

A Formalization of Functor Quasi-Categories in Lean 4

by

Jack McKoen

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Mathematics

Department of Mathematical and Statistical Sciences

University of Alberta

© Jack McKoen, 2026

Abstract

This thesis presents a complete formalization, in the Lean 4 proof assistant, of the result that the internal hom between quasi-categories, known as the functor quasi-category, is itself a quasi-category. Quasi-categories provide a combinatorial model for ∞ -categories, and the functor quasi-category models the ∞ -category of functors between them.

Toward this goal, and following the proofs presented by Lurie in [15] and Rezk in [21], many non-trivial results in simplicial homotopy theory are formalized regarding simplicial sets, lifting properties, saturated collections of morphisms, and inner anodyne maps.

This work represents one of the first formalizations of quasi-category theory within an interactive theorem prover. It establishes a foundation for further development of ∞ -category theory in Lean and constitutes a key milestone toward the formalization of the ∞ -cosmos of quasi-categories, as defined by Riehl and Verity in [24].

Preface

This thesis is an original work by Jack McKoen, supported by a Canada Graduate Research Scholarship from the Natural Sciences and Engineering Research Council of Canada. No part of this thesis has been previously published.

Acknowledgements

First and foremost, I would like to thank Adam Topaz for his supervision, for introducing me to formalized mathematics, and for always supporting my mathematical interests. I thank Joël Riou for his consistent guidance and inspiration. I thank Rindra Razafy for being a tireless friend and accompanying me every step of the way. Finally, I thank everyone who stepped foot in my office, CAB 548.

Table of Contents

1	Introduction	1
1.1	Motivating Formalization	1
1.2	Motivating Higher Categories	2
1.3	Outline	5
2	Lean	7
2.1	Mathlib	9
2.2	Universes and Type Classes	9
2.3	Dependent Types	10
2.3.1	Dependent Functions	10
2.3.2	Dependent Products	12
2.4	Inductive Types	14
2.5	An Example Formalization	16
2.5.1	(Co)products of Presheaves	16
3	Simplicial Sets	19
3.1	Definitions and Examples	19
3.2	Simplicial Subsets	21
3.2.1	Examples of Subsets	21
3.3	Quasi-Categories as ∞ -Categories	24
3.4	The Functor Quasi-Category	30
4	Lifting Properties and Saturated Collections	34
4.1	Lifting Problems	34
4.2	Saturated Collections of Morphisms	35

4.3	Adjunction of Lifting Properties	41
4.3.1	Interactions with Saturated Collections	44
5	Monomorphisms of Simplicial Sets	47
5.1	Non-Degenerate Simplices	48
6	Inner Anodyne Morphisms	51
6.1	Generating Inner Horn Inclusions	53
6.2	Generating Pushout-Products	55
6.2.1	Non-Degenerate Simplices of $\Delta^{n+1} \times \Delta^2$	56
6.2.2	Constructing a Filtration	60
6.2.3	Properties of the Filtration	61
7	Conclusion	76
7.1	Functor Quasi-Categories are Quasi-Categories	76
7.2	Further Work	79

Chapter 1

Introduction

This thesis describes a complete proof and formalization of a result from ∞ -category theory. In this chapter, we introduce the concept and purpose of computer-formalized mathematics before briefly motivating ∞ -category theory. Finally, we explain the motivation and goal of this thesis in particular, and present an outline of the rest of the thesis.

1.1 Motivating Formalization

The process of formalizing mathematics involves translating mathematical definitions, theorems, and proofs into a formal language in a precise and verifiable way. Proof assistants, also called interactive theorem provers, allow users to construct formal proofs with the assistance of computer software. The proof assistant used in this thesis is Lean [20], specifically its fourth major version, Lean 4. Lean functions both as a programming language and as a proof system: Lean code corresponds to the logical steps of a proof, which are then translated into a formal language and automatically checked for correctness. To trust the correctness of a Lean formalization, one needs only to trust the Lean kernel itself and that the intended mathematical statement has been encoded faithfully.

The purpose of formalization is manifold. The most immediate benefit is verification: formal proofs assure correctness and have even been used to

verify research papers prior to peer review [8, 18]. Formalization may also expose gaps (usually minor) in accepted mathematics [3]. Beyond verification, formalization serves a pedagogical role: formal libraries of mathematics, such as Lean’s Mathlib [17], serve as an “interactive textbook” which offers quick access to known results and their proofs. Collaboration is another significant advantage: the modular nature of formalization lends itself to group efforts, and large-scale collaborative projects, such as Terence Tao’s Equational Theories Project [26], would be nearly impossible to coordinate through traditional means. Many of these ideas and their implications are discussed in Patrick Massot’s essay on formalization [16].

A central goal of the formalized mathematics community is for formal libraries to reach the level of contemporary research. From there, new results can be attainably formalized to accrue all of the aforementioned benefits. It is also hoped that this will make the discovery of new mathematics easier: researchers may query libraries for existing results, or use proof assistants to assist with proofs via automation or similar AI tooling. Major collaborative projects such as the completed Liquid Tensor Experiment [25], the ongoing formalization of Fermat’s Last Theorem [3], and the ongoing ∞ -cosmos project [23] were all started with the aim to bring Lean-formalized mathematics closer to research-level mathematics. The purpose of this thesis is aligned with that same goal: to advance the formalized foundations of ∞ -category theory, bringing them closer to the level required for contemporary mathematics.

1.2 Motivating Higher Categories

∞ -categories are meant to generalize classical categories to account for “higher-dimensional” structures. In particular, ∞ -category theory aims to extend category theory to meet the needs of homotopy theory. In a topological space X , we have points, paths between points, homotopies between paths, homotopies between homotopies, and so on. Similarly, in an ∞ -category, we want objects, 1-morphisms between objects, 2-morphisms between 1-morphisms, and so on, with $(n + 1)$ -morphisms between n -morphisms for all $n \in \mathbb{N}$.

The fundamental ∞ -groupoid of a topological space X , $\Pi_\infty X$, has the

general structure we expect from an ∞ -category: its objects are points of X , its 1-morphisms are paths, its 2-morphisms are homotopies between paths, and its $(n + 1)$ -morphisms are homotopies between n -morphisms. However, this structure is too strict. Indeed, $\Pi_\infty X$ is an ∞ -groupoid, because all of its n -morphisms are equivalences. We want to relax this structure for general ∞ -categories, and defining them as quasi-categories provides a natural approach.

Quasi-categories, first developed by Boardman and Vogt as “weak Kan complexes” [2] and further developed by Joyal [11–13] and Lurie [14, 15], are a convenient model for ∞ -categories. Kan complexes are a model for ∞ -groupoids, and they are defined as simplicial sets (particular presheaves of sets) which satisfy a “horn-filling” condition. Quasi-categories are simplicial sets satisfying a relaxed version of this “horn-filling” condition, so we view quasi-categories as a natural generalization of Kan complexes, and our ∞ -categories as a natural generalization of ∞ -groupoids.

In Chapter 3, we briefly discuss how quasi-categories lend themselves to a higher categorical framework, but a full treatment lies beyond the scope of this thesis. A comprehensive development of quasi-categories and their relationship to higher category theory can be found in the works of Lurie [14, 15], Rezk [21], or Cisinski [5]; a key takeaway is that we can work with quasi-categories in a manner analogous to classical categories. This thesis explains part of that analogy and presents a detailed formalization of it.

In classical category theory, given two categories \mathcal{C} and \mathcal{D} we obtain the functor category, denoted by $\text{Fun}(\mathcal{C}, \mathcal{D})$ or $\mathcal{D}^{\mathcal{C}}$, whose objects are functors $\mathcal{C} \rightarrow \mathcal{D}$ and whose morphisms are natural transformations between such functors. In the ∞ -categorical setting, we similarly expect an ∞ -category of functors between two ∞ -categories—and indeed, such a construction exists. Using the language of simplicial sets, it is straightforward to define a simplicial set which behaves like an object of functors between two quasi-categories; however, it is non-trivial to prove that this construction is a quasi-category. This leads to the definition of the functor quasi-category. Specifically, this thesis culminates in the following formalized theorem:

Theorem. *If X and Y are quasi-categories, then the internal hom between X and Y is a quasi-category.*

The internal hom between X and Y , to be defined and formalized in Chapter 3, serves as the functor quasi-category between X and Y . A stronger form of the above theorem is proved in Chapter 7 as Theorem 7.1.1 and is formalized in Lean at the following link as the statement `ihom_isQuasicategory`: [↗_{JM}](#). In this linked file, the command `#print axioms ihom_isQuasicategory` allows anyone to check the axioms relied upon in this statement—in particular, this verifies that the formalized proof does not have any gaps [7, 8.5]. Chapter 2 introduces Lean and says more about how to read Lean code.

This thesis presents one of the first formalizations of quasi-category theory within an interactive theorem prover, and for this reason, defining the functor quasi-category was a reasonable first goalpost. Following an approach derived from [15, Tag 0066] and [21, Thm. 22.4], the construction necessitates a broad formalization of quasi-categorical language which directly benefits future ∞ -categorical formalization projects and provides explicit details to proofs which are often “black boxed” or left opaque in the literature (as discussed in Chapter 6).

Additionally, just as the functor category is an essential construction for many interesting applications (like the Yoneda lemma), the same is true for the functor quasi-category; the ∞ -categorical versions of the Yoneda lemma [15, Tag 03M2], adjunctions [15, Tag 02EJ], (co)limits [15, Tag 02H0], and more all depend on its definition. Moreover, the notion of an ∞ -cosmos, an approach to higher categories developed by Riehl and Verity [24], emphasizes the importance of the functor quasi-category by generalizing it: an ∞ -cosmos is a category “enriched over quasi-categories” satisfying some additional axioms [24, Def. 1.2.1]. The ∞ -cosmos of quasi-categories serves as the prototypical example, making this formalization essential to the ongoing ∞ -cosmos formalization project—a formalization of ∞ -cosmos theory in Lean 4 [23].


Therefore, the functor quasi-category is both a technically and conceptually natural choice for formalization.

1.3 Outline


This thesis is structured with goal of proving Theorem 7.1.1. To that end, we introduce just enough theory, sequentially, to accomplish this. The following is an outline:

- Chapter 2 is an introduction to Lean with a view toward the formalizations presented in further chapters.
- Chapter 3 is an overview of the language and theory of simplicial sets to be used throughout the rest of this thesis. This is where quasi-categories and functor quasi-categories are defined.
- Chapter 4 develops lifting properties and how they interact with the quasi-category condition.
- Chapters 5 and 6 develop enough theory of simplicial sets to tackle Theorem 7.1.1. Specifically, Chapter 5 investigates monomorphisms and Chapter 6 concerns inner anodyne morphisms.
- Chapter 7 concludes this thesis with a proof that the functor quasi-category is a quasi-category. Immediate consequences of this formalization and comments on further work are discussed.

