sigma_\omega : T \otimes X_\omega \to X_{\omega+}$ as the following coequalizer.




```
Definition free_monoid_coeq_sequence_limit_ordinal_step_on (A : C) :
    pair_diagram.
```

where can : $\operatorname{colim}(T \otimes X_\alpha) \to T \otimes \operatorname{colim} X_\alpha$ is the canonical map out of the colimit. Like the successor ordinal steps, we define $x_\omega : X_\omega \to X_{\omega+}$ to be $(t \otimes X_\omega) \cdot \sigma_\omega$. The only thing we will be using the limit ordinal step for, is to define when the sequence converges:

Definition 7.30 (`free_monoid_coeq_sequence_converges_on`)**.** *Let (T, t) be a pointed object of $\mathcal{V}$, and $A : \mathcal{V}$. We say that the free algebra sequence for A* converges at ω *if and only if the limit ordinal step*

$$x_\omega : X_\omega \to X_{\omega+}$$

is an isomorphism.

```
Definition free_monoid_coeq_sequence_converges_on (A : C) :=
    is_z_isomorphism (
      pair_diagram_horizontal_arrow
        (free_monoid_coeq_sequence_limit_ordinal_step_on A)
    ).
```

Let us go over the morphisms in the limit ordinal step.

- `can`**:** Firstly, there is the canonical morphism $\operatorname{colim}(T \otimes X_\alpha) \to T \otimes \operatorname{colim} X_\alpha$. This morphism is just the canonical morphism induced by the colimit, and is defined termwise as
$$T \otimes \mathtt{in}_\alpha^{\to} : T \otimes X_\alpha \to T \otimes \operatorname{colim} X_\alpha,$$
where $\mathtt{in}_\alpha^{\to}$ is the map from X_α into the colimit.

- `colim`σ`_on`**:** Secondly, there is the map $\operatorname{colim} \sigma_\alpha$. This map is defined simply as σ_α termwise, but let us verify that this colimit is indeed well-defined. For this we need any diagram
$$\begin{array}{ccc} T \otimes X_\alpha & \xrightarrow{T \otimes x_\alpha} & T \otimes X_{\alpha+} \\ {\scriptstyle \sigma_\alpha}\downarrow & & \downarrow{\scriptstyle \sigma_{\alpha+}} \\ X_{\alpha+} & \xrightarrow[x_{\alpha+}]{} & X_{\alpha++} \end{array}$$
to commute. This can be seen easily if we split the map $x_{\alpha+}$ back into the composite $t \otimes X_{\alpha+} \cdot \sigma_{\alpha+}$:
$$\begin{array}{ccccc} T \otimes X_\alpha & & \xrightarrow{T \otimes x_\alpha} & & T \otimes X_{\alpha+} \\ {\scriptstyle \sigma_\alpha}\downarrow & & & & \downarrow{\scriptstyle \sigma_{\alpha+}} \\ X_{\alpha+} & \xrightarrow[t \otimes X_{\alpha+}]{} & T \otimes X_{\alpha+} & \xrightarrow[\sigma_{\alpha+}]{} & X_{\alpha++} \end{array}$$
and we see that commutativity here follows from the commutativity of the coequalizer diagram.

- `chain_shift_left_colim_iso`**:** The isomorphism $\operatorname{colim} X_{\alpha+} \cong X_\omega$ is one that may be easily overlooked when writing classical mathematics, but in a formalization it definitely still takes some work to define this morphism (about 150 lines of work).

Now that we have defined the sequence, there are two things that we want to show. Firstly, we want to show that the sequence converges using the smallness requirement **V3** on our pointed object T. Secondly, we want to show that we can indeed construct a monoid in $\mathcal{V}$ when this sequence converges.

7.2.5 Convergence of the Sequence

Before we show when the sequence converges, we give an important definition for 'smallness', which will be used throughout the rest of the argument.

Definition 7.31. *We say that a functor $F : \mathcal{C} \longrightarrow \mathcal{C}'$ is* ω-small *if and only if it preserves colimits for any chain d in $\mathcal{C}$. In other words, if for any chain $d := \{ X_v \}_{v:\mathbb{N}}$ in $\mathcal{C}$, we have that*

$$\operatorname{colim} F X_v \cong F \operatorname{colim} X_v.$$

This is equivalent to saying that there is an inverse

$$F \operatorname{colim} X_v \to \operatorname{colim} F X_v$$

to the canonical map out of the colimit.

Lemma 7.32 (`T_preserves_diagram_impl_convergence_on`)**.** *The free T-algebra sequence for A converges for any $A : \mathcal{V}$ if the leftwhiskering functor $T \otimes (-)$ preserves the colimit.*

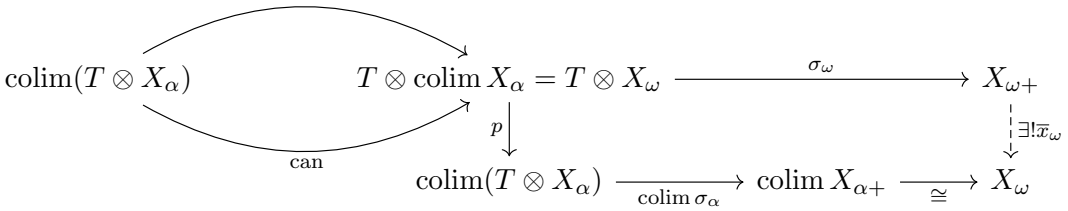
Assumption **V3** tells us precisely that $T \otimes (-)$ *is ω-small*. The following uses assumption **V3** to show convergence, following directly from the previous lemma.

Lemma 7.33 (`T_preserves_chains_impl_T_preserves_diagram_on`)**.** *The free T-algebra sequence for A converges for any $A : \mathcal{V}$ if the leftwhiskering functor $T \otimes (-)$ is ω-small.*

We show the first lemma. For this, we need to construct an inverse to the morphism x_ω, i.e. a morphism

$$\overline{x}_\omega : X_{\omega+} \to X_\omega,$$

given that $T \otimes (-)$ preserves the colimit of the sequence for A. We can construct this morphism using the properties of coequalizers. We get the unique morphism $\overline{x}_\omega$ if the solid lines commute in the following diagram.



where p is the inverse to can, which we get from the assumption that $T \otimes (-)$ preserves the colimit of the transfinite sequence. We can show the following.

Lemma 7.34 (`T_preserves_diagram_impl_convergence_on`)**.** *If $T \otimes (-)$ preserves the transfinite sequence, the above diagram commutes, and the unique arrow $\overline{x}_\omega$ exists. In particular, $\overline{x}_\omega$ is an inverse to x_ω.*

We now know the sequence converges on any A given that $T \otimes (-)$ preserves the colimit. We are however mostly interested in the sequence on the monoidal unit I. We call the limit for this sequence T^∞. One can show the following.

Lemma 7.35 (`free_monoid_coeq_sequence_converges_gives_adjoint_mon_alg`)**.** *If the free monoid sequence for I converges at ω, the limit T^∞ forms a T-algebra, with algebra morphism*

$$\tau^\infty := T \otimes T^\infty \xrightarrow{\sigma_\omega} X_{\omega+} \xrightarrow{\overline{x}_\omega} T^\infty.$$

7.2.6 Obtaining the Free Monoid

We now know when the sequence converges for any $A : \mathcal{V}$, and in particular, we have obtained a T-algebra T^∞ from the sequence on I. To reiterate, we used that $\mathcal{V}$ has all connected colimits (assumption **V1**) and that the leftwhiskering functor $T \otimes (-)$ is ω-small (assumption **V3**). In his articles, Garner uses an adjoint to the forgetful functor from the category of T-algebras to $\mathcal{V}$ to get a monoid. This argument turned out to be rather hard to formalize, so we abandoned the effort to do so. Instead, we define a way to find the monoid structure more directly, allowing for a much more intuitive construction. There is an obvious choice for the unit: the canonical inclusion into the colimit

$$\eta^\infty := \mathtt{in}_0^{\rightarrow} : I = X_0 \hookrightarrow T^\infty.$$

And so it remains to find a multiplication.

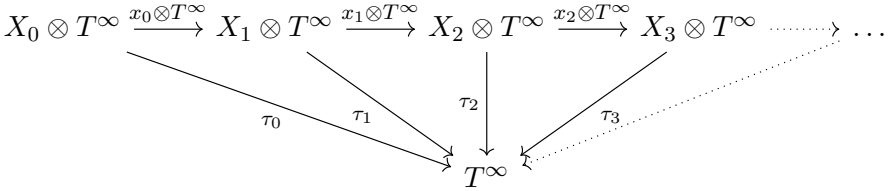
Example 7.36. *Let us again ponder a bit on what the involved objects really mean, if we think about them in the example of factorizations of topological spaces, where the tensor product with $T = L^1$ corresponds with gluing all possible cells to the middle object. In this case, the object T^∞ basically corresponds with infinite steps of glueing cells, and associating 'older, duplicate' ones from the previous step. The convergence of the sequence, and in particular the algebra map $\tau^\infty : T \otimes T^\infty \to T^\infty$ obtained from this fact, tells us that we can collapse* any *cells we glue to the middle object of T^∞ back down to cells that are already in T^∞. A multiplication $T^\infty \otimes T^\infty \to T^\infty$ would correspond to gluing (and collapsing duplicate) cells infinitely many times to T^∞, and then associating those with ones already in T^∞. Since we inductively define how we glue cells, it makes sense that we could also inductively collapse these glued cells back down to T^∞ as we go. This seems like a sensible idea, and something we should be able to do.*

Example 7.37. *The example of* **SET** *is much simpler. The sequence converges right away to L^1, as every step besides the '0th step' is just identity on L^1. The limit T^∞ will also just be L^1. The idea of inductive collapsing simply does the same thing every step in the sequence does: it collapses the two copies of Y in $(X \sqcup Y) \sqcup Y$ down to only one copy in the factorizations of a morphism $f : X \to Y$.*

By assumption **V2**, the rightwhiskering functor $(-) \otimes T^\infty$ preserves colimits of chains and coequalizers. In other words, we have

$$T^\infty \otimes T^\infty \cong \operatorname{colim}(X_\alpha \otimes T^\infty),$$

as well as the preservation of coequalizers. We can then define a morphism $T^\infty \otimes T^\infty \to T^\infty$ if we can define compatible morphisms $X_\alpha \otimes T^\infty \to T^\infty$. That is to say, we want to find a family of morphisms τ_α such that we have the following cocone.