Throughout this thesis, most definitions and statements will be accompanied by a link to a Lean-formalized equivalent hosted on GitHub, according to the following formatting:

- _{JM} denotes a link to a formalization by the author, either in Mathlib or in the dedicated repository for this thesis, which can be found at <https://github.com/mckoen/quasicategory>.

All references to this repository are to a version from October 2025 (commit `c1069f678377651da8ea1a4a07b2f9a1008cc8c9`).

-  denotes a link to a formalization which is not by the author. This will either be a link to Mathlib, Lean's core library, or Joël Riou's repository for his work towards formalizing the Quillen model category structure on

simplicial sets, which can be found at <https://github.com/joelriou/topcat-model-category>.

All references to Mathlib are to a version from March 2025 (commit `f816d2aac7e4ba072d493b3aa440f987cdbc2be6`), which is the version relied upon in this thesis. The authors of such a file are listed at the top of the file.

All references to Lean's core library are to a version from October 2025 (commit `66466a7b5db73989b88c3d74cf8c68f4f11991b0`). The authors of such a file are listed at the top of the file.

All references to Joël Riou's repository are to a version from April 2025 (commit `4b4b46eda5c6038efa64d7c70e9c46cfb6e5b6dd`), which is the version relied upon in this thesis.

All explicitly written code is by the author unless otherwise stated.

Chapter 2

Lean

Lean is a proof assistant and functional programming language based on dependent type theory and the calculus of inductive constructions [20]. A detailed and accessible introduction to Lean and the reading of Lean code can be found in Anne Baanen’s PhD thesis [1, Ch. 2], and the standard reference text is Theorem Proving in Lean by Avigad et al. [10].

A brief overview is given in this chapter. In simple type theory, every expression t has an associated *type* T . We write $t : T$ and say t is a *term* of type T . For example:

- If n is a natural number, then $n : \mathbb{N}$.
- If f is a function between types A and B , then $f : A \rightarrow B$.
- If $a : A$ and $f : A \rightarrow B$, then $f(a) : B$.
- $\mathbb{N} : \text{Type}$, where Type is the type universe of “small” types (see Section 2.3).

Throughout this thesis, we will generally view terms of Type as analogous to sets, and Type as analogous to the category **Set**. Indeed, within Type we have functions, products, disjoint unions, and more, all of which behave as expected under this analogy.

A distinguished type is $\text{Prop} : \text{Type}$, the type of propositions. A term $P : \text{Prop}$ represents a proposition, and a term $p : P$ represents a proof of that

proposition. This correspondence between propositions and types is known as the *Propositions-as-Types* principle [27, I.1.11], or the Curry–Howard correspondence. Under this correspondence, propositions correspond to types, and proofs correspond to terms; to prove a proposition is to construct a term of the corresponding type.

In Lean, then, constructing proofs is literally constructing terms. *Tactics* are tools that assist in this construction. For example:

```
def exampleDef (a : ℝ) : ℝ → ℝ := fun b ↦ b + a

lemma exampleLemma (a : ℝ) : 2 * a = exampleDef a a := by
  rw [two_mul]
  rfl
```

The keywords `def` and `lemma` introduce definitions and lemmas, respectively. The parenthesized terms specify parameters (variables). Given some $a : \mathbb{R}$, the expression `exampleDef a` has type $\mathbb{R} \rightarrow \mathbb{R}$, while `exampleLemma a` is a term of the proposition $2 * a = \text{exampleDef } a \ a$, that is, a proof of that equality.

The keyword `fun` defines a function, so `exampleDef a : ℝ → ℝ` is defined by $b \mapsto b + a$. The keyword `by` in `exampleLemma` switches Lean into *tactic mode* [10, Ch. 5], in which proofs can be constructed interactively using tactics. The tactic `rw` (rewrite) applies the existing lemma `two_mul`, which asserts that $2 * a = a + a$, leaving the goal $a + a = \text{exampleDef } a \ a$. This is true by definition (the terms are *definitionally equal*), which the tactic `rfl` (reflexivity) observes, and this completes the proof.

This simple example illustrates how mathematics is generally done in Lean: proofs are terms, and tactics are steps for constructing them. The remainder of this chapter examines how more complex mathematical structures are represented in Lean.

2.1 Mathlib

Mathlib [17] is Lean’s mathematical library and the foundation for nearly all formalization projects in Lean. As of October 2025, it contains nearly two million lines of code [6], encompassing a broad range of mathematical areas. Importantly, Mathlib contains a comprehensive sublibrary of category theory.

Sections 2.2 to 2.4 briefly cover three major aspects of Lean as they relate to examples drawn from Mathlib and from formalizations developed in this thesis.

2.2 Universes and Type Classes

In dependent type theory, types are terms too—every type has a type:

$\mathbb{N} : \text{Type}$, $\text{Type} : \text{Type } 1$, $\text{Type } 1 : \text{Type } 2$, $\text{Type } 2 : \text{Type } 3$, and so on.

There is a “type hierarchy” indexed by natural numbers, where $\text{Type } n : \text{Type } n + 1$. We can think of these as nested universes of larger and larger types, which avoids Russell’s paradox. With this in mind, we consider how categories may be defined in Lean:

In Mathlib, a category is defined as a collection of objects in some type universe $\text{Type } u$, together with a collection of morphisms in a (possibly different) universe $\text{Type } v$. In Mathlib [↗](#), this is expressed by declaring a type

$$(\mathbb{C} : \text{Type } u)$$

and an *instance*

$$[\text{Category}.\{v\} \mathbb{C}].$$

The square brackets indicate that `Category` is a *type class* [10, Ch. 10]: a mechanism for bundling the data of a structure and retrieving it automatically when needed. The `Category` type class handles all of the data defining the category structure on \mathbb{C} , including the morphisms and proofs of the category axioms.

Type classes make such arguments implicit. When a definition or theorem requires a `[Category.{v} C]` parameter, Lean’s *type class resolution* system [10, Ch. 10] searches for an instance automatically. For example, if `C` is known to be a poset, Lean can synthesize (through Mathlib) a `Category` instance on `C` without explicit user input [↗](#). Likewise, if a category structure on `C` has already been declared or defined, Lean infers it automatically.

More broadly, algebraic structures such as groups and rings, as well as order-theoretic, analytic, or topological structures, are all implemented through the type class system. This design allows formalized mathematics in Lean to mirror the implicit conventions of informal mathematics: once a structure on an object is known, it does not need to be restated.

2.3 Dependent Types

Lean is based on dependent type theory, which is called *dependent* because “types can depend on parameters”, as detailed in [10, Sec. 2.8]. The following is a brief summary of two important examples of dependent types: dependent functions and dependent products.

2.3.1 Dependent Functions

Given $\alpha : \text{Type}$ and $\beta : \alpha \rightarrow \text{Type}$, we can think of β as a family of types $\beta(a)$ parameterized by the terms a of α . A *dependent function type*

$$(a : \alpha) \rightarrow \beta(a)$$

is a dependent type which can be thought of as the family of functions f from α such that for each term a of α , $f(a)$ is a term of $\beta(a)$. This generalizes functions $\alpha \rightarrow \beta$ by allowing β to depend on α [10, Sec. 2.8].

The universal quantifier is an example of a dependent function type: Let $\alpha : \text{Type}$ and replace β above with $p : \alpha \rightarrow \text{Prop}$. If for every $x : \alpha$, there is a proof of $p(x)$, i.e. $\forall (x : \alpha), p(x)$, then this specifies exactly how to construct

a dependent function $(x : \alpha) \rightarrow p(x)$. In Lean, the syntax

$$\forall (x : \alpha), p\ x$$

is equivalent to

$$(x : \alpha) \rightarrow p\ x.$$

We illustrate this with the following example. In Mathlib, a collection of morphisms in a category is typically defined via a “morphism property”, as formalized below [↗](#):

```
def MorphismProperty (C : Type u) [Category.{v} C] :=
  ∀ {X Y : C} (f : X → Y), Prop
```

We interpret this as a collection of morphisms characterized by a proposition. Viewed as a dependent function type, the type of `MorphismProperty C` is

$$\{\{X\ Y : C\}\} \rightarrow (X \rightarrow Y) \rightarrow \text{Prop}.$$

The curly braces signify that `X Y : C` are *implicit* variables, meaning Lean should figure out what they are based on the morphism `X → Y`. Given this translation, the above defines a morphism property on a category \mathcal{C} as a function which takes a morphism $f : X \rightarrow Y$ in \mathcal{C} and returns a proposition (typically) depending on f . For example, the collection of monomorphisms in a category \mathcal{C} is formalized as follows [↗](#):

```
def monomorphisms (C : Type u) [Category.{v, u} C] :
  MorphismProperty C := fun _ _ f => Mono f
```

`Mono f : Prop` is the proposition that says `f` is a monomorphism, so a term of `Mono f` is a proof that `f` is a monomorphism. The `fun` keyword defines a function, so `monomorphisms C` is the dependent function which maps a morphism `f` in \mathcal{C} to the proposition that `f` is a monomorphism. A term `h : monomorphisms C f` is the result of mapping `f` by this function, i.e. `h : Mono f`. Therefore, this morphism property characterizes all monomorphisms.

2.3.2 Dependent Products

Another example of a dependent type is a *dependent (cartesian) product*, which generalizes the cartesian product $\alpha \times \beta$ by allowing β to depend on α [10, Sec. 2.8]. For $\alpha, \beta : \text{Type}$, terms of the product type $\alpha \times \beta$ are pairs (a, b) with $a : \alpha$ and $b : \beta$. If instead $\beta : \alpha \rightarrow \text{Type}$, the dependent product

$$(a : \alpha) \times \beta(a), \quad \text{or} \quad \Sigma (a : \alpha), \beta(a)$$

consists of all pairs (a, b) such that $a : \alpha$ and $b : \beta(a)$.

The following is an example that makes a good case for the utility of a dependent product and will come up in Chapter 6.

Let $0 \leq a \leq b \leq n$ for some $n \in \mathbb{N}$. Suppose we want to work with all such pairs (b, a) for a fixed n . In set theory, we might write this as

$$\{(b, a) \in \mathbb{N} \times \mathbb{N} : 0 \leq a \leq b \leq n\} = \bigcup_{0 \leq a \leq b \leq n} \{(b, a)\} \subset \mathbb{N} \times \mathbb{N},$$

but in dependent type theory, we can express this as the *dependent product*

$$(b : \text{Fin}(n + 1)) \times \text{Fin}(b + 1)$$

where $\text{Fin } n = \{0, 1, \dots, n - 1\}$ is a finite type with exactly n terms. A term of $\text{Fin}(n + 1)$ is a natural number b together with a proof that $b < n + 1$; thus, terms of this dependent product are exactly the pairs (b, a) with $0 \leq a \leq b \leq n$. There are obvious notational advantages to this approach, but there are also advantages in its implementation: a dependent (binary) product is a simple type with two constructors, but the subset of $\mathbb{N} \times \mathbb{N}$ defined above is much messier to define within type theory.

Another advantage of this dependent product—and the main advantage relied upon in Chapter 6—is that it’s simple to define an order on it:

Definition 2.3.1. \hookrightarrow Let $i, j : (b : \text{Fin}(n + 1)) \times \text{Fin}(b + 1)$. Let $(b, a) = i$ and let $(b', a') = j$. The lexicographic order is defined by $i \leq j \iff b < b'$ or $b = b'$ and $a \leq a'$.

In Lean, we write $\Sigma_l (b : \text{Fin } (n + 1)), \text{Fin } (b + 1)$ to denote the dependent product with the lexicographic order. The Σ is because this order is defined for more general Σ -types.

In Chapter 6, we will define a few filtrations which are indexed by similar dependent products. Further, we can define a lexicographic successor which makes it simple to show that these filtrations are monotone.

Definition 2.3.2. $\hookrightarrow_{\text{JM}}$ Let $i : (b : \text{Fin } (n + 1)) \times \text{Fin } (b + 1)$. Let $(b, a) = i$. Define the lexicographic successor function,

$$\text{succ} : (b : \text{Fin } (n + 1)) \times \text{Fin } (b + 1) \rightarrow (b : \text{Fin } (n + 1)) \times \text{Fin } (b + 1),$$

by

$$\text{succ } i = \begin{cases} (b, a + 1) & \text{if } a < b \\ (b + 1, 0) & \text{if } a = b < n \\ (n, n) & \text{if } a = b = n \end{cases}$$

In Lean, this is written as

```
def succ : (Σ_l (b : Fin (n + 1)), Fin (b + 1)) →
  (Σ_l (b : Fin (n + 1)), Fin (b + 1)) :=
fun ⟨b, a⟩ ↦
  if h₁ : a.1 < b then ⟨b, ⟨a + 1, ...⟩⟩
  else if h₂ : b = Fin.last n then ⟨Fin.last n, Fin.last n⟩
  else ⟨⟨b + 1, ...⟩, ⟨0, ...⟩⟩
```

(with ellipses to omit code fragments for clarity). This means that the lexicographic order on $(b : \text{Fin } (n + 1)) \times \text{Fin } (b + 1)$ is given by

$$(0, 0) < (1, 0) < (1, 1) < (2, 0) < (2, 1) < \dots < (n, 0) < \dots < (n, n),$$

with each pair in the sequence being the successor of the pair before it. This successor function satisfies every property you would want a successor to satisfy: $i \leq \text{succ } i$, $\text{succ } i \leq i$ if and only if $i = \top$ (the top element, i.e. (n, n)), and $i < j$ implies $\text{succ } i \leq j$. In fact, these requirements go toward proving that the lexicographic order defines an instance of `SuccOrder` \hookrightarrow on

$\Sigma_l (b : \text{Fin } (n + 1)), \text{Fin } (b + 1) \curvearrowright_{\text{JM}}$: a type class for an order which has a sensible successor function.

2.4 Inductive Types

An inductive type is a type which is built up from a specified list of constructors [10, Ch. 7]. In Lean, the prototypical example is the natural numbers \curvearrowright :

```
inductive Nat where
  | zero : Nat
  | succ (n : Nat) : Nat
```

A natural number is either zero or the successor of another natural number. Thus, there are only two constructors for `Nat`, corresponding to each case. In fact, an inductive type contains only objects constructed from its specified constructors. When proving a statement about a term `n : Nat`, we can perform induction on `n` to replace the goal of our proof with two separate goals specified by the constructors: one where `n` is replaced by zero and one where `n` is replaced by a successor, equipped with the appropriate inductive hypothesis. We can similarly perform induction on any inductive type.

The following provides some motivation for using inductive types. Suppose we want to define a `MorphismProperty` which is meant to be the collection containing a single morphism `p : X → Y`. We will generalize this idea in Chapters 5 and 6 to collections of specific morphisms. A `MorphismProperty` is a collection of morphisms characterized by a proposition, so a naive approach would be the following:

```
def morphism {X Y : C} (p : X → Y) : MorphismProperty C :=
  fun _ _ f ↦ f = p
```

The intended meaning is that `morphism p` should be the collection of all morphisms `f` that are equal to `p`. However, this definition does not “typecheck”: in Lean, `f = p` is only meaningful if `f` and `p` have the same type. Since `f` is an arbitrary morphism and `p : X → Y` has fixed domain and codomain, the two types generally differ. Thus, equality in this form cannot be expressed directly. Analogously to how equality of objects is not the right notion of

equivalence in category theory, this form of equality is too rigid in type theory. One could attempt to include proofs that the domain and codomain of f coincide with those of p , but this approach quickly becomes cumbersome.

Instead, we can define an inductive predicate that characterizes the morphism p without appealing to direct equality \checkmark_{JM} :

```
inductive Morphism {X Y : C} (p : X → Y) : {A B : C} → (A →
  B) → Prop
| mk : (Morphism p) p
```

For any morphism $f : A \rightarrow B$ in C , the type `Morphism p f` consists of the propositions generated by its single constructor `mk`. Just as every natural number reduces inductively to `succ` or `zero`, any term of `Morphism p f` reduces f , by induction, to the specific morphism p . A term of `Morphism p f` implies that f is *heterogeneously* equal to p , which is closer to the form of equality we want, but is still cumbersome to work with—it is better to inductively replace f with p and avoid notions of equality at all.

To understand this construction, compare it to the inductive definition of the existential quantifier \exists :

```
inductive Exists {α : Sort u} (p : α → Prop) : Prop where
| intro (w : α) (h : p w) : Exists p
```

Here, given a predicate $p : \alpha \rightarrow \text{Prop}$, `Exists p` is an inductive proposition with one constructor taking a witness $w : \alpha$ and a proof $h : p w$. Every term of `Exists p` reduces to such a pair, and conversely, any such pair constructs an instance of existence.

Similarly, a term of `Morphism p f` reduces f to p , and conversely, only a term of `Morphism p p` can be constructed. Using this inductive predicate, we can now define the desired morphism property \checkmark_{JM} :

```
def morphism {X Y : C} (p : X → Y) : MorphismProperty C :=
  fun _ _ f ↦ morphism p f
```

This definition captures the intended meaning: `morphism p` represents the collection of all morphisms that are (in the appropriate sense) equal to the given morphism p .

2.5 An Example Formalization

Functors are defined in Mathlib in the following way [↗](#):

```
structure Functor (C : Type u1) [Category.{v1} C] (D : Type u2)
  [Category.{v2} D] : Type max v1 v2 u1 u2 where
  obj : C → D
  map : ∀ {X Y : C}, (X → Y) → ((obj X) → (obj Y))
  map_id : ∀ X : C, map (1 X) = 1 (obj X) := by aesop_cat
  map_comp : ∀ {X Y Z : C} (f : X → Y) (g : Y → Z),
    map (f >> g) = map f >> map g := by aesop_cat
```

The keyword **structure** indicates the definition of a structure, which is a specific kind of inductive type (see [10, Ch. 9]). The **Functor** structure has four fields: **obj** specifies how objects in **C** are mapped to objects in **D**, **map** specifies how morphisms in **C** are mapped to morphisms in **D**, **map_id** ensures that **map** takes identity morphisms (denoted by **1**) to identity morphisms, and **map_comp** ensures that **map** respects composition (denoted by **>>**). At the end of the **map** and **map_id** fields, we see **:= by aesop_cat**. This means that when constructing a **Functor**, Lean’s automation will try to automatically complete the **map** and **map_id** proofs, based on **obj** and **map**, by applying **aesop_cat**, a tactic which tries to automatically complete simple category theory proofs.

Simplicial sets, the central object of study in this thesis, are certain presheaves of sets. As such, we need to work with **Functor**’s a lot. When this project began, Mathlib already contained extensive API for **Functor**’s, but it lacked some explicit results concerning (co)products of presheaves. The following section describes how this gap was filled and illustrates the kind of considerations that aren’t often made in informal mathematics, but are required in formalization.

2.5.1 (Co)products of Presheaves

A presheaf of sets is a contravariant functor valued in **Set**. We work explicitly with finite products of presheaves throughout this thesis, so we present here a formalization of (co)products of **Set**-valued functors more generally. Since

Lean is based on type theory, the concept of a **Set**-valued functor $\mathcal{F} : \mathcal{C} \rightarrow \mathbf{Set}$ is replaced with a functor $\mathcal{F} : \mathcal{C} \rightarrow \text{Type } w$. One can conclude that the category of functors $\mathcal{F} : \mathcal{C} \rightarrow \text{Type } w$ has (co)products since $\text{Type } w$ does $\hookrightarrow \hookrightarrow$ (it is helpful to think of $\text{Type } w$ as a category analogous to **Set**), and functor categories inherit (co)limits from their target category \hookrightarrow . However, this says nothing about the explicit “pointwise” formula for a (co)product of such functors, which is necessary for many proofs regarding simplicial sets. This highlights a major difference between formalized and informal mathematics. On paper, such details are typically omitted: one writes $\mathcal{F} \times \mathcal{G}$ and implicitly understands it to be defined pointwise.

The product of $\mathcal{F}, \mathcal{G} : \mathcal{C} \rightarrow \text{Type } w$ is given by $(\mathcal{F} \times \mathcal{G})a := \mathcal{F}a \times \mathcal{G}a$ where on the right we have a product of types. In Lean $\hookrightarrow_{\text{JM}}$:

```
def prod (F G : C => Type w) : C => Type w where
  obj a := F.obj a × G.obj a
  map f a := (F.map f a.1, G.map f a.2)
```

Note that the `map_id` and `map_comp` fields were automatically completed by Lean’s automation and are not necessary to write down. Similarly, the coproduct is given by $(\mathcal{F} \sqcup \mathcal{G})a := \mathcal{F}a \sqcup \mathcal{G}a$ where on the right we have a disjoint union of types. In Lean $\hookrightarrow_{\text{JM}}$:

```
def coprod : C => Type w where
  obj a := F.obj a ⊕ G.obj a
  map f x := by
    cases x with
    | inl x => exact .inl (F.map f x)
    | inr x => exact .inr (G.map f x)
```

The \oplus symbol denotes a disjoint union of types. When defining `map`, we take a term $x : \mathcal{F}.obj\ a \oplus \mathcal{G}.obj\ a$ and use the tactic `cases` on it. This is because $\mathcal{F}.obj\ a \oplus \mathcal{G}.obj\ a$ is an inductive type with two constructors, `inl` and `inr`, which specify whether a term belongs to $\mathcal{F}.obj\ a$ or $\mathcal{G}.obj\ a$.