In order to do this, we will show something slightly different first.

Lemma 7.38 (`free_monoid_coeq_sequence_on_Tinf_pd_Tinf_map`)**.** *There is a family of maps $\{ \tau_\alpha : X_\alpha \otimes T^\infty \to T^\infty \}$ such that the following diagram commutes for any α.*

$$\begin{array}{ccc} T \otimes X_\alpha \otimes T^\infty & \xrightarrow{\sigma_\alpha \otimes T^\infty} & X_{\alpha+} \otimes T^\infty \\ {\scriptstyle T \otimes \tau_\alpha}\downarrow & & \downarrow{\scriptstyle \tau_{\alpha+}} \\ T \otimes T^\infty & \xrightarrow[\tau^\infty]{} & T^\infty \end{array}$$

Remark. *The map $\sigma_\alpha \otimes T^\infty$ is indeed a coequalizer arrow, since we assume that $(-) \otimes T^\infty$ preserves coequalizers.*

Remark. *The observant reader may have noticed that we left out an associator in the above diagram. Though necessary for formal proofs, associators and unitors reduce readability and may be cumbersome to work with, so we will leave them out for the rest of this section. See for example* `Tinf_mul_assoc_pointwise_associator_mess`*, a lemma used in a later proof.*

Example 7.39. *Let us just again think about this intuitively for* **TOP** *or* **SSET**. *In these cases, the maps τ_α represent 'collapsing X_α into T^∞'. Let us think about what the diagrams for τ_α mean: going around the top and right, we first collapse cells from $T \otimes X_\alpha$ down to $X_{\alpha+}$, and then collapse $X_{\alpha+}$ into T^∞ (which has 'all the cells'). Going around the left and bottom, we first collapse X_α into T^∞, and then collapse $T (\cong X_1)$ into T^∞. It seems reasonable that this diagram should commute.*

We want to define this family of morphisms inductively, but we will need information about the morphisms τ_α and $\tau_{\alpha+}$ in order to define $\tau_{\alpha++}$. Similar to the pair diagrams used to define the free algebra sequence, we introduce a helper type

```
Definition Tinf_pd_Tinf_map
    (n : nat)
    (p := free_monoid_coeq_sequence_on I_{V} n) :=
  ∑ (τn : pair_diagram_lob p ⊗_{V} Tinf --> Tinf)
    (τn1 : pair_diagram_rob p ⊗_{V} Tinf --> Tinf),
      (pair_diagram_arr p ⊗^{V}_{r} Tinf) · τn1
      = α_{V} _ _ _ · (T ⊗^{V}_{l} τn) · (Mon_alg_map _ (pr2 TinfM)).
```

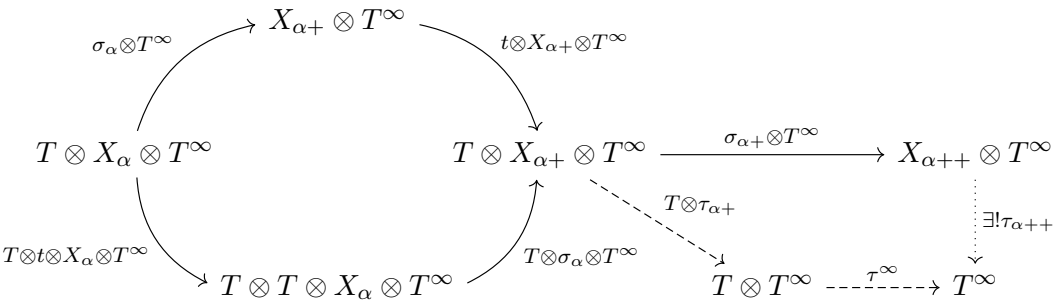
We define objects of this type inductively. Defining τ_0 and τ_1 is simple, we need the following diagram to commute:

$$\begin{array}{ccc} T \otimes I \otimes T^\infty & \xrightarrow{\mathrm{id}_{T\otimes I} \otimes T^\infty} & T \otimes I \otimes T^\infty \\ {\scriptstyle T\otimes\tau_0}\downarrow & & \downarrow{\scriptstyle \tau_1} \\ T \otimes T^\infty & \xrightarrow[\tau^\infty]{} & T^\infty \end{array}$$

There are some obvious choices for the maps. We set τ_0 to be the left unitor $\mathtt{lu}_\mathcal{V} T^\infty$, and we set τ_1 to be the right unitor of T, composed with τ^∞:

$$\tau_1 = ((\mathtt{ru}_\mathcal{V} T) \otimes T^\infty) \cdot \tau^\infty.$$

It is easy to check that the diagram commutes. For the inductive step we use coequalizer properties. Since we know that $(-) \otimes T^\infty$ preserves coequalizers, the solid arrows in the following diagram from a coequalizer diagram.



We can show that the map $\tau_{\alpha++}$ out of the coequalizer exists if the required commutativity relations hold.

Lemma 7.40 (`free_monoid_coeq_sequence_on_Tinf_pd_Tinf_map_coeqout`)**.** *The composites*

$$(\sigma_\alpha \otimes T^\infty)\cdot(t \otimes X_{\alpha+} \otimes T^\infty)\cdot(T \otimes \tau_{\alpha+})\cdot\tau^\infty$$

and

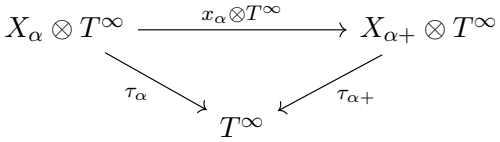
$$(T \otimes x_\alpha \otimes T^\infty)\cdot(T \otimes \tau_{\alpha+})\cdot\tau^\infty$$

are equal, and thus the unique map $\tau_{\alpha++}$ exists, and makes the diagram commute.

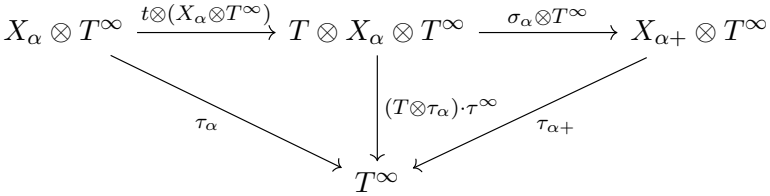
The required relation for $\tau_{\alpha+}$ and $\tau_{\alpha++}$ needed to construct a `Tinf_pd_Tinf_map` can be read off in the diagram. We can then show the following.

Lemma 7.41 (`free_monoid_coeq_sequence_colim_on_Tinf_Tinf_map`)**.** *The morphisms τ_α form a cocone on $\{ X_\alpha \otimes T^\infty \}$ with vertex T^∞.*

Proof. In other words, for any α, we want that the following diagram commutes



Note that by the properties of the τ_α, we know that the right triangle commutes in the following diagram



For the left triangle, whiskering properties give us that

$$(t \otimes (X_\alpha \otimes T^\infty))\cdot(T \otimes \tau_\alpha)\cdot\tau^\infty = \tau_\alpha\cdot(t \otimes T^\infty)\cdot\tau^\infty = \tau_\alpha.$$

where we used the property of the algebra map of T^∞ in the last step. □

And so, we have defined a map $\mu^\infty : T^\infty \otimes T^\infty \to T^\infty$ (`Tinf_monoid_mul`). We are left to show that the monoid axioms hold. We briefly go over the ideas of these proofs.

Remark. *It is important to keep in mind that μ^∞ collapses the* left *copy of T^∞ into the* right *copy, and not the other way around. It is also important to keep the difference between* left *and* right *tensoring with T^∞ in mind. Intuitively,* left *tensoring with T^∞ 'glues all cells to space on the right'. So $T^\infty \otimes X_\alpha$ means we glue 'all cells' to X_α. Whereas* right *tensoring with T^∞ means we 'replace the base space in the left factor with T^∞'. Meaning that $X_\alpha \otimes T^\infty$ means we have effectively glued α steps of cells to the 'space with all cells T^∞'.*

Lemma 7.42 (`Tinf_monoid_unit_left`)**.** *The left unit axiom holds for $(T^\infty, \eta^\infty, \mu^\infty)$.*

Proof. That is, we wish to show that

$$(\eta^\infty \otimes T^\infty)\cdot\mu^\infty = \mathrm{id}_{T^\infty}\,.$$

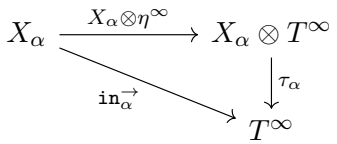
Now, since η^∞ is defined as $\mathtt{in}_0^{\rightarrow}$, and μ^∞ collapses the left copy of T^∞ into the right copy, this is pretty much trivial. Intuitively, $\eta^\infty \otimes T^\infty$ includes 'the space with all cells T^∞' into 'the space with all cells *with all cells again* $T^\infty \otimes T^\infty$'. The multiplication μ^∞ then collapses this second gluing of cells back down into T^∞. □