It is then necessary to show that `prod` and `coprod` are actually isomorphic to the categorical product and coproduct, respectively. This is concluded after

showing that they are (co)limiting (co)cones over the binary (co)fan diagram (i.e. they satisfy the same universal property) $\begin{array}{c} \nearrow \\ \text{JM} \end{array}$ $\begin{array}{c} \nearrow \\ \text{JM} \end{array}$:

```
def binaryProductIso : F × G ≅ prod F G :=
  limit.isoLimitCone (binaryProductLimitCone F G)
```

```
def binaryCoproductIso : F ⊔ G ≅ coprod F G :=
  colimit.isoColimitCocone (binaryCoproductColimitCocone F G)
```

Here `binaryProductLimitCone` is a proof that the cone with point `prod F G` for the binary fan given by `F` and `G` is a limit cone, and `limit.isoLimitCone` says that limit cones over the same diagram have isomorphic points. Therefore, $F \times G \cong \text{prod } F \ G$. The dual argument shows that $F \sqcup G \cong \text{coprod } F \ G$.

This identification allows us to work interchangeably with abstract finite (co)products of simplicial sets and their explicit pointwise definitions.

Chapter 3

Simplicial Sets

Simplicial sets are the fundamental objects of study in this thesis. This chapter defines simplicial sets and basic terminology before discussing simplicial subsets, which many distinguished simplicial sets are examples of. Finally, we define Kan complexes, quasi-categories, and functor quasi-categories. Many of the statements in this chapter are adapted from [21, 22].

3.1 Definitions and Examples

Definition 3.1.1. \hookrightarrow The *simplex category* Δ is defined as follows:

- its objects are finite, non-empty sets $[n] := \{0, 1, \dots, n\}$
- its morphisms are monotone functions $[n] \rightarrow [m]$

Notation 3.1.2. If $\delta : [n] \rightarrow [m]$, it is often helpful to denote this map by a list of its values [21, Rem. 2.2]: we write $\delta = \langle \delta_0, \dots, \delta_n \rangle$ with $0 \leq \delta_0 \leq \dots \leq \delta_n \leq m$ to represent the function $k \mapsto \delta_k$.

Definition 3.1.3. For every $n \in \mathbb{N}$, there are two distinguished maps in the simplex category:

- \hookrightarrow the *face map* $d^i = \langle 0, 1, \dots, \hat{i}, \dots, n+1 \rangle : [n] \rightarrow [n+1]$ omits i in its image, and

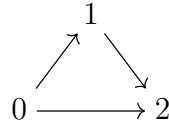
- \hookrightarrow the *degeneracy* map $s^i = \langle 0, 1, \dots, i, i, \dots, n \rangle : [n+1] \rightarrow [n]$ duplicates i in its image.

Definition 3.1.4. $\hookrightarrow \hookrightarrow$ Given a category \mathcal{C} , a *simplicial \mathcal{C} -object* is a functor $\Delta^{\text{op}} \rightarrow \mathcal{C}$. A *simplicial set* is a functor $\Delta^{\text{op}} \rightarrow \mathbf{Set}$. The category of simplicial sets is denoted by \mathbf{SSet} or $\mathbf{Set}^{\Delta^{\text{op}}}$.

Notation 3.1.5. Given a simplicial set X , we denote $X[n]$ by X_n and refer to this set as the *n -simplices* of X .

Definition 3.1.6. \hookrightarrow The *standard n -simplex* Δ^n is defined to be the Yoneda embedding of $[n]$, that is, the functor $\text{Hom}_{\Delta}(-, [n])$. Its k -simplices are the order-preserving maps $[k] \rightarrow [n]$.

The standard n -simplex is a fundamental object in the study of simplicial sets. One interprets Δ^n as analogous to the topological n -simplex, which is the *geometric realization* of Δ^n . This geometric perspective of simplicial sets is discussed in detail in [5, 15, 21]. We won't develop this perspective, but we heavily rely on it for intuition. For example, the 2-simplex Δ^2 is visualized as follows:



The face and degeneracy maps $d^i : [n] \rightarrow [n+1]$ and $s^i : [n+1] \rightarrow [n]$ induce maps $\Delta^n \rightarrow \Delta^{n+1}$ and $\Delta^{n+1} \rightarrow \Delta^n$: We think of $\Delta^n \rightarrow \Delta^{n+1}$ as the inclusion of the n -simplex as a face of the $(n+1)$ -simplex, and we think of $\Delta^{n+1} \rightarrow \Delta^n$ as collapsing the $(n+1)$ -simplex onto one of its faces.

Theorem 3.1.7. $\hookrightarrow_{\text{JM}}$ Δ^0 is terminal in the category of simplicial sets.

Proof. This follows from the fact that $[0]$ is terminal in Δ . □

Remark 3.1.8. \hookrightarrow By the Yoneda lemma, the n -simplices of a simplicial set X are in bijection with morphisms (natural transformations) $\Delta^n \rightarrow X$.

Notation 3.1.9. Given $x \in X_n$, we will write x to denote both the n -simplex and the morphism $\Delta^n \rightarrow X$.

3.2 Simplicial Subsets

Most of the simplicial sets considered in this thesis are simplicial subsets of the standard n -simplex or, in Chapter 6, of a product of standard n -simplices. Here we define simplicial subsets, important examples of them, and some basic results about them.

Definition 3.2.1. \hookrightarrow A simplicial subset (or subcomplex) Y of a simplicial set X is a subpresheaf, that is, $Y \subseteq X$ if $Y_n \subseteq X_n$ for all n , and for any $f : [m] \rightarrow [n]$,

$$Xf|_{Y_n} = Yf.$$

For every simplicial subset $Y \subseteq X$, there is a canonical monomorphism $Y \hookrightarrow X$. In Lean, this is denoted by $Y.\iota : Y.\text{toSSet} \rightarrow X$ \hookrightarrow , where $Y.\text{toSSet} : \text{SSet}$ is how we work with $Y : X.\text{Subcomplex}$ as a simplicial set instead of a simplicial subset. This distinction is not often made on paper, but one must be consciously aware of it when formalizing.

Simplicial subsets of X form a complete lattice \hookrightarrow , with supremum and infimum being the union and intersection taken element-wise,

$$(X \cup Y)_n := X_n \cup Y_n$$

$$(X \cap Y)_n = X_n \cap Y_n.$$

Theorem 3.2.2. \hookrightarrow If $S, T \subseteq X$, then the following diagram is a pushout square:

$$\begin{array}{ccc} S \cap T & \hookrightarrow & T \\ \downarrow & & \downarrow \\ S & \hookrightarrow & S \cup T \end{array}$$

Proof. Colimits of simplicial sets are computed pointwise, so this reduces to an obvious pushout square in **Set**. \square

3.2.1 Examples of Subsets

The following are some important examples of simplicial subsets.

Definition 3.2.3. \hookrightarrow Let X, Y be simplicial sets and let $f : X \rightarrow Y$. We obtain a subset of Y , called the *range* of f , by $(\text{range } f)_n := \text{range}(f_n : X_n \rightarrow Y_n)$.

Example 3.2.4. \hookrightarrow If $x \in X_n$ is an n -simplex of X , then it generates a subset of X by taking the range of $x : \Delta^n \rightarrow X$.

Lemma 3.2.5. \hookrightarrow Let $x \in X_n$ and $A \subseteq X$. Then $\text{range } x \subseteq A \iff x \in A_n$.

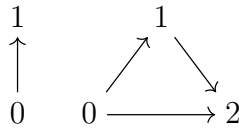
Proof. Let $\text{range } x \subseteq A$. To show $x \in A_n$, it suffices to show $x \in (\text{range } x)_n = \text{range}(x_n : \Delta^n \rightarrow X_n)$. Since $x = x_n(\text{id} : [n] \rightarrow [n])$, we are done.

Conversely, let $x \in A_n$. Let $k \in \mathbb{N}$, and let $y \in (\text{range } x)_k$. Let $f \in \Delta_k^n$ such that $x_k f = y$. Because A is a subset of X , $A f x = X f x = x_k f = y \in A_k$. \square

Definition 3.2.6. \hookrightarrow Let X, Y be simplicial sets, let $f : X \rightarrow Y$, and let $A \subseteq X$ be a subset of X . We obtain a subset of Y , $f(A)$, called the *image* of A under f , by $f(A)_n := f_n(A_n)$.

Definition 3.2.7. \hookrightarrow We can extend Definition 3.1.6 to subsets $S \subseteq [n]$ by letting Δ^S denote the simplicial set whose k -simplices are given by the order-preserving maps $[k] \rightarrow S$. Δ^S is naturally viewed as a subset of Δ^n .

Example 3.2.8. An important example of Definition 3.2.7 is $\Delta^{[n] \setminus i}$, the i -th *face* of Δ^n . Equivalently, the i -th face of Δ^n is the subset given by the range of the i -th face map $d^i : \Delta^{n-1} \rightarrow \Delta^n$ \hookrightarrow . One views the face $\Delta^{[n] \setminus i}$ as the face of Δ^n “opposite the vertex” i . For example, the face $\Delta^{[2] \setminus 2}$ of Δ^2 is visualized as follows:



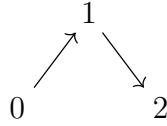
Definition 3.2.9. \hookrightarrow \hookrightarrow The *boundary* $\partial\Delta^n$ of the standard n -simplex is the simplicial subset of Δ^n whose k -simplices are given by the non-surjective functions $[k] \rightarrow [n]$. Equivalently, the boundary is the subset given by the union of all the faces $\Delta^{[n] \setminus i}$ of Δ^n . One views the boundary of the n -simplex as the n -simplex without its “interior”.

Remark 3.2.10. The inclusion $\partial\Delta^n \hookrightarrow \Delta^n$ is called the n -th *boundary inclusion*. These inclusions constitute a distinguished collection of morphisms in **SSet** which will be studied in Chapter 5.

Definition 3.2.11. $\boxtimes \boxtimes$ For $0 \leq i \leq n$, The i -th *horn* Λ_i^n of the standard n -simplex is the simplicial subset of Δ^n whose set of k -simplices is given by

$$\{\alpha : [k] \rightarrow [n] : \alpha([k] \cup \{i\}) \neq [n]\}.$$

Equivalently, the i -th horn is given by the union of all faces of Δ^n except the i -th one. One views the horn Λ_i^n as the n -simplex without its “interior” or the face “opposite the vertex” i . For example, the horn Λ_1^2 of Δ^2 is visualized as follows:



Definition 3.2.12. Let $s \in X_n$ be an n -simplex of X . We will call $s(\Delta^{[n]\setminus i})$ the i -th face of range $s \subseteq X$. Equivalently, the i -th face of range s is the range of $s \circ d^i : \Delta^{n-1} \rightarrow X$.

If $X = \Delta^m$, then s corresponds to a map $[n] \rightarrow [m]$ and we can represent s as $\langle s_0, \dots, s_n \rangle$ with $0 \leq s_0 \leq \dots \leq s_n \leq m$ (as in Notation 3.1.2). In this case, we can represent the i -th face of range s as $\langle s_0, \dots, \hat{s}_i, \dots, s_n \rangle$. These representations are very helpful for visualizing certain tedious arguments made in Chapter 6.