Lemma 7.43 (`Tinf_monoid_unit_right`)**.** *The right unit axiom holds for* $(T^\infty, \eta^\infty, \mu^\infty)$*.*

Proof. That is, we wish to show that

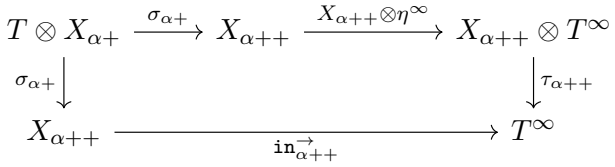
$$(T^\infty \otimes \eta^\infty) \cdot \mu^\infty = \mathrm{id}_{T^\infty}\,.$$

This is where the 'order of collapsing' for μ^∞ comes into play. We need that, at any ordinal α in the colimit, we have that

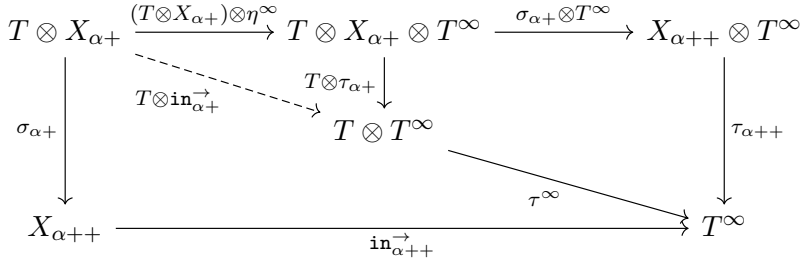


Going around the top and bottom, we include X_α into 'the space with all cells attached *with* α *more steps of attaching cells*', and then collapse the last α steps back down into T^∞. The diagonal arrow just includes X_α into T^∞. Again, intuitively this relation makes sense.

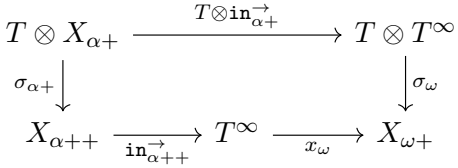
We show this inductively, and use the fact that the object $X_{\alpha++}$ is a coequalizer object. Checking for $\alpha = 0, 1$ is simple. Then the uniqueness of maps out of the coequalizer makes it so it suffices to show that the following diagram commutes.



Whiskering properties and the relation on τ_α allow us to rewrite this into



where the dashed arrow follows by induction. It now suffices to show that the bottom triangle commutes. We use the definition of τ^∞, and the fact that x_ω is an isomorphism. It now suffices to show that



which follows from the coequalizer properties of the limit ordinal step. In this proof, we left out a lot of monoidal unitors and associators for clarity. □

For the final monoid axiom, associativity, we will make use of the transfinite sequence on other A, not just that on I. We first note the following, using assumption **V2**.

Lemma 7.44 (`rt_chain_colim_iso`)**.** *The free* T*-algebra sequence on* A *is the same as the free* T*-algebra sequence on* I*, rightwhiskered with* A*, up to isomorphism. In other words, let* X_α *denote the terms in the sequence on* I*, and* X^A_α *those in the sequence on* A*. We have that*

$$\mathrm{colim}(X_\alpha \otimes A) \cong \mathrm{colim}\, X^A_\alpha.$$

Remark. *This may seem like a simple statement, one that may even be omitted from a paper. However, proving this is far from trivial in formalization, taking about 500 lines of formalization to prove in the accompanying formalization.*

Lemma 7.45 (`Tinf_monoid_assoc`)**.** *The associativity axiom holds for* $(T^\infty, \eta^\infty, \mu^\infty)$*.*

Proof. That is, we wish to show that

$$(T^\infty \otimes \mu^\infty) \cdot \mu^\infty = (\mu^\infty \otimes T^\infty) \cdot \mu^\infty.$$

Effectively, what this tells us is that given three copies of T^∞, as $T^\infty \otimes T^\infty \otimes T^\infty$, it does not matter whether we first collapse the two leftmost copies, and then collapse it into the last copy, or if we first collapse the two rightmost copies, and then collapse the leftmost copy into that result. Intuitively, it should hold. We use a similar strategy as `Tinf_monoid_unit_right` to show this formally.

By assumption **V2**, the rightwhiskering functor $(-) \otimes (T^\infty \otimes T^\infty)$ preserves the colimit of the transfinite sequence, we again pass to the individual terms of the colimit. It suffices to show that

$$\begin{array}{ccc} X_\alpha \otimes T^\infty \otimes T^\infty & \xrightarrow{\tau_\alpha \otimes T^\infty} & T^\infty \otimes T^\infty \\ {\scriptstyle X_\alpha \otimes \mu^\infty}\downarrow & & \downarrow{\scriptstyle \mu^\infty} \\ X_\alpha \otimes T^\infty & \xrightarrow[\tau_\alpha]{} & T^\infty \end{array}$$

We again show this inductively, where the cases for $\alpha = 0, 1$ are simple to show. By assumption **V2**, we also know that $(-) \otimes (T^\infty \otimes T^\infty)$ preserves coequalizers, making it so that $X_{\alpha++} \otimes T^\infty \otimes T^\infty$ is a coequalizer. Using the coequalizer properties, it suffices to show that

$$\begin{array}{ccccc} T \otimes X_{\alpha+} \otimes T^\infty \otimes T^\infty & \xrightarrow{\sigma_{\alpha+} \otimes (T^\infty \otimes T^\infty)} & X_{\alpha++} \otimes T^\infty \otimes T^\infty & \xrightarrow{X_{\alpha++} \otimes \mu^\infty} & X_{\alpha++} \otimes T^\infty \\ {\scriptstyle \sigma_{\alpha+} \otimes (T^\infty \otimes T^\infty)}\downarrow & & & & \downarrow{\scriptstyle \tau_{\alpha++}} \\ X_{\alpha++} \otimes T^\infty \otimes T^\infty & \xrightarrow[\tau_{\alpha++} \otimes T^\infty]{} & T^\infty \otimes T^\infty & \xrightarrow[\mu^\infty]{} & T^\infty \end{array}$$

Using whiskering properties and induction, we transform this into

$$\begin{array}{ccccc} T \otimes X_{\alpha+} \otimes T^\infty \otimes T^\infty & & \xrightarrow{(T \otimes X_{\alpha+}) \otimes \mu^\infty} & & T \otimes X_{\alpha+} \otimes T^\infty \\ & \searrow^{T \otimes \tau_{\alpha+} \otimes T^\infty} & & & \downarrow{\scriptstyle T \otimes \tau_{\alpha+} \otimes T^\infty} \\ {\scriptstyle \sigma_{\alpha+} \otimes (T^\infty \otimes T^\infty)}\downarrow & & T \otimes T^\infty \otimes T^\infty & \xrightarrow{T \otimes \mu^\infty} & T \otimes T^\infty \\ & & {\scriptstyle \tau^\infty \otimes T^\infty}\downarrow & & \downarrow{\scriptstyle \tau^\infty} \\ X_{\alpha++} \otimes T^\infty \otimes T^\infty & \xrightarrow[\tau_{\alpha++} \otimes T^\infty]{} & T^\infty \otimes T^\infty & \xrightarrow[\mu^\infty]{} & T^\infty \end{array}$$

where the dashed arrows follow from induction. The dotted arrow makes the bottom left-hand 'square' commute, as it is the relation on $\tau_{\alpha+}$ and $\tau_{\alpha++}$ rightwhiskered with T^∞. It suffices to show that the bottom right square commutes. This square boils down to the case $\alpha = 1$. We use the fact that $T^\infty \otimes T^\infty$ is isomorphic to the colimit of the free T-algebra sequence on T^∞ itself, from Lemma 7.44, combined with assumption **V3**, implying that $T \otimes (-)$ preserves the free T-algebra sequence on any $A : \mathbf{LNWFS}_\mathcal{C}$. These facts allow us to show this termwise in the colimit of the sequence on T^∞. It suffices to show that

$$\begin{array}{ccc} T \otimes X_\beta \otimes T^\infty & \xrightarrow{T \otimes \tau_\beta} & T \otimes T^\infty \\ {\scriptstyle \sigma_\beta \otimes T^\infty}\downarrow & & \downarrow{\scriptstyle \tau^\infty} \\ X_{\beta+} \otimes T^\infty & \xrightarrow[\tau_{\beta+}]{} & T^\infty \end{array}$$

for any vertex β in our diagram. This is precisely the relation on τ_β and $\tau_{\beta+}$. □

We used some properties on $\mathcal{V}$ and T that we need to show for the construction to work with $\mathcal{V} = \mathbf{LNWFS}_{\mathcal{C}}$ and $T = L^1$. The first of which is assumption **V1**, saying that $\mathbf{LNWFS}_{\mathcal{C}}$ has all connected, non-empty colimits. Secondly, we used assumption **V2**, saying that $(-) \otimes T^\infty$ and $(-) \otimes (T^\infty \otimes T^\infty)$ preserve chains and coequalizers. In fact, we can show something stronger, which is that the rightwhiskering functor $(-) \otimes A : \mathbf{LNWFS}_{\mathcal{C}} \longrightarrow \mathbf{LNWFS}_{\mathcal{C}}$ preserves colimits for any $A : \mathbf{LNWFS}_{\mathcal{C}}$. Lastly, we used assumption **V3**, saying that $L^1 \otimes (-)$ is ω-small. We will reduce this assumption to a much weaker requirement, that only involves the morphism class J used to define L^1. Garner uses high level arguments in his article, but since these are not available to us in `UniMath`, we have to come up with different, more direct arguments.

7.2.7 Cocompleteness of $\mathbf{LNWFS}_{\mathcal{C}}$

First we show assumption **V1**, by showing that $\mathbf{LNWFS}_{\mathcal{C}}$ has all connected, non-empty colimits whenever $\mathcal{C}$ is cocomplete. More specifically, we will only use coequalizers and chains, but it is possible to show that $\mathbf{LNWFS}_{\mathcal{C}}$ has all connected colimits. In his article, Garner uses some high level, category theoretical machinery to show that $\mathbf{LNWFS}_{\mathcal{C}}$ is cocomplete whenever $\mathcal{C}$ is [9, Proposition 4.18]. Sadly, this machinery is not available to us in `UniMath`.

We take a more direct approach, and first show that $\mathbf{Ff}_{\mathcal{C}}$ has all connected, non-empty colimits. We will then use this to show that $\mathbf{LNWFS}_{\mathcal{C}}$ has all connected, non-empty colimits as well. The colimit of a diagram in $\mathbf{LNWFS}_{\mathcal{C}}$ will then lie over the colimit of the projected diagram in $\mathbf{Ff}_{\mathcal{C}}$, as one might expect with the displayed nature of $\mathbf{LNWFS}_{\mathcal{C}}$ over $\mathbf{Ff}_{\mathcal{C}}$. In short, we will first show the following.

Lemma 7.46 (`ColimFfCocone`)**.** *The category of functorial factorizations $\mathbf{Ff}_{\mathcal{C}}$ has all connected, non-empty colimits whenever $\mathcal{C}$ is cocomplete.*

For an empty diagram, we could simply take the colimit to be the monoidal unit I, as it is initial. We do not really care about empty diagrams though, only about the following two specific cases.

Corollary 7.47 (`ChainsFf`, `CoequalizersFf`)**.** *The category of functorial factorizations $\mathbf{Ff}_{\mathcal{C}}$ has all colimits of chains, and all coequalizers.*

To show that $\mathbf{Ff}_{\mathcal{C}}$ has all connected, non-empty colimits, we cannot simply use the fact that functor categories $[\mathcal{A}, \mathcal{B}]$ are cocomplete whenever $\mathcal{B}$ is. To illustrate why, let us take a look at the constructive proof proof for this in `UniMath` (`ColimsFunctorCategory`).

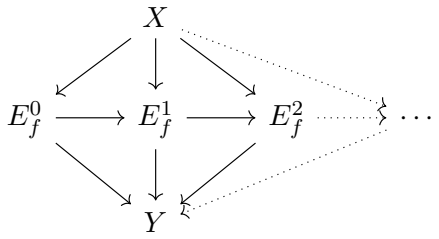
Let g be a graph, and $d = \{F_v\}_{v:g}$ a diagram over the graph g in $[\mathcal{A}, \mathcal{B}]$. We define the colimit F_∞ pointwise, i.e. $F_\infty(a)$ is defined as the colimit $\operatorname{colim} F_v(a)$ in $\mathcal{B}$ for any $a : \mathcal{A}$. Consider what this pointwise diagram would look like if we take $\{F_v\}_{v:g}$ to be a chain (so a diagram on $g = \mathbb{N}$) of lifted functors corresponding to functorial factorizations, in $[\mathcal{C}^{\mathbf{2}}, \mathcal{C}^{\mathbf{3}}]$, at the point $(f : X \to Y) : \mathcal{C}^{\mathbf{2}}$:

$$\begin{array}{ccccccc}
X & = & X & = & X & \cdots\cdots\rightarrow & X^\infty \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
K_0 f & \longrightarrow & K_1 f & \longrightarrow & K_2 f & \cdots\cdots\rightarrow & E_f^\infty \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
Y & = & Y & = & Y & \cdots\cdots\rightarrow & Y^\infty
\end{array}$$

Note that in this colimit, we take a colimit of objects in $\mathcal{C}^{\mathbf{3}}$. It seems like everything is fine here, but the domain of the colimit is defined as the colimit of the chain of all identities on X, and similarly, the codomain is defined as the colimit of the chain of identities on Y. Obviously,

these will be isomorphic to X and Y respectively, but they need not be *definitionally* equal. We could define F_∞ by correcting the domain and codomain with these isomorphisms. This would become quite cumbersome to work with, since we want to define a comonad structure on the left functor of F_∞ when defining colimits on $\mathbf{LNWFS}_\mathcal{C}$, and we would rather keep our terms simple.

Instead, we will define the colimit more constructively, in a similar fashion as the above proof. First, let us recall what the actual data is in a functorial factorization: the middle object. We define a sequence of middle objects, and only take the middle map of the section transformations to form a diagram that looks like this



Let E_f^∞ be the colimit of the middle objects, which exists because $\mathcal{C}$ is cocomplete. We get map $E_f^\infty \to Y$ using the properties of colimits. We still need the map $X \to E_f^\infty$, which we can only define when the colimit is non-empty, namely as the canonical inclusion of $X \to E_f^{v_0} \to E_f^\infty$ for an arbitrary vertex $v_0 : g$. For this to be well-defined, we need a transformation of sections $F_v f \hookrightarrow F_\infty f$ including any $F_v(f)$ into the colimit, which can only be shown to define a cocone whenever the diagram is connected.

Using this constructed colimit in $\mathbf{Ff}_\mathcal{C}$, we want to construct a colimit for any connected, non-empty diagram in $\mathbf{LNWFS}_\mathcal{C}$.

Lemma 7.48 (`ColimLNWFSCocone`)**.** *The category* $\mathbf{LNWFS}_\mathcal{C}$ *has all connected, non-empty colimits whenever* $\mathcal{C}$ *is cocomplete.*

Again, we only care about two specific cases.

Corollary 7.49 (`ChainsLNWFS`, `CoequalizersLNWFS`)**.** *The category* $\mathbf{LNWFS}_\mathcal{C}$ *has all colimits of chains, and all coequalizers.*

Suppose we have a connected, non-empty diagram in $\mathbf{LNWFS}_\mathcal{C}$. We can project this diagram down to one in $\mathbf{Ff}_\mathcal{C}$ and obtain the colimit F_∞ of the projected diagram. We want to define a comonad structure on L_{F_∞}, using the comonad structures on the individual L_{F_v}. Garner's argument is fairly brief, stating that the forgetful functor $\mathbf{LNWFS}_\mathcal{C} \to \mathbf{Ff}_\mathcal{C}$ creates colimits, since $\mathbf{LNWFS}_\mathcal{C}$ can be defined as the category of comonoids in $\mathbf{Ff}_\mathcal{C}$ for a certain comonoidal structure $(\odot, \bot)$. This comonoidal structure is dual to $(\otimes, I)$, and is defined with unit

$$\bot : X \xrightarrow{f} Y \quad \mapsto \quad X \xrightarrow{f} Y \xrightarrow{\mathrm{id}_Y} Y.$$

and the comultiplication on two factorizations F, F' is defined as

$$F' \odot F : X \xrightarrow{f} Y \quad \mapsto \quad X \xrightarrow{\lambda'_{\lambda_f}} E'_{\lambda_f} \xrightarrow{\rho'_{\lambda_f} \cdot \rho_f} Y.$$

There is a more elaborate argument for this by Johnstone [26, Lemma 1.1.8]. This argument defines the morphism $L_{F_\infty} \Longrightarrow L^2_{F_\infty}$, by using the fact that the colimit distributes over the comonoidal structure $(\odot, \bot)$. In other words, Johnstone says that

$$\mathrm{colim}_{v,w}(F_v \odot F_w) = \mathrm{colim}_v(F_v \odot F_v)$$

which boils down to saying that

$$L^2_{F_\infty} = \mathrm{colim}_{v,w}\, L_{F_v} \cdot L_{F_w} = \mathrm{colim}_v\, L^2_{F_v}$$

for the left functors. Sadly, the `UniMath` library again does not provide this type of machinery. In fact, this second equality will not even really be an equality, but rather an isomorphism. Let us recall again what the actual data in the comonad structure is. At any vertex $v : g$, at any arrow f, it is the map σ_f^v in the component Σ_f^v of the comultiplication of the comonad structure on L_{F_v}

$$\Sigma_f^v := \begin{array}{ccc} X & = & X \\ {\scriptstyle \lambda_f^v}\downarrow & & \downarrow{\scriptstyle \lambda^v_{\lambda_f^v}} \\ E_f^v & \xrightarrow[\sigma_f^v]{} & E^v_{\lambda_f^v} \end{array}$$

Again for illustration, let us assume the diagram in $\mathbf{LNWFS}_{\mathcal{C}}$ we are considering is a chain. We define a pointwise diagram of the maps σ_f^v:

$$\begin{array}{ccccccc} E_f^0 & \longrightarrow & E_f^1 & \longrightarrow & E_f^2 & \dashrightarrow & \cdots \\ {\scriptstyle \sigma_f^0}\downarrow & & {\scriptstyle \sigma_f^1}\downarrow & & {\scriptstyle \sigma_f^2}\downarrow & & \\ E^0_{\lambda_f^0} & \longrightarrow & E^1_{\lambda_f^1} & \longrightarrow & E^2_{\lambda_f^2} & \dashrightarrow & \cdots \end{array}$$

The colimit of this diagram exists, since $\mathcal{C}^{\mathbf{2}}$ is cocomplete whenever $\mathcal{C}$ is (Lemma 3.14 or `arrow_colims`). This yields an arrow

$$\sigma_f^{\infty,*} : E_f^\infty \to E_f^{\infty,\infty}$$

where the domain is definitionally equal to the middle object of $F_\infty f$, as desired. However, the codomain $E_f^{\infty,\infty}$ is not definitionally equal to the codomain of $L_{F_\infty} \cdot L_{F_\infty} f$. This problem corresponds exactly with the distributing of the colimit over the composition of the left functors. This is because the codomain of $L_{F_\infty} \cdot L_{F_\infty} f$ is definitionally equal to

$$\operatorname{colim}_v \operatorname{cod} L_{F_v} \left(L_{F_\infty} f\right) = \operatorname{colim}_v \operatorname{cod} L_{F_v} \left(\operatorname{colim}_w L_{F_w} f\right),$$

where we take the colimits sequentially, whereas $E_f^{\infty,\infty}$ is definitionally equal to

$$\operatorname{colim}_v \operatorname{cod} L^2_{F_v} f,$$

where we effectively take the colimits simultaneously. We define a morphism

$$\sigma^\infty_{\mathrm{cod}} : \operatorname{colim}_v \operatorname{cod} L^2_{F_v} f \to \operatorname{colim}_v \operatorname{cod} L_{F_v} \left(L_{F_\infty} f\right),$$

termwise, with

$$\operatorname{cod} L^2_{F_v} f \to \operatorname{cod} L_{F_v} \left(L_{F_\infty} f\right)$$

as

$$\left[L_{F_v}\left(\mathtt{in}^{\rightarrow}_{L_{F_v} f} : L_{F_v} f \hookrightarrow L_{F_\infty} f\right)\right]_{11}.$$

This gives us the morphism

$$\sigma_f^\infty = \sigma_f^{\infty,*} \cdot \sigma^\infty_{\mathrm{cod}} : \operatorname{cod} \lambda_f^\infty \to \operatorname{cod} \lambda^\infty_{\lambda_f^\infty}$$

that we need. One can show the following.