Theorem 3.2.13. \boxtimes Let $n > 0$ and let $A \subseteq \Delta^n$. Then

$$A \subseteq \Lambda_i^n \iff \Delta^{[n]\setminus i} \not\subseteq A.$$

Proof. If $A \subseteq \Lambda_i^n$ and $\Delta^{[n]\setminus i} \subseteq A$, then $\Delta^{[n]\setminus i} \subseteq \Lambda_i^n$ and this leads to a contradiction. The proof of the converse is omitted. \square

Corollary 3.2.14. \boxtimes_{JM} Let $n > 0$, let X be a simplicial set, let $A \subseteq X$, and let $f : \Delta^n \rightarrow X$ be a monomorphism such that $A \subseteq \text{range } f$. Then

$$A \subseteq f(\Lambda_i^n) \iff f(\Delta^{[n]\setminus i}) \not\subseteq A,$$

i.e. a subset of range f is contained in the i -th horn of range f if and only if the i -th face of range f is not contained in A .

Remark 3.2.15. An inclusion $\Lambda_i^n \hookrightarrow \Delta^n$ is called a *horn inclusion*. When $0 < i < n$, these are called *inner horn inclusions*. In Chapter 6, we study the collection of inner horn inclusions.

3.3 Quasi-Categories as ∞ -Categories

The following is a concise overview of the motivations behind viewing Kan complexes as ∞ -groupoids and quasi-categories as ∞ -categories. This section is included for conceptual background only; no results here are required for the formalization presented in this thesis. Most proofs are omitted, but can be found in [5, 15, 21].

First, we define Kan complexes and quasi-categories:

Definition 3.3.1. \square A simplicial set X is a *Kan complex* if every morphism $\Lambda_i^n \rightarrow X$ from a horn factors through Δ^n ,

$$\begin{array}{ccc} \Lambda_i^n & \longrightarrow & X \\ \downarrow & \nearrow \text{---} & \\ \Delta^n & & \end{array}$$

that is, there exists a “lift” $\Delta^n \rightarrow X$ making the above diagram commute.

If we restrict the above definition to only require lifts for inner horns, we define a quasi-category:

Definition 3.3.2. \square A simplicial set X is a *quasi-category* if every morphism $\Lambda_i^n \rightarrow X$ from an inner horn factors through Δ^n ,

$$\begin{array}{ccc} \Lambda_i^n & \longrightarrow & X \\ \downarrow & \nearrow \text{---} & \\ \Delta^n & & \end{array}$$

that is, there exists a “lift” $\Delta^n \rightarrow X$ making the above diagram commute.

Remark 3.3.3. The Kan complex and quasi-category condition are often called *horn-lifting conditions* or *horn-filling condition*. We will examine the consequences of this property in Chapter 4.

To illustrate the significance of these definitions, we first connect categories to simplicial sets via the nerve construction:

Definition 3.3.4. \curvearrowright Given a category \mathcal{C} , the *nerve of \mathcal{C}* , $N(\mathcal{C})$, is the simplicial set defined by

$$N(\mathcal{C})_n := \text{Fun}([n], \mathcal{C}),$$

where $[n] \in \mathbf{\Delta}$ is viewed as the category

$$0 \rightarrow 1 \rightarrow \cdots \rightarrow n.$$

The n -simplices of $N(\mathcal{C})$ are the n -composable morphisms in \mathcal{C} . This assignment defines a fully faithful functor $N : \mathbf{Cat} \rightarrow \mathbf{SSet}$.

We have the following two classical theorems:

Theorem 3.3.5. [15, Tag 0037] *The nerve of a groupoid is a Kan complex.*

Theorem 3.3.6. [15, Tag 003D] \curvearrowright *The nerve of a category is a quasi-category.*

Remark 3.3.7. Thus, groupoids correspond to Kan complexes and categories to quasi-categories. Moreover,

- (a) the nerve of a groupoid satisfies a stronger condition than a Kan complex: every horn-filling is *unique* (for $n \geq 2$).
- (b) the nerve of a category satisfies a stronger condition than a quasi-category: every inner horn-filling is *unique*.

In fact, any simplicial set satisfying (a) is the nerve of a groupoid and any simplicial set satisfying (b) is the nerve of a category. Weakening these conditions is how we get ∞ -groupoids and ∞ -categories. Recall that we want an ∞ -category to have $(n+1)$ -morphisms between n -morphisms for all $n \in \mathbb{N}$.

In a Kan complex or quasi-category, the n -simplices should be thought of as the n -morphisms. We will develop some language that lets us view a simplicial set in this way and see why weakened horn-filling conditions are significant.

Definition 3.3.8. The *objects* of a simplicial set S are the elements of S_0 (morphisms $\Delta^0 \rightarrow S$), and the *morphisms* of S are the elements of S_1 (morphisms $\Delta^1 \rightarrow S$).

Geometrically, the 0-simplex is a point and the 1-simplex is a line segment,



so we view a morphism $\Delta^0 \rightarrow S$ as a “point in S ”, and a morphism $\Delta^1 \rightarrow S$ as a “line segment in S ”, that is, we view them as objects and morphisms of S respectively.

Definition 3.3.9. If $f \in S_1$ is a morphism, then we will call the object

$$\text{dom}(f) := f \circ d^1 : \Delta^0 \rightarrow S$$

the *domain* of f and the object

$$\text{codom}(f) := f \circ d^0 : \Delta^0 \rightarrow S$$

the *codomain* of f , where $d^i : \Delta^0 \rightarrow \Delta^1$ are the face maps (Definition 3.1.3). We write $f : X \rightarrow Y$ to mean $X = \text{dom}(f)$ and $Y = \text{codom}(f)$.

If $f : X \rightarrow Y$ is a morphism in S , then X and Y correspond to the two ways to include a “point” into the endpoints of the “line segment” f , as specified by the face maps $d^0, d^1 : \Delta^0 \rightarrow \Delta^1$.

Definition 3.3.10. Given an object $X : \Delta^0 \rightarrow S$ in S , the *identity morphism* id_X denotes the morphism $\Delta^1 \rightarrow S$ in S given by precomposing $X : \Delta^0 \rightarrow S$ with the unique degeneracy map $s : \Delta^1 \rightarrow \Delta^0$ (Definition 3.1.3).

We think of the degeneracy map $s : \Delta^1 \rightarrow \Delta^0$ as collapsing the “line segment” to the “point”. Composing with an object $X : \Delta^0 \rightarrow S$ collapses the “line segment” to X , giving us a morphism $X \rightarrow X$.

Remark 3.3.11. If \mathcal{C} is a category, then we make the following observations:

- The objects of $N(\mathcal{C})$ are the objects of \mathcal{C} ,
- the morphisms of $N(\mathcal{C})$ are the morphisms of \mathcal{C} , and
- the identity morphisms of $N(\mathcal{C})$ are the identity morphisms of \mathcal{C} .

This justifies viewing S_0 and S_1 as objects and morphisms in a general simplicial set S . Next, we consider S_2 :

Definition 3.3.12. If $f : X \rightarrow Y$, $g : Y \rightarrow Z$, and $h : X \rightarrow Z$ are morphisms of S , then we say h is a *composition* of f and g if there exists a 2-simplex

$$\sigma : \Delta^2 \rightarrow S$$

such that

$$\sigma \circ d^0 = g, \quad \sigma \circ d^1 = h, \quad \text{and} \quad \sigma \circ d^2 = f,$$

where $d^i : \Delta^2 \rightarrow \Delta^1$ are the face maps (Definition 3.1.3).

Notation 3.3.13. We write $h = g \circ f$ if there exists a 2-simplex σ which presents h as a composition of f and g . Such an h is not unique, but is well-defined up to *homotopy* (see Remark 3.3.16).

The 2-simplex is a triangle, so we view a morphism $\sigma : \Delta^2 \rightarrow S$ as follows:

$$\begin{array}{ccc} & Y & \\ f \nearrow & & \searrow g \\ X & \xrightarrow{h} & Z \end{array}$$

Here f , g , and h are the “faces” of σ viewed as morphisms in S (corresponding to the three ways to include a “line segment” into the faces of the “triangle” σ), and σ presents h as a composition of f and g .

Definition 3.3.14. A morphism $f : X \rightarrow Y$ of S is an isomorphism if there exists morphisms $g : Y \rightarrow X$ and $g' : Y \rightarrow X$ in S such that $f \circ g = \text{id}_Y$ and

$g' \circ f = \text{id}_X$:

$$\begin{array}{ccc} & Y & \\ f \nearrow & & \searrow g' \\ X & \xrightarrow{\text{id}_X} & X \end{array} \quad \begin{array}{ccc} & X & \\ g \nearrow & & \searrow f \\ Y & \xrightarrow{\text{id}_Y} & Y \end{array}$$

Definition 3.3.15. If $f, g : X \rightarrow Y$ are morphisms of S , then f is *homotopic* to g if $g = \text{id}_Y \circ f$ or, equivalently, $g = f \circ \text{id}_X$:

$$\begin{array}{ccc} & Y & \\ f \nearrow & & \searrow \text{id}_Y \\ X & \xrightarrow{g} & Y \end{array} \quad \begin{array}{ccc} & X & \\ \text{id}_X \nearrow & & \searrow f \\ X & \xrightarrow{g} & Y \end{array}$$

Remark 3.3.16. Composition and isomorphism are not unique in S , but they are unique up to homotopy: if h and h' are compositions of f and g , then h is homotopic to h' . Similarly, the g and g' appearing in Definition 3.3.14 will be homotopic.

This leads to the definition of the *homotopy category* of S . We omit discussion here, but a comprehensive treatment of homotopy and the homotopy category of a simplicial set can be found in [5, 15].

Remark 3.3.17. If \mathcal{C} is a category, then we make the following observations:

- Morphisms $f : X \rightarrow Y$, $g : Y \rightarrow Z$ in \mathcal{C} have a unique composition in $N(\mathcal{C})$ given by their composition in \mathcal{C} ,
- a morphism $f : X \rightarrow Y$ is an isomorphism in $N(\mathcal{C})$ if and only if it is an isomorphism in \mathcal{C} , and
- morphisms $f, g : X \rightarrow Y$ in \mathcal{C} are homotopic in $N(\mathcal{C})$ if and only if $f = g$,

This justifies our definitions of composition and isomorphism in a general simplicial set and indicates that homotopy correctly generalizes the strict uniqueness of each.

Thus, 2-simplices of a simplicial set S express composition and invertibility of morphisms of S up to homotopy. Similarly, n -simplices can be composed and inverted up to homotopies given by higher simplices.