Lemma 7.50 (`LNWFS_colim_comul_monad_ax`)**.** *The morphisms σ_f^∞ defined above can be used to form a natural transformation $\Sigma^\infty : L_{F_\infty} \Longrightarrow L^2_{F_\infty}$, extending the left functor of F_∞ to a comonad.*

7.2.8 Right-closedness of $\mathbf{LNWFS}_{\mathcal{C}}$

The other property of $\mathbf{LNWFS}_{\mathcal{C}}$ that we want to show is assumption **V2**, saying the following.

Lemma 7.51 (`LNWFS_rt_all_colimits`)**.** *The rightwhiskering functor*

$$(-) \otimes A : \mathbf{LNWFS}_{\mathcal{C}} \longrightarrow \mathbf{LNWFS}_{\mathcal{C}}$$

preserves colimits for any $A : \mathbf{LNWFS}_{\mathcal{C}}$*. In other words,* $\mathbf{LNWFS}_{\mathcal{C}}$ *is* right-closed*.*

Here, we are again only interested in two cases.

Corollary 7.52 (`LNWFS_rt_chain`, `LNWFS_rt_coeq`)**.** *The rightwhiskering functor* $(-) \otimes A : \mathbf{LNWFS}_{\mathcal{C}} \longrightarrow \mathbf{LNWFS}_{\mathcal{C}}$ *preserves colimits of chains and coequalizers for any* $A : \mathbf{LNWFS}_{\mathcal{C}}$*.*

Here too, Garner uses a fairly high level argument, but it is possible to take a more direct approach. Similar to the last two proofs, we first show that $(-) \otimes A : \mathbf{Ff}_{\mathcal{C}} \longrightarrow \mathbf{Ff}_{\mathcal{C}}$ preserves colimits. We do this by showing that there is an inverse

$$F_\infty \otimes A \to \operatorname{colim}(F_v \otimes A)$$

to the canonical map out of the colimit. We then show that this inverse is also a morphism of $\mathbf{LNWFS}_{\mathcal{C}}$ whenever the chain was projected down from $\mathbf{LNWFS}_{\mathcal{C}}$ to begin with. Firstly though, we show the following.

Lemma 7.53 (`Ff_rt_all_colimits`)**.** *The rightwhiskering functor* $(-) \otimes A : \mathbf{Ff}_{\mathcal{C}} \longrightarrow \mathbf{Ff}_{\mathcal{C}}$ *preserves colimits for any* $A : \mathbf{Ff}_{\mathcal{C}}$*.*

We define the inverse directly. Like before, we do this pointwise on an arrow $(f : X \to Y) : \mathcal{C}^{\mathbf{2}}$, by defining an arrow of the middle objects. From the way the tensor product is defined, this means that we need to find an isomorphism

$$\operatorname{dom} \rho^{\infty}_{\rho^A_f} \to \operatorname{colim} \operatorname{dom} \rho^{v}_{\rho^A f}$$

where ρ^∞ corresponds with F_∞, ρ^v with F_v and ρ^A with A. We simply define this termwise as

$$\operatorname{colim} \mathrm{id}_{\operatorname{dom} \rho^{v}_{\rho^A_f}} .$$

Though it may seem trivial that this morphism lifts to a morphism in $\mathbf{LNWFS}_{\mathcal{C}}$ whenever the sequence in $\mathbf{Ff}_{\mathcal{C}}$ was obtained from one in $\mathbf{LNWFS}_{\mathcal{C}}$, we use the following lemma. This strategy proved to be much more performant in the proof checker, and the lemma can be applied in some other proofs as well.

Lemma 7.54 (`Ff_iso_inv_LNWFS_mor`)**.** *Suppose* L *and* L' *are LNWFSs over functorial factorizations* F *and* F' *respectively. Let* $\tau : F \to F'$ *be an isomorphism which preserves the LNWFS structrure, or in other words, there is a morphism* $\Gamma : L \to L'$ *over* τ*. Then* $\tau^{-1} : F' \to F$ *also preserves the LNWFS structure, or in other words:* Γ *is an isomorphism with inverse* Γ^{-1} *over* τ^{-1}*.*

Proof. We have to show that τ^{-1} preserves the counits and comultiplication. We first show that it preserves the counit. This means that for any $f : X \to Y$, we need

$$\begin{array}{ccccc} X & = & X & = & X \\ {\scriptstyle \lambda'_f}\downarrow & & {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle f} \\ E'_f & \xrightarrow[\tau_f^{-1}{}_{11}]{} & E_f & \xrightarrow{\rho_f} & Y \end{array} \quad = \quad \begin{array}{ccc} X & = & X \\ {\scriptstyle \lambda'_f}\downarrow & & \downarrow{\scriptstyle f} \\ E'_f & \xrightarrow[\rho'_f]{} & Y \end{array}$$

as morphisms in the arrow category $\mathcal{C}^{\mathbf{2}}$. This follows pretty much directly from precomposing with the inverse τ_f of τ_f^{-1}, and using the counit preservation law of τ.

Showing that the multiplication is preserved is a bit more involved. We wish to show that the following morphisms of the arrow category are equal:

$$\begin{array}{ccccc} X & = & X & = & X \\ {\scriptstyle \lambda'_f}\downarrow & & {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E'_f & \xrightarrow[{\tau_f^{-1}}_{11}]{} & E_f & \xrightarrow{\sigma_f} & E_{\lambda_f} \end{array} \quad = \quad \begin{array}{ccccccc} X & = & X & = & X & = & X \\ {\scriptstyle \lambda'_f}\downarrow & & \downarrow{\scriptstyle \lambda'_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda'_{\lambda_f}} & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E'_f & \xrightarrow{\sigma'_f} & \square & \xrightarrow[L'({\tau_f^{-1}})_{11}]{} & \square & \xrightarrow[{\tau^{-1}_{\lambda_f}}_{11}]{} & E_{\lambda_f} \end{array}$$

We again precompose with τ_f to obtain

$$\begin{array}{ccc} X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_f & \xrightarrow{\sigma_f} & E_{\lambda_f} \end{array} \quad = \quad \begin{array}{ccccccc} X & = & X & = & X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & {\scriptstyle \lambda'_f}\downarrow & & \downarrow{\scriptstyle \lambda'_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_f & \xrightarrow[{\tau_f}_{11}]{} & E'_f & \xrightarrow{\sigma'_f} & \square & \xrightarrow[L'({\tau_f^{-1}})_{11}]{} & \square & \xrightarrow[{\tau^{-1}_{\lambda_f}}_{11}]{} & E_{\lambda_f} \end{array}$$

Rewriting the multiplication law for τ gives

$$\begin{array}{ccc} X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_f & \xrightarrow{\sigma_f} & E_{\lambda_f} \end{array} \quad = \quad \begin{array}{ccccccccc} X & = & X & = & X & = & X & = & X \\ {\scriptstyle \lambda_f}\downarrow & & {\scriptstyle \lambda_{\lambda'_f}}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda'_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_f & \xrightarrow{\sigma_f} & \square & \xrightarrow[L({\tau_f})_{11}]{} & \square & \xrightarrow[{\tau_{\lambda'_f}}_{11}]{} & \square & \xrightarrow[L'({\tau_f^{-1}})_{11}]{} & \square & \xrightarrow[{\tau^{-1}_{\lambda_f}}_{11}]{} & E_{\lambda_f} \end{array}$$

for which it suffices to show that

$$\begin{array}{ccccccc} X & = & X & = & X & = & X \\ {\scriptstyle \lambda_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda'_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_{\lambda_f} & \xrightarrow[L({\tau_f})_{11}]{} & \square & \xrightarrow[{\tau_{\lambda'_f}}_{11}]{} & \square & \xrightarrow[L'({\tau_f^{-1}})_{11}]{} & \square & \xrightarrow[{\tau^{-1}_{\lambda_f}}_{11}]{} & E_{\lambda_f} \end{array}$$

is identity on λ_{λ_f}. In fact, if we look closely, this boils down to the commutativity of whiskering in the comonoidal structure $(\odot, \bot)$ on $\mathbf{Ff}_{\mathcal{C}}$. In other words, we want to know that

$$(\alpha \odot \Lambda) \cdot (L' \odot \beta) = (L \odot \beta) \cdot (\alpha \odot \Lambda')$$

for $\alpha : L \to L'$ and $\beta : \Lambda \to \Lambda'$ morphisms of $\mathbf{LNWFS}_{\mathcal{C}}$. That is, for any morphism $f : X \to Y$, we need that

$$\alpha_{\Lambda f} \cdot L'(\beta_f) = L(\beta_f) \cdot \alpha_{\Lambda' f}.$$

This proof can be found in the formalization (`LNWFS_comon_structure_whiskercommutes`), and it allows us to rewrite the middle two squares to obtain

$$\begin{array}{ccccccc} X & = & X & = & X & = & X \\ {\scriptstyle \lambda_{\lambda_f}}\downarrow & & \downarrow{\scriptstyle \lambda_{\lambda'_f}} & & \downarrow{\scriptstyle \lambda_{\lambda_f}} & & \downarrow{\scriptstyle \lambda_{\lambda_f}} \\ E_{\lambda_f} & \xrightarrow[L({\tau_f})_{11}]{} & \square & \xrightarrow[L({\tau_f^{-1}})_{11}]{} & \square & \xrightarrow{\tau_{\lambda_f}{}_{11}} & \square & \xrightarrow[{\tau^{-1}_{\lambda_f}}_{11}]{} & E_{\lambda_f} \end{array}$$

where the two halves are both identity by functoriality and naturality. □

7.2.9 Reducing the Smallness Requirement

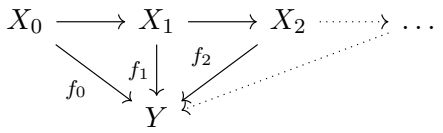
The last thing we want to do is to reduce the smallness requirement **V3** on $\mathbf{LNWFS}_{\mathcal{C}}$ to one that only involves the morphism class J used to define L^1. The smallness requirement on $\mathbf{LNWFS}_{\mathcal{C}}$ we currently need in order to show that the free monoid exists is that L^1 is ω-small. We first generically reduce this requirement to one on $\mathbf{Ff}_{\mathcal{C}}$, then on the left functor for a functorial factorization, then finally we reduce it to a (somewhat familiar) requirement on J. For the generic part, we follow Garner's proof [22, Proposition 32], which may not be the quickest way there, but it does work. For the smallness of the one-step comonad itself we take a slightly different route.

In this section, all diagrams we consider are chains, as they are sufficient to show the smallness requirement, but it should be possible to prove these statements for any connected, non-empty colimit. This is why we will still call the chains d for 'diagram', and label the vertices with v for 'vertex', as opposed to more common labeling for natural numbers.

First, Lemma 7.54 gives us the following.

Lemma 7.55 (`Ff_lt_preserves_colim_impl_LNWFS_lt_preserves_colim`)**.** *Let $d = \{ L_v \}_{v:\mathbb{N}}$ be a chain in $\mathbf{LNWFS}_{\mathcal{C}}$, lying over a chain $\delta = \{ F_v \}_{v:\mathbb{N}}$ in $\mathbf{Ff}_{\mathcal{C}}$. Suppose $L : \mathbf{LNWFS}_{\mathcal{C}}$ lies over $F : \mathbf{Ff}_{\mathcal{C}}$. Then the leftwhiskering functor $L \otimes (-)$ preserves the colimit of d whenever $F \otimes (-)$ preserves the colimit of δ.*

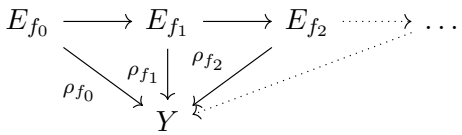
Next, we want to reduce the smallness on a factorization $F : \mathbf{Ff}_{\mathcal{C}}$ to some smallness requirement on the right functor R_F. Garner notes that R_F restricts to endofunctors $R_F\big|_Y : \mathcal{C}/Y \longrightarrow \mathcal{C}/Y$ of the slice category, and shows that it suffices to assume that every $R_F\big|_Y$ is ω-small. We take a slightly different approach. Let $d = \{ X_v \}_{v:\mathbb{N}}$ be a chain in the $\mathcal{C}$, and let cc_d^Y be a cocone on d, i.e. a diagram



Note that '$F \otimes (-)$ preserving the colimit of any chain' means that there is an isomorphism

$$\operatorname{colim}(F f_v) \cong F \operatorname{colim} f_i$$

for any such cocone. Define $R\left(\mathrm{cc}_d^Y\right)$ to be the cocone



Definition 7.56 (`FR_slice_omega_small`)**.** *We say that R_F is* slice ω-small *if and only if for any chain d and cocone cc_d^Y, the domain*

$$\operatorname{dom}(R_F \operatorname{colim} f_v)$$

is a colimit for the chain $\{ E_{f_v} \}_{v:\mathbb{N}}$.

```
Definition FR_slice_omega_small (F : Ff C) : UU :=
    ∏ (d : chain C) (y : C) (ccy : cocone d y),
      isColimCocone _ _ (dom_fact_R_colimArrow_cocone F ccy).
```

To reduce our smallness requirement, we show the following.

Lemma 7.57 (`FR_lt_preserves_colim_impl_Ff_lt_preserves_colim`)**.** *$F \otimes (-)$ is ω-small if R_F is slice ω-small.*

To show this, we note that the canonical isomorphism

$$\operatorname{colim}(\operatorname{dom}(R_F f_v)) \cong \operatorname{dom}(R_F \operatorname{colim} f_v)$$

from the smallness requirement on R_F precisely gives us the middle morphism of

$$\operatorname{colim}(F f_v) \cong F \operatorname{colim} f_v$$

that we need.

The final 'generic' step is to show that for any functorial factorization F, the right functor R_F is slice ω-small whenever the left functor L_F is ω-small.

Lemma 7.58 (`FR_slice_omega_small_if_L_omega_small`)**.** *For any functorial factorization F, the right functor R_F is slice ω-small whenever the left functor L_F is ω-small.*

The main point of the proof is that we note that for any chain d in $\mathcal{C}$ and cocone cc_d^Y like before, the canonical arrow

$$\operatorname{colim} f_v : \operatorname{colim}(X_v) \to Y$$

is in fact a colimit for the chain $\{ f_v \}_{v:\mathbb{N}}$ in $\mathcal{C}^{\mathbf{2}}$.

So in order to apply our free monoid construction on the one-step comonad L^1 in $\mathbf{LNWFS}_{\mathcal{C}}$, we are left with a smallness requirement on L^1 that says that the left functor L^1 is ω-small. We recall how L^1 was constructed, which is pointwise as a pushout of the functor

$$K : \mathcal{C}^{\mathbf{2}} \longrightarrow \mathcal{C}^{\mathbf{2}} : f \mapsto \bigsqcup_{x:S_f} g_x.$$

We can show the following.

Lemma 7.59 (`L1_small_if_K_small`)**.** *The one-step comonad L^1 is ω-small whenever K is.*

In fact, we show something even stronger.

Lemma 7.60 (`L1_preserves_colim_if_K_preserves_colim`)**.** *The one-step comonad L^1 preserves any colimit which K does.*

For this to hold, we need to find an inverse

$$L^1(\operatorname{colim} f_v) \to \operatorname{colim}\left(L^1 f_v\right)$$

to the canonical map

$$\operatorname{colim}\left(L^1 f_v\right) \to L^1(\operatorname{colim} f_v)$$

for any chain $d = \{ f_v \}_{v:\mathbb{N}}$ in $\mathcal{C}^{\mathbf{2}}$. To construct this inverse, first recall the natural transformation

$$\xi : K \Longrightarrow L^1$$

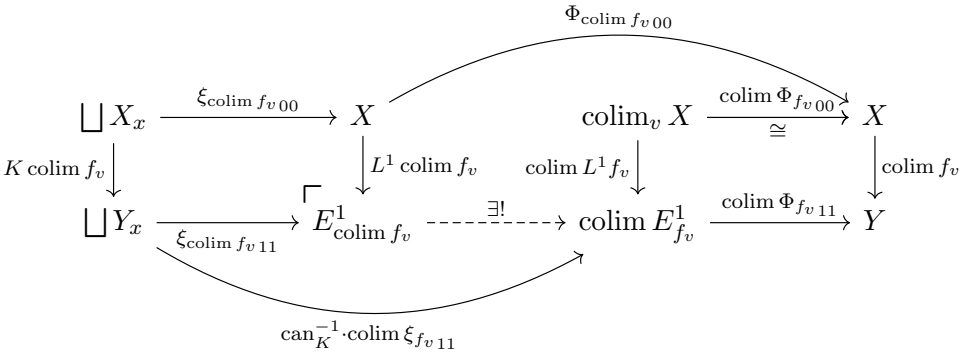
for which every component is the pushout $Kf \to L^1 f$, and recall the counit

$$\Phi : L^1 \Longrightarrow \mathrm{id}_{\mathcal{C}^{\mathbf{2}}}.$$

Garner then considers the following diagram [9, Proposition 4.22]

$$\begin{array}{ccccc}
K(\operatorname{colim} f_v) & \xrightarrow[\cong]{\mathrm{can}_K^{-1}} & \operatorname{colim} K f_v & \xrightarrow{\operatorname{colim} \xi_{f_v}} & \operatorname{colim} L^1 f_v \\
{\scriptstyle \xi_{\operatorname{colim} f_v}}\downarrow & & & & \downarrow{\scriptstyle \operatorname{colim} \Phi_{f_v}} \\
L^1(\operatorname{colim} f_v) & & \xrightarrow[\Phi_{\operatorname{colim} f_v}]{} & & \operatorname{colim} f_v
\end{array}$$

where $\mathrm{can}_K : \operatorname{colim} K f_v \to K \operatorname{colim} f_v$ is the isomorphism we get from the smallness of K. Garner mentions that the left map is a pushout, and the right map is an isomorphism on the domain, and that the corresponding two classes of maps form a (strong) factorization system. We will not show this, nor will we show the commutativity of the full diagram, but we will construct the lift using the properties of this diagram (and of course the necessary commutativity). Let us draw out the left and right vertical arrows, and part of the horizontal maps, and the idea will become more clear.