Next, we consider the significance of horn-filling. The horns of the 2-simplex exclude its faces, so we view morphisms $\Lambda_0^2 \rightarrow S$, $\Lambda_1^2 \rightarrow S$, and $\Lambda_2^2 \rightarrow S$ as follows:

$$\begin{array}{ccc}
 \begin{array}{ccc} & Y & \\ f \nearrow & & \\ X & \xrightarrow{h} & Z \end{array} &
 \begin{array}{ccc} & Y & \\ f \nearrow & & \searrow g \\ X & & Z \end{array} &
 \begin{array}{ccc} & Y & \\ & & \searrow g \\ X & \xrightarrow{h} & Z \end{array}
 \end{array}$$

Recall the horn-filling condition for Kan complexes (Definition 3.3.1), which says that every morphism $\Lambda_i^n \rightarrow S$ extends to a morphism $\Delta^n \rightarrow S$. In the case $n = 2$, this means that the preceding horn diagrams extend to compositions:

$$\begin{array}{ccc}
 \begin{array}{ccc} & Y & \\ f \nearrow & \dashrightarrow & \\ X & \xrightarrow{h} & Z \end{array} &
 \begin{array}{ccc} & Y & \\ f \nearrow & & \searrow g \\ X & \dashrightarrow & Z \end{array} &
 \begin{array}{ccc} & Y & \\ \dashrightarrow & & \searrow g \\ X & \xrightarrow{h} & Z \end{array}
 \end{array}$$

This is a significant condition—filling certain horns corresponds to the existence of inverses:

Theorem 3.3.18. *Every morphism in a Kan complex is an isomorphism.*

Proof. Let S be a Kan complex and let $f : X \rightarrow Y$ be a morphism. Consider the morphisms $\Lambda_0^2 \rightarrow S$ and $\Lambda_2^2 \rightarrow S$ given by the following diagrams:

$$\begin{array}{ccc}
 \begin{array}{ccc} & Y & \\ f \nearrow & & \\ X & \xrightarrow{\text{id}_X} & X \end{array} &
 \begin{array}{ccc} X & & \\ & \searrow f & \\ Y & \xrightarrow{\text{id}_Y} & Y \end{array}
 \end{array}$$

The horn-filling condition implies that there exist morphisms $g, g' : Y \rightarrow X$ as follows:

$$\begin{array}{ccc}
 \begin{array}{ccc} & Y & \\ f \nearrow & \dashrightarrow g' & \\ X & \xrightarrow{\text{id}_X} & X \end{array} &
 \begin{array}{ccc} & X & \\ g \nearrow & & \searrow f \\ Y & \xrightarrow{\text{id}_Y} & Y \end{array}
 \end{array}$$

from which it follows that f is an isomorphism. □

Remark 3.3.19. In light of Theorem 3.3.18 and the language now established, we reconsider Remark 3.3.7. The nerve of a category \mathcal{C} satisfies (b) because

composition in \mathcal{C} is unique. Similarly, the nerve of a groupoid \mathcal{C} satisfies (a).

By relaxing conditions (a) and (b), inverses and compositions become unique only up to homotopy, and this is how we generalize categories and groupoids to ∞ -categories and ∞ -groupoids. Intuitively, and in analogy to the fundamental groupoid of a topological space, we may think of the n -simplices of a Kan complex or quasi-category as its n -morphisms: 0-simplices are objects, 1-simplices are 1-morphisms, and 2-simplices are 2-morphisms, which may be thought of as homotopies between 1-morphisms. Similarly, higher $(n + 1)$ -simplices express homotopies between n -simplices. This is an imprecise but useful heuristic, though a further investigation is beyond the scope of this thesis. For example, a more precise view of 2-simplices as 2-morphisms can be found at [9].

In a Kan complex, all n -morphisms are invertible, which gives rise to an ∞ -groupoid structure. In a quasi-category, all $n \geq 2$ morphisms are invertible, which gives rise to an ∞ -category structure (in particular, this is an $(\infty, 1)$ -categorical structure [4]).

We now move on to the functor quasi-category.

3.4 The Functor Quasi-Category

Given two quasi-categories S and X , we want a simplicial set Y to serve as a simplicial set of functors (i.e. natural transformations) between S and X . In the sense of Definition 3.3.8, this means Y_0 should identify to functors $S \rightarrow X$ and Y_1 should identify to “natural transformations” between such functors. This is possible thanks to the cartesian closed monoidal structure on **SSet**.

If a category \mathcal{C} has finite products, then it has a *cartesian* monoidal category structure whose tensor product is given by the categorical (cartesian) product, and whose terminal object (the empty product) is the unit. This applies to categories of **Set**-valued (or **Type**-valued) functors and, in particular, to simplicial sets, which inherit finite products from **Set** (or in Lean, **Type**).

Definition 3.4.1.  A *closed* monoidal category is a symmetric monoidal

category (\mathcal{C}, \otimes) equipped with a functor

$$[-.-] : \mathcal{C}^{\text{op}} \times \mathcal{C} \rightarrow \mathcal{C},$$

called the *internal hom functor* \boxtimes , and for every object $X \in \mathcal{C}$ an adjunction

$$(X \otimes -) \dashv [X, -].$$

Remark 3.4.2. The internal hom generalizes the classical hom functor in **Set**,

$$\text{Hom}(-, -) : \mathbf{Set}^{\text{op}} \times \mathbf{Set} \rightarrow \mathbf{Set},$$

which satisfies the adjunction $(X \times -) \dashv \text{Hom}(X, -)$:

$$\text{Hom}(X \times Y, Z) \simeq \text{Hom}(Y, \text{Hom}(X, Z)).$$

The object $[X, Y]$ in a closed monoidal category plays the role of an “internal object of morphisms,” analogous to $\text{Hom}(X, Y)$ in **Set**.

If S and X are simplicial sets, then the collection of morphism between them, $\mathbf{SSet}(S, X)$, does not have the structure of a simplicial set, but the internal hom between S and X does. We start by defining this more generally:

Theorem 3.4.3. $\boxtimes_{\text{JM}} \boxtimes_{\text{JM}}$ *The internal hom $[F, G]$ of functors $\mathcal{F}, \mathcal{G} : \mathcal{C} \rightarrow \mathbf{Set}$ is given by*

$$[\mathcal{F}, \mathcal{G}]_c := \mathbf{Set}^c(\mathcal{F} \times \mathbf{Set}^c(c, -), \mathcal{G}),$$

where $\mathbf{Set}^c(c, -) : \mathcal{C} \rightarrow \mathbf{Set}$ denotes the covariant Yoneda embedding of c .

Proof. It is a straightforward verification to show that

$$\mathbf{Set}^c(\mathcal{A} \times \mathcal{F}, \mathcal{G}) \cong \mathbf{Set}^c(\mathcal{F}, [\mathcal{A}, \mathcal{G}]) :$$

Let $f : \mathcal{A} \times \mathcal{F} \rightarrow \mathcal{G}$. Define $g : \mathcal{F} \rightarrow [\mathcal{A}, \mathcal{G}]$ as follows: Let $c \in \mathcal{C}$ and $x \in \mathcal{F}c$. Define a morphism

$$\hat{x} : \mathcal{A} \times \mathbf{Set}^c(c, -) \rightarrow \mathcal{G}$$

by $\hat{x}_{c'} : (a, \varphi) \mapsto f_{c'}(a, \mathcal{F}\varphi(x))$. Define $g_c : \mathcal{F}c \rightarrow [\mathcal{A}, \mathcal{G}]_c$ by $x \mapsto \hat{x}$.

Conversely, let $f : \mathcal{F} \rightarrow [\mathcal{A}, \mathcal{G}]$. Define $g : \mathcal{A} \times \mathcal{F} \rightarrow \mathcal{G}$ as follows: Let $c \in \mathcal{C}$ and $x \in \mathcal{F}c$. Then f_c determines a morphism

$$f_c(x) : \mathcal{A} \times \mathbf{Set}^c(c, -) \rightarrow \mathcal{G},$$

so define $g_c : \mathcal{A}c \times \mathcal{F}c \rightarrow \mathcal{G}c$ by $(a, x) \mapsto f_c(x)(a, \text{id}_c)$.

It is straightforward to show that these assignments are inverses. \square

Definition 3.4.4. If S and X are quasi-categories, then the *functor quasi-category* between S and X is the internal hom of S and X :

$$[S, X]_n = \mathbf{SSet}(S \times \Delta^n, X).$$

Our goal is to show that $[S, X]$ is a quasi-category when S and X are. In fact, it is sufficient that X is a quasi-category. We prove this in Theorem 7.1.1.

Notation 3.4.5. $[S, X]$ is often denoted by $\text{Fun}(S, X)$ or X^S to indicate its similarity to a functor category. For simplicity, we will only write $[S, X]$ throughout this thesis.

Remark 3.4.6. Note that

$$[S, X]_0 = \mathbf{SSet}(S \times \Delta^0, X)$$

is isomorphic to

$$\mathbf{SSet}(S, X),$$

because Δ^0 is the monoidal unit in \mathbf{SSet} . Therefore, the objects of $[S, X]$, in the sense of Definition 3.3.8, are exactly the functors $S \rightarrow X$.

Similarly,

$$[S, X]_1 = \mathbf{SSet}(S \times \Delta^1, X)$$

are exactly the *natural transformations* between such functors (see [5, 15, 21] for how natural transformations are defined; they are not relevant for this thesis).

Hence, the internal hom correctly models the quasi-category whose objects are functors between quasi-categories and whose morphisms are natural

transformations between such functors.

Chapter 4

Lifting Properties and Saturated Collections

In this chapter, we introduce the language of lifting properties as a way to reframe the quasi-category condition. To that end, we also define the notion of a saturated collection of morphisms, and some consequences of the adjunction presented in Theorem 3.4.3.

4.1 Lifting Problems

Definition 4.1.1. \boxtimes Let $f : A \rightarrow B$ and $g : X \rightarrow Y$ be morphisms in a category \mathcal{C} . We say f has the *left lifting property* (llp) with respect to g , and g has the *right lifting property* (rlp) with respect to f , and we write $f \boxtimes g$, if every diagram of the following form admits a *lift* $h : B \rightarrow X$:

$$\begin{array}{ccc} A & \xrightarrow{u} & X \\ f \downarrow & \nearrow h & \downarrow g \\ B & \xrightarrow{v} & Y \end{array}$$

i.e for any $u : A \rightarrow X$ and $v : B \rightarrow Y$ such that $g \circ u = v \circ f$, there exists a “lift” $h : B \rightarrow X$ making the above diagram commute. We often call such diagrams *lifting problems* and h a *solution* to the lifting problem.

If T is a collection of morphisms in a category \mathcal{C} , then we say a morphism

f has llp with respect to T , and we write $f \sqsubset T$, if f has llp with respect to every morphism in T . The collection of all morphisms f such that $f \sqsubset T$ is denoted by $\text{llp}(T)$. The dual notions for rlp are denoted similarly.