The existence (and uniqueness) of the dashed arrow follow from the pushout property of the left-hand square, in case the required diagrams commute (`L1_colim_L1_map_ispushoutOut`). The uniqueness of this map explains why the factorization system Garner mentions is strong. One can show that this morphism is indeed the morphism on the codomains that we need for our inverse

$$L^1 \operatorname{colim} f_v \to \operatorname{colim} L^1 f_v.$$

The morphism on the domains is simply the composite

$$\Phi_{\operatorname{colim} f_v 00} \cdot \left(\operatorname{colim} \Phi_{f_v 00}\right)^{-1}$$

used to define the pushout arrow. Commutativity of our inverse then follows directly from the properties of a pushout. Indeed, we have the following.

Lemma 7.61 (`L1_colim_L1_map_is_inverse_in_precat`)**.** *The morphism*

$$L^1 \operatorname{colim} f_v \to \operatorname{colim} L^1 f_v$$

constructed above is an inverse to the canonical morphism

$$\operatorname{colim} L^1 f_v \to L^1 \operatorname{colim} f_v.$$

The last step is to reduce the new smallness requirement on K to one that only involves our morphism class J. This will involve the notion of *presentable* objects in a category.

Definition 7.62 (`presentable`)**.** *Let $\mathcal{C}$ be a category and X and object in $\mathcal{C}$. Then X is called* presentable *if and only if the covariant homset functor*

$$\operatorname{Hom}(X, -) : \mathcal{C} \longrightarrow \mathbf{SET}$$

is ω-small.

```
Definition presentable {C : category} (x : C) :=
    preserves_colimits_of_shape
      (cov_homSet_functor x)
      nat_graph.
```

The goal is to show the following.

Lemma 7.63 (`K_small_if_J_small`)**.** *The functor K preserves colimits of chains whenever every arrow $g \in J$ is presentable.*

Remark. *In this lemma, we ask that any* morphism *$g \in J$ is presentable. This means that g is presentable in the* arrow category *$\mathcal{C}^{\mathbf{2}}$. We do not require anything specific about presentability in the base category $\mathcal{C}$.*

Again, let $d := \{ f_v \}_{v:\mathbb{N}}$ be a chain. Similar to before, we want to find an inverse

$$K \operatorname{colim} f_v \to \operatorname{colim} K f_v$$

to the canonical map out of the colimit. For brevity, we set

$$f_\infty := \operatorname{colim} f_v.$$

Since $K f_\infty$ is defined as the coproduct

$$\bigsqcup_{x:S_{f_\infty}} g_x$$

in the arrow category, it suffices to define morphisms

$$g_x \to \operatorname{colim} K f_v$$

for any $x : S_{f_\infty}$. Now, since all $g \in J$ are presentable, there is a canonical isomorphism of sets

$$(g_x \to f_\infty) \cong \operatorname{colim}_v (g_x \to f_v).$$

Unpacking the set quotient that defnies the colimit in **SET**, this isomorphism says that a lifting problem $x : S_{f_\infty}$ corresponds with an entire equivalence class of 'compatible' lifting problems $S_{f_\infty}^{g_x} \subseteq \bigsqcup_{v:\mathbb{N}}(g_x \to f_v)$. Here, 'compatible' means they correspond through composition with the edges $f_v \to f_{v+}$ in the chain d. It tells us that we can define a morphism $g_x \to \operatorname{colim} K f_v$ if we can define morphisms

$$i_{x_v} : g_x = g_{x_v} \to \operatorname{colim} K f_v$$

given any lifting problem $x_v : g_x \to f_v$ in $S_{f_\infty}^{g_x}$, preserving the equivalence relation on $\operatorname{colim}_v (g_x, f_v)$. This preservation requires that if x_u and x_v are in the same equivalence class in $\operatorname{colim}_v (g_x \to f_v)$ in **SET**, then i_{x_u} and i_{x_v} are the same. Defining these morphisms is easy to do though. We define them as the composite

$$i_{x_v} := g_x = g_{x_v} \xrightarrow{\mathtt{in}^{\sqcup}_{x_v}} K f_v \xrightarrow{\mathtt{in}^{\to}_{v}} \operatorname{colim} K f_v.$$

We can show the following.

Lemma 7.64 (`presentable_lp_homSet_colim_colimK_fun_iscomprel`)**.** *The morphisms i_{x_v} are compatible in the way described above.*

Indeed, the coproduct of the morphisms $g_x \to \operatorname{colim} K f_v$ obtained in this way yields the inverse we are looking for.

7.3 The Small Object Argument

In the end, the smallness requirement we impose on our morphism class J is the following.

```
Definition class_presentable {C : category} (J : morphism_class C) :=
  ∏ (g : arrow C), J _ _ g -> (presentable g).
```

Allowing us to prove the main theorem of interest.

Theorem 7.65 (small_object_argument)**.** *Let J be a morphism class in a cocomplete category $\mathcal{C}$, such that any $g \in J$ is presentable in the arrow category $\mathcal{C}^{\mathbf{2}}$. Then there exists an NWFS in $\mathcal{C}$.*

```
Theorem small_object_argument
    (HJ : class_presentable J) :
  total_category (NWFS C).
Proof.
  set (lnwfs_monoid :=
    Tinf_monoid
      (@LNWFS_tot_monoidal C)
      (LNWFS_pointed one_step_comonad_as_LNWFS)
      (ChainsLNWFS CC)
      (CoequalizersLNWFS CC)
      (free_monoid_coeq_sequence_converges_for_osc HJ)
      (LNWFS_rt_coeq CC)
      (LNWFS_rt_chain CC)
      (osc_preserves_diagram_on HJ)
  ).

  exact (_,, LNWFS_tot_monoid_is_NWFS lnwfs_monoid).
Defined.
```

Example 7.66. *Unfortunately, we cannot finish the example in* **TOP** *this way, as the arrows in*

$$J = \{\, j_n : S^n \hookrightarrow D_{n+1} \mid n = -1, 0, 1, \dots \,\}$$

are not *presentable in the category* **TOP***. In fact, the smallness requirement will not hold for* any *(non-empty) class J. There is, however, another smallness requirement on the class J that can be shown to be sufficient, that does hold for this class J. The NWFS that is generated in this way is precisely the one with Serre fibrations as right class and left class the retracts of relative CW-complexes, which are also weak homotopy equivalences. Indeed, this is one of the factorization systems from the* classical model structure on **TOP** *[9].*

Example 7.67. *We can, however, apply the theorem on* **SSET***, with the same class J. In this case, the NWFS that is generated is the one with Kan fibrations as right class, and the corresponding left class from the WFS in the* Quillen model structure on **SSET** *[9].*

Example 7.68. *The (trivial) example on* **SET***, with $J = \{\, \emptyset \to * \,\}$, generates the NWFS with underlying factorization [9]*

$$X \xrightarrow{f} Y \mapsto X \xrightarrow{\mathrm{in}_X^{\sqcup}} X \sqcup Y \xrightarrow{f \sqcup \mathrm{id}_Y} Y.$$

Chapter 8

Discussion

8.1 Conclusion

We have elaborated, rephrased and formalized Garner's algebraic small object argument. We formalized the results, providing a computer verified proof for the construction. Additionally, we have also rephrased the argument using more modern theory like that of displayed categories and a modern notion of monoidal categories. Moreover, we filled in the gaps in the theory and made the results more direct, intuitive and accessible. Furthermore, we pointed out some constructive issues in the classical small object argument that Garner did not touch upon, mainly involving the possibly unsuspected use of the axiom of choice.

Let us briefly go over some of the main differences in the argument by Garner and this thesis. First and foremost, we filled the gaps which Garner left in his papers. The first thing that comes to mind was the lifting of the monoidal structure on $\mathbf{Ff}_{\mathcal{C}}$ to one on $\mathbf{LNWFS}_{\mathcal{C}}$. Something that Garner left out, but which took over 1000 lines of formalization and is in fact the file that takes the longest to check in the entire formalization. We also went over the transfinite construction in a lot more detail. We elaborated the construction, constructed the free monoid in a more intuitive and detailed way, following various examples.

Secondly, we introduced more modern language, in the form of displayed categories [11] and a more modern notion of monoidal categories [12] [13]. Furthermore, because of the limitations of the `UniMath` library, we often had to take a more direct or basic approach to proving certain statements, making the proofs more accessible, simple and often (much) more elaborate. This makes it so the argument can now be followed in more detail. To alleviate these limitations somewhat, we also formalized some more general theory, such as maps between coproducts using different indexing types (`CoproductOfArrowsInclusion`) or some theory on connected graphs (`connected_graph_zig_zag_strong_induction`).

Thirdly, we left out a lot of complex theory that Garner uses. This is, again, partly because of the way we are limited in the available results in `UniMath`, but it contributes to the accessibility of the proofs. Complex constructions such as twofold monoidal categories are left out, and the construction of the adjoint that Garner uses to obtain the free monoid from the transfinite sequence is replaced with a more direct and intuitive construction of the free monoid.

Finally, we hope to have made the idea behind the construction more clear from the start. The construction may be complex for a reader unfamiliar with the small object argument or the theory of (N)WFSs in general. This thesis attempts to give a more streamlined introduction to the theory of model categories, and mainly the small object argument.