Remark 4.1.2. $\hookrightarrow_{\text{JM}}$ A simplicial set S is a quasi-category if and only if every inner horn inclusion belongs to $\text{llp}(S \rightarrow \Delta^0)$, because Δ^0 is terminal (Theorem 3.1.7):

$$\begin{array}{ccc} \Lambda_i^n & \longrightarrow & S \\ \downarrow & \nearrow & \downarrow \\ \Delta^n & \longrightarrow & \Delta^0 \end{array}$$

We define llp and rlp using the `MorphismProperty` API discussed in Chapter 1 $\hookrightarrow_{\text{JM}}$:

```
def llp (T : MorphismProperty C) : MorphismProperty C := fun _
  _ f ↦
  ∀ {X Y : C} (g : X → Y) (_ : T g), HasLiftingProperty f g
```

```
def rlp (T : MorphismProperty C) : MorphismProperty C := fun _
  _ f ↦
  ∀ {X Y : C} (g : X → Y) (_ : T g), HasLiftingProperty g f
```

Given a collection of morphisms T in a suitable category, $\text{llp}(T)$ and $\text{rlp}(T)$ have many nice properties. In particular, $\text{llp}(T)$ satisfies certain properties which make it a *saturated* collection.

4.2 Saturated Collections of Morphisms

Let T be a collection of morphisms in a category C . When making arguments about $\text{llp}(T)$ —which we will need to do, because of Remark 4.1.2—it is useful to consider how “big” this collection is. $\text{llp}(T)$ is stable under a number of important properties, which provide convenient ways of showing when a morphism belongs to $\text{llp}(T)$.

Definition 4.2.1. $\lrcorner T$ is stable under *cobase change* (or *pushouts*) if for any pushout square

$$\begin{array}{ccc} X & \longrightarrow & X' \\ f \downarrow & & \downarrow f' \\ Y & \longrightarrow & Y' \end{array}$$

$f \in T$ implies $f' \in T$. That is, if f' is a pushout of $f \in T$, then $f' \in T$.

Lemma 4.2.2. \lrcorner_{JM} Monomorphisms in \mathbf{SSet} are stable under cobase change.

Proof. This follows from a more general fact that \mathbf{SSet} is an *adhesive* category. \square

Lemma 4.2.3. $\lrcorner_{\text{JM}} \text{lp}(T)$ is stable under cobase change.

Proof. Let f' be a cobase change of f and let $f \sqsupseteq g$ for some $g : A \rightarrow B$. We claim that $f' \sqsupseteq g$. Consider the following lifting problem:

$$\begin{array}{ccc} X' & \xrightarrow{u} & A \\ f' \downarrow & & \downarrow g \\ Y' & \xrightarrow{v} & B \end{array}$$

Graft the lifting problem onto the pushout square of f' and f to obtain a lift $h : Y \rightarrow A$:

$$\begin{array}{ccccc} X & \xrightarrow{s} & X' & \xrightarrow{u} & A \\ f \downarrow & & \downarrow & \nearrow h & \downarrow g \\ Y & \xrightarrow{t} & Y' & \xrightarrow{v} & B \end{array}$$

This induces a map $h' : Y' \rightarrow A$ which will be a solution to the lifting problem:

$$\begin{array}{ccc} X & \xrightarrow{s} & X' \\ f \downarrow & & \downarrow f' \\ Y & \xrightarrow{t} & Y' \end{array} \begin{array}{c} \xrightarrow{u} \\ \dashrightarrow h' \\ \xrightarrow{h} \end{array} A$$

To show that h' is a solution, we check that $h' \circ f' = u$ and $g \circ h' = v$. The first equality is true by construction of h' and the second equality is true by

observing that $(g \circ h') \circ f' = g \circ u = v \circ f'$ and $(g \circ h') \circ t = g \circ h = v \circ t$. Therefore, h' is a lift and $f' \sqsupseteq g$. \square

Definition 4.2.4. $\hookrightarrow_{\text{JM}}$ T is stable under *retracts* if for any diagram of the form

$$\begin{array}{ccccc} & & \text{id} & & \\ & \curvearrowright & & \curvearrowleft & \\ X' & \longrightarrow & X & \longrightarrow & X' \\ f' \downarrow & & \downarrow f & & \downarrow f' \\ Y' & \longrightarrow & Y & \longrightarrow & Y' \\ & \curvearrowleft & & \curvearrowright & \\ & & \text{id} & & \end{array}$$

$f \in T$ implies $f' \in T$. That is, if f' is a retract (in the arrow category of \mathcal{C}) of $f \in T$, then $f' \in T$.

Lemma 4.2.5. $\hookrightarrow_{\text{JM}}$ *Monomorphisms are stable under retracts.*

Proof. Let f be a monomorphism and let f' be a retract of f :

$$\begin{array}{ccccc} & & \text{id} & & \\ & \curvearrowright & & \curvearrowleft & \\ X' & \xrightarrow{i} & X & \xrightarrow{r} & X' \\ f' \downarrow & & \downarrow f & & \downarrow f' \\ Y' & \xrightarrow{i'} & Y & \xrightarrow{r'} & Y' \\ & \curvearrowleft & & \curvearrowright & \\ & & \text{id} & & \end{array}$$

We claim f' is a monomorphism. The morphisms i and i' are split monomorphisms since they admit retractions r and r' . In particular, i and i' are monomorphisms, so $f \circ i$ is also a monomorphism. Let $g_1, g_2 : Z \rightarrow X'$. Then

$$\begin{aligned} g_1 = g_2 &\iff (f \circ i) \circ g_1 = (f \circ i) \circ g_2 \\ &\iff (i' \circ f') \circ g_1 = (i' \circ f') \circ g_2 \\ &\iff f' \circ g_1 = f' \circ g_2, \end{aligned}$$

so f' is a monomorphism. \square

Lemma 4.2.6. $\hookrightarrow_{\text{JM}}$ $llp(T)$ is stable under retracts.

Proof. Let $f \sqsupseteq g$ for some $g : A \rightarrow B$. We claim that $f' \sqsupseteq g$. Consider the

following lifting problem:

$$\begin{array}{ccc} X' & \xrightarrow{u} & A \\ f' \downarrow & & \downarrow g \\ Y' & \xrightarrow{v} & B \end{array}$$

Graft the lifting problem onto the retract diagram of f' and f to obtain a lift $h : Y \rightarrow A$:

$$\begin{array}{ccccccc} X' & \xrightarrow{i} & X & \xrightarrow{i'} & X' & \xrightarrow{u} & A \\ f' \downarrow & & f \downarrow & \nearrow h' & \downarrow & \nearrow h & \downarrow g \\ Y' & \xrightarrow{r} & Y & \xrightarrow{r'} & Y' & \xrightarrow{v} & B \end{array}$$

Let $h' = h \circ r$. Observe that $(h \circ r) \circ f' = h \circ (f \circ i) = (u \circ i') \circ i = u$ and $g \circ (h \circ r) = (v \circ r') \circ r = v$, so $h' : Y' \rightarrow A$ is a solution to the lifting problem. Therefore, $f' \sqsupseteq g$. \square

Definition 4.2.7. \hookrightarrow A morphism $f : X \rightarrow Y$ in \mathcal{C} is a transfinite composition (of shape J) of morphisms in T if there exists a well-ordered set J and a functor $\mathcal{F} : J \rightarrow \mathcal{C}$ such that

- (a) for every limit element $j \in J$, $\mathcal{F}(j)$ is a colimit over \mathcal{F} restricted to $\{i \in J : i < j\}$,
- (b) there exists an isomorphism $X \cong \mathcal{F}(\perp)$, where \perp is the bottom element of J ,
- (c) there exists a colimiting cocone over \mathcal{F} whose point is Y ,
- (d) the composition $X \rightarrow \mathcal{F}(\perp) \rightarrow Y$ is equal to f ,
- (e) for every $j \in J$ which is not maximal, $\mathcal{F}(j \leq j+1) : \mathcal{F}(j) \rightarrow \mathcal{F}(j+1)$ belongs to T .

T is stable under transfinite composition if any transfinite composition (of any shape) of morphisms in T belongs to T .

Remark 4.2.8. Transfinite composition is usually stated in terms of ordinals (i.e. well-ordered sets), as in [5, 15, 21]. As might be expected, we instead use well-ordered types in Lean.

Lemma 4.2.9. \hookrightarrow *Monomorphisms of simplicial sets are stable under transfinite composition.*

Proof. The proof is omitted here, but this reduces to checking that monomorphisms of types are stable under transfinite composition, which is shown here: \hookrightarrow . \square

Lemma 4.2.10. \hookrightarrow *$llp(T)$ is stable under transfinite composition.*

Proof. The proof is omitted here, but can be found in [15, Tag 006R]. \square

Definition 4.2.11. \hookrightarrow *T is stable under coproducts (of shape J) if for any family of morphisms $\{f_j : X_j \rightarrow Y_j\}_{j \in J}$ in T , the morphism $\coprod_j f_j : \coprod_j X_j \rightarrow \coprod_j Y_j$ is in T .*

T is stable under coproducts if it is stable under coproducts of any shape.

Lemma 4.2.12. \hookrightarrow *Monomorphisms of simplicial sets are stable under coproducts.*

Proof. The proof is omitted here, but this reduces to checking that monomorphisms of types are stable under coproducts, which is shown here: \hookrightarrow . \square

Lemma 4.2.13. \hookrightarrow_{JM} *$llp(T)$ is stable under coproducts.*

Proof. Let $\{f_j : X_j \rightarrow Y_j\} \sqsupseteq g$ for some $g : A \rightarrow B$. We claim that $\coprod_j f_j \sqsupseteq g$. Consider the following lifting problem:

$$\begin{array}{ccc} \coprod_j X_j & \longrightarrow & A \\ \coprod_j f_j \downarrow & & \downarrow g \\ \coprod_j Y_j & \longrightarrow & B \end{array}$$

It is straightforward to verify that a solution $\coprod_j Y_j \rightarrow A$ is given by the coproduct of the lifts given by each $f_j \sqsupseteq g$. \square

A collection of morphisms that is stable under the above conditions is called *saturated*:

Definition 4.2.14. $\hookrightarrow_{\text{JM}}$ A collection of morphisms T which is closed under cobase change, retracts, transfinite composition, and coproducts is called *saturated* (or *weakly saturated*).

Corollary 4.2.15. \hookrightarrow A saturated collection of morphisms is stable under composition with isomorphisms.

Proof. This follows from stability under cobase change. \square

Remark 4.2.16. Corollary 4.2.15 is very useful: if two morphisms are isomorphic (in the arrow category), then one belongs to a saturated collection if and only if the other does. We will use this result throughout the rest of this thesis.