The formalization gave more insight into the details of the theory, and it showed how few assumptions Garner's algebraic small object argument really needs. In the formalization of the algebraic small object argument, we never had to assume any categories to be univalent. Merely assuming we have small homsets is sufficient. The modern, HoTT compatible notion

of monoidal categories due to Ahrens, Matthes and Wullaert [12] [13] also proved to be sufficient for the construction. Moreover, we never used the univalence axiom in formalizing the algebraic small object argument. We did however use function extensionality, though it is not uncommon to assume this axiom. We primarily used this to show the equality of morphisms of functorial factorizations, but also in some other proofs. The formalization also pointed out constructive issues in the theory of plain weak factorization systems, mainly referring to the use of the axiom of choice.

8.2 Remarks on the Coq Formalization

At the end, the formalization was about 15000 LoC, and the entire formalization took about 9 minutes to compile on my (fairly old) laptop. Compiling the `LNWFSMonoidalStructure.v` file, containing the definitions for the monoidal structure on $\mathbf{LNWFS}_{\mathcal{C}}$, took up most of that time. There are some strategies in formalization that can greatly influence the compilation time. Some strategies that I learned while formalizing are the following.

- Use abstraction properly. That is to say, finish proofs with `Qed`, and not with `Defined` whenever you can. This makes it so the proof terms are saved as *opaque* terms, which cannot be unfolded in other proofs. Doing this makes the proof terms of other (possibly non-opaque) proofs become smaller in turn. When writing proofs that should be transparent (so which have to be finished off with `Defined`), one could either split out any subterms which are propositional into opaque lemmas, or use the `abstract` tactic to create an opaque term from within the proof. This strategy reduced the compilation time of the `FFMonoidalStructure.v` file from a couple of minutes down to only a few seconds.
- Do not leave `cbn`, `simpl` or `unfold` tactics in finished proofs. Though very useful when writing proofs, the proof checker will 're-type check' any unfolded terms upon saving the proof, greatly increasing the compilation time. Generally, unless specifically needing them to use the `rewrite` tactic, these tactics are *not* needed for the proof to work, and should only be used when actively writing the proof.
- This point specific to `UniMath`, and not for general Coq formalization: try to use `etrans` and `apply` instead of `rewrite`. The `rewrite` tactic produces very large terms, called `internal_paths_rew_r`, which are much larger terms than any terms `etrans` and `apply` create. Proofs involving the `rewrite` tactic produce much larger terms, and take much longer to check.
- Try to prove something more generic. For example, proving statements about the transfinite sequence would have become very slow, would we have shown the construction specifically for $\mathbf{LNWFS}_{\mathcal{C}}$. The more generic approach allowed us to use much simpler data and propositions. This did not only reduce the compilation time, but it also simplified a lot of goals, making the formalization process easier and quicker.
- Try to prove something more specific. For example, Garner introduces the theory of twofold monoidal categories. Though it may be a useful framework to get results on paper, setting up the theory would require much more work than showing the desired results directly.

  Another example is the `LNWFS_comon_structure_whiskercommutes` lemma, where we showed a property of the comonoidal structure $(\odot, \bot)$ on $\mathbf{Ff}_{\mathcal{C}}$ without explicitly introducing it. Though this proof is slow and tedious in its own right, defining this comonoidal structure would have taken much longer to do than just proving this single result.

  Of course, this point seems to contradict the previous point. It is important to decide when to use either of these strategies. This may be done by trial and error, but gen-

erally it is good to keep in mind how much extra data 'something specific' introduces, increasing the compilation time and possibly making goals less comprehensible. In contrast, it is good to consider how much extra work proving 'something more generic' entails.

When actively developing parts of the theory, it is good to split the proofs up into smaller parts. One could use the `admit` tactic, in combination with the `Admitted` vernacular to give up on certain proofs, and do them later. It may also be useful to use a lemma like

```
Lemma todo {A : UU} : A. Admitted.
```

which allows one to `apply todo` in any proof, giving up on that part of the proof, but not losing the transparency of the proof by still being able to finish it off with `Defined` instead of `Admitted`.

When formalizing theory, it is good to work in a combination of the formalization and on paper. Sometimes, one might recognize certain patterns in a goal that occur in other places as well, or one can simplify the proof into something simpler, simply by using common strategies. For example, when proving statements about colimits or pushouts, a proof involving the colimit could often be reduced to one involving only terms in the diagram. One should not get lost in formalization though. It is good to step back and really write out the goal you are trying to prove on paper to see where you are going with the formalization.

It is also good to keep in mind that a library like `UniMath` is not complete or perfect. Not all the proofs in the library are written optimally, with common use of the `rewrite` tactic, and many leftover `cbn`, `simpl` and `unfold` tactics. Fortunately, the library is open source, and one could contribute optimizations like this, or missing theory like maps between coproducts with different indexing types (`CoproductOfArrowsInclusion`) or some theory on connected graphs (`connected_graph_zig_zag_strong_induction`).

8.3 Future Work

There is still some obvious work to be done in the formalization. First of all, the smallness requirement on the morphism class J used to generate the one-step comonad can be further reduced from presentability of the arrows in J to presentability of the *domains* of the arrows in J.

Secondly, Garner's argument hypothesizes either of two smallness requirements, of which we have only shown the construction to work with the simpler one. The classical example of topological spaces does *not* satisfy this smallness requirement, though the example of simplicial sets does. It would be valuable to formalize this second smallness requirement as well.

The construction can also be generalized even further, by using more general ordinals, and not just the first limit ordinal ω. Unfortunately, more work has to be done on the theory of ordinals before being able to use them in this construction. We made a small effort to define ordinals using set truncations, as they are defined in the HoTT book [14], but this proved to be very cumbersome to work with, so we abandoned the effort.

Other than that, Garner's small object argument shows that the generated NWFS is in fact the *cofibrantly generated*, relating the R-**Map** class of the generated NWFS back to the morphism class J used to generate it. Some theory on this has already been formalized, but a lot is missing. If this is formalized, more examples could be worked out as well, most importantly in **SSET** and **TOP**.

Originally, the idea for the project was to also have a parallel development of the theory in Lean. This is still an interesting effort to pursue, since it may make the interaction between the theory and the foundations even clearer. Lean is built on a different Calculus

of Constructions than Coq and `UniMath`, also supporting classical reasoning and defining additional axioms, even having a built-in choice principle [19] [17].

Furthermore, it would be valuable to contribute the theory to `UniMath`. The formalization has been set up in a way that this should be easy to do, as it should only require us to rename a couple of import statements. This would be useful for any formalization efforts that build upon this theory.

One such effort is the theory of model 2-categories. In a conversation with PhD student Kobe Wullaert, he made clear that the formalization may be useful as inspiration or basis for him to formalize the theory of model 2-categories. This thesis and the accompanying formalization already give a good indication of how univalent foundations interact with the theory, and could be a good starting point for developing such theory.

Proof finished.

Bibliography

[1] D.G. Quillen. *Homotopical Algebra*. Lecture notes in mathematics. Springer-Verlag, 1967.

[2] M. Hovey. *Model Categories*. Mathematical surveys and monographs. American Mathematical Society, 2007.

[3] J P May and K Ponto. *More Concise Algebraic Topology: Localization, Completion, and Model Categories*. University of Chicago Press, Chicago, 2011.

[4] Jacob Lurie. Derived algebraic geometry iii: Commutative algebra, 2009.

[5] F. Déglise. Condensed and locally compact abelian groups. `https://deglise.perso.math.cnrs.fr/docs/2020/condensed.pdf`, 2020. Expansion of lectures on Condensed Mathematics by Peter Scholze.

[6] Philippe Gaucher. A model category for the homotopy theory of concurrency. *Homology, Homotopy and Applications*, 5(1):549–599, 2003.

[7] Julia E. Bergner. A survey of $(\infty, 1)$-categories, 2006.

[8] Marco Grandis and Walter Tholen. Natural weak factorization systems. *Archivum Mathematicum*, 42, 01 2006.

[9] Richard Garner. Understanding the small object argument. *Applied Categorical Structures*, 17(3):247–285, apr 2008.

[10] Emily Riehl. Algebraic model structures. *The New York Journal of Mathematics [electronic only]*, 17:173–231, 2011.

[11] Benedikt Ahrens and Peter Lefanu Lumsdaine. Displayed categories. 2018.

[12] Benedikt Ahrens, Ralph Matthes, and Kobe Wullaert. Formalizing monoidal categories and actions for syntax with binders, 2023.

[13] Ralph Matthes, Kobe Wullaert, and Benedikt Ahrens. Substitution for non-wellfounded syntax with binders, 2023.

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

[15] Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after voevodsky), 2018.

[16] nLab. `https://ncatlab.org/nlab/`. Various sub-pages.

[17] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[18] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath — a computer-checked library of univalent mathematics. Available at `http://unimath.org`.

[19] Lean. `https://lean-lang.org/`.

[20] Arne Strøm. The homotopy category is a homotopy category. *Arch. Math. (Basel)*, 23:435–441, 1972.

[21] Tom de Jong, Nicolai Kraus, Fredrik Nordvall Forsberg, and Chuangjie Xu. Set-theoretic and type-theoretic ordinals coincide. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, jun 2023.

[22] Richard Garner. Cofibrantly generated natural weak factorisation systems. 2007.

[23] Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

[24] Nicola Gambino, Christian Sattler, and Karol Szumiło. The constructive kan–quillen model structure: Two new proofs. *The Quarterly Journal of Mathematics*, 73(4):1307–1373, April 2022.

[25] G.M. Kelly. A unified treatment of transfinite constructions for free algebras, free monoids, colimits, associated sheaves, and so on. *Bulletin of the Australian Mathematical Society*, 22(1):1–83, 1980.

[26] Peter T Johnstone. *Sketches of an elephant: a Topos theory compendium*. Oxford logic guides. Oxford Univ. Press, New York, NY, 2002.