We implement saturated collections with the type class system. Given $T : \text{MorphismProperty } C$, the following class asserts that T is saturated:

```
Saturated {C : Type u} [Category.{v} C] (T : MorphismProperty
  C) : Prop where
  IsStableUnderCobaseChange : T.IsStableUnderCobaseChange
  IsStableUnderRetracts : T.IsStableUnderRetracts
  IsStableUnderCoproducts : IsStableUnderCoproducts.{w} T
  IsStableUnderTransfiniteComposition :
    IsStableUnderTransfiniteComposition.{w} T
```

The lemmas in this section yield the following important theorems:

Theorem 4.2.17. $\hookrightarrow_{\text{JM}}$ For any collection of morphisms T in a category \mathcal{C} , $\text{llp}(T)$ is saturated.

Theorem 4.2.18. $\hookrightarrow_{\text{JM}}$ The collection of monomorphisms of simplicial sets is saturated.

Definition 4.2.19. $\hookrightarrow_{\text{JM}}$ Given a collection of morphisms S , we let \overline{S} denote the saturated collection generated by S (that is, the smallest one containing S). We call \overline{S} the saturation of S . It follows that a saturated collection contains S if and only if it contains \overline{S} $\hookrightarrow_{\text{JM}}$.

We define the saturation of $T : \text{MorphismProperty } C$ using an inductive type:

```

inductive SaturatedClass (T : MorphismProperty C) : {X Y : C} →
  (X → Y) → Prop
| of {X Y : C} (f : X → Y) (h : T f) : SaturatedClass T f
| pushout ... : ...
| retract ... : ...
| coproduct ... : ...
| transfinite ... : ...

```

```

def saturation : MorphismProperty C := fun _ _ f ↦
  SaturatedClass.{w} T f

```

Here ellipses signify the omission of code fragments for clarity. The constructors of `SaturatedClass` ensure that any term of `SaturatedClass T f` reduces inductively to `f` already being in `T` (this is the constructor `of`) or `f` being a pushout (cobase change), retract, coproduct, or transfinite composition of morphisms in `T`. Using the same design established in Section 2.4, `saturation` captures the intended meaning of the saturation of a collection morphisms.

Remark 4.2.20. Theorem 4.2.17 implies that $\text{llp}(S \rightarrow \Delta^0)$ is saturated for any simplicial set S . In particular, if S is a quasi-category, then $\text{llp}(S \rightarrow \Delta^0)$ contains all inner horn inclusions. Definition 4.2.19 implies that $\text{llp}(S \rightarrow \Delta^0)$ contains the saturation of all inner horn inclusions. We will investigate this saturated collection in Chapter 6.

4.3 Adjunction of Lifting Properties

We aim to prove a lifting result involving an internal hom functor $[-, -]$. As a consequence of the cartesian closedness of `SSet`, certain lifting problems involving $[-, -]$ are equivalent to lifting problems involving $- \times -$. In this section, we investigate that connection. Most statements in this section are adapted from [21].

Definition 4.3.1. \mathcal{C}_{JM} Let $f : A \rightarrow B, g : K \rightarrow L$ be morphisms of simplicial sets and consider the following diagram:

$$\begin{array}{ccc}
 A \times K & \xrightarrow{A \triangleleft g} & A \times L \\
 f \triangleright K \downarrow & & \downarrow \\
 B \times K & \longrightarrow & (A \times L) \coprod_{A \times K} (B \times K) \\
 & \searrow & \downarrow f \triangleright L \\
 & & B \times L \\
 & \xrightarrow{B \triangleleft g} & \\
 & \nearrow f \square g &
 \end{array}$$

The induced map $f \square g$ is called the *pushout-product* of f and g . Here \triangleright and \triangleleft denote the “whiskering” operation: $f \triangleright K = f \times \text{id}_K$.

Example 4.3.2. If $K = \emptyset$ is the initial object and $L = *$ is the terminal object, then $(A \times *) \coprod_{A \times \emptyset} (B \times \emptyset) \cong A$, $B \times * \cong B$, and $f \square g \cong f$.

Definition 4.3.3. Let $g : K \rightarrow L, h : X \rightarrow Y$ be morphisms of simplicial sets and consider the following diagram:

$$\begin{array}{ccc}
 [L, X] & \xrightarrow{[g, X]} & [K, X] \\
 \downarrow h \square g & & \downarrow [K, h] \\
 [L, Y] & \xrightarrow{[g, Y]} & [K, Y] \\
 \downarrow [L, h] & & \downarrow [K, h] \\
 [L, Y] & \xrightarrow{[g, Y]} & [K, Y]
 \end{array}$$

The induced map $h \square g$ is called the *pullback-power* of h and g .

Remark 4.3.4. \mathcal{C}_{JM} The above constructions determine bifunctors $\square : \mathbf{SSet}^2 \times \mathbf{SSet}^2 \rightarrow \mathbf{SSet}^2$ and $\square : (\mathbf{SSet}^2)^{\text{op}} \times \mathbf{SSet}^2 \rightarrow \mathbf{SSet}^2$ where \mathbf{SSet}^2 denotes the *arrow category* of \mathbf{SSet} .

These constructions are generalized with the notion of a two-variable adjunction [24], or the weaker notion of a *parameterized adjunction* as formalized in Mathlib \mathcal{C}_{JM} \mathcal{C}_{JM} \mathcal{C}_{JM} . For the scope of this thesis, it is enough to consider just the pushout-product and pullback-power as defined in the context of simplicial sets.

The adjunction between products and internal homs leads to the following result:

Theorem 4.3.5. \square *Let $f : A \rightarrow B$, $g : K \rightarrow L$, $h : X \rightarrow Y$ be morphisms of simplicial sets. The following lifting problems are equivalent:*

$$\begin{array}{ccc}
 (A \times L) \coprod_{A \times K} (B \times K) & \longrightarrow & X \\
 f \square g \downarrow & \nearrow & \downarrow h \\
 B \times L & \longrightarrow & Y
 \end{array}
 \iff
 \begin{array}{ccc}
 A & \longrightarrow & [L, X] \\
 f \downarrow & \nearrow & \downarrow [h \square g] \\
 B & \longrightarrow & [K, X] \times_{[K, Y]} [L, Y]
 \end{array}$$

Proof. See [21, Prop. 21.5]. \square

If $Y = *$ is the terminal objects, then we have the following corollary:

Corollary 4.3.6. *Let $f : A \rightarrow B$, $g : K \rightarrow L$ be morphisms of simplicial sets, and let X be a simplicial set. The following lifting problems are equivalent:*

$$\begin{array}{ccc}
 (A \times L) \coprod_{A \times K} (B \times K) & \longrightarrow & X \\
 f \square g \downarrow & \nearrow & \downarrow \\
 B \times L & \longrightarrow & *
 \end{array}
 \iff
 \begin{array}{ccc}
 A & \longrightarrow & [L, X] \\
 f \downarrow & \nearrow & \downarrow [g, X] \\
 B & \longrightarrow & [K, X]
 \end{array}$$

Proof. This follows from the isomorphism $[K, X] \times_{[K, *]} [L, *] \cong [K, X]$. \square

If $K = \emptyset$ is the initial object, then we have the following corollary:

Corollary 4.3.7. *Let $f : A \rightarrow B$, $h : X \rightarrow Y$ be morphisms of simplicial sets, and let L be a simplicial set. The following lifting problems are equivalent:*

$$\begin{array}{ccc}
 A \times L & \longrightarrow & X \\
 f \triangleright L \downarrow & \nearrow & \downarrow h \\
 B \times L & \longrightarrow & Y
 \end{array}
 \iff
 \begin{array}{ccc}
 A & \longrightarrow & [L, X] \\
 f \downarrow & \nearrow & \downarrow [L, h] \\
 B & \longrightarrow & [L, Y]
 \end{array}$$

Proof. This follows from the isomorphisms $(A \times L) \coprod_{A \times \emptyset} (B \times \emptyset) \cong A \times L$ and $[\emptyset, X] \times_{[\emptyset, Y]} [L, Y] \cong [L, Y]$. \square

4.3.1 Interactions with Saturated Collections

In Chapters 6 and 7, we make extensive use of the pushout-product construction. To that end, we record some facts about how pushout-products behave with respect to cobase change, retracts, transfinite composition, and the taking of coproducts.

Let $f : A \rightarrow B$, $g : K \rightarrow L$, and $h : X \rightarrow Y$ be morphisms of simplicial sets. The results in this section are true and formalized for more general categories, but we will restrict our attention to \mathbf{SSet} .

Lemma 4.3.8. $\hookrightarrow_{\text{JM}}$ *If f is a retract of g , then $f \square h$ is a retract of $g \square h$.*

Proof. $-\square h$ is a functor (Remark 4.3.4) between the arrow category of \mathbf{SSet} and functors preserve retracts. \square

Lemma 4.3.9. $\hookrightarrow_{\text{JM}}$ *If f is a cobase change of g , then $f \square h$ is a cobase change of $g \square h$.*

Proof. This proof is adapted from [24, Prop. C.2.9]. Let the following diagram be a pushout:

$$\begin{array}{ccc} K & \longrightarrow & A \\ g \downarrow & & \downarrow f \\ L & \longrightarrow & B \end{array}$$

Let $\text{pt } fh = (A \times Y) \coprod_{A \times X} (B \times X)$ and $\text{pt } gh = (K \times Y) \coprod_{K \times X} (L \times X)$. We wish to show the following diagram (induced by the functoriality of $-\square h$) is a pushout square:

$$\begin{array}{ccc} \text{pt } gh & \longrightarrow & \text{pt } fh \\ g \square h \downarrow & & \downarrow f \square h \\ L \times Y & \longrightarrow & B \times Y \end{array}$$

$-\times -$ is cocontinuous in both variables, so the following two diagrams are pushout squares:

$$\begin{array}{ccc} K \times X & \longrightarrow & A \times X \\ g \triangleright X \downarrow & & \downarrow f \triangleright X \\ L \times X & \longrightarrow & B \times X \end{array} \quad \begin{array}{ccc} K \times Y & \longrightarrow & A \times Y \\ g \triangleright Y \downarrow & & \downarrow f \triangleright Y \\ L \times Y & \longrightarrow & B \times Y \end{array}$$

By pasting,

$$\begin{array}{ccc}
 K \times Y & \longrightarrow & A \times Y \\
 \downarrow & & \downarrow \\
 \text{pt } gh & \longrightarrow & \text{pt } fh \\
 \downarrow & & \downarrow \\
 L \times Y & \longrightarrow & B \times Y
 \end{array}$$

it suffices to prove the top square is a pushout, because the outer square is a pushout. By pasting again,

$$\begin{array}{ccccc}
 K \times X & \longrightarrow & K \times Y & \longrightarrow & A \times Y \\
 \downarrow & & \downarrow & & \downarrow \\
 L \times X & \longrightarrow & \text{pt } gh & \longrightarrow & \text{pt } fh
 \end{array}$$

it suffices to prove the outer square is a pushout. Pasting once more,

$$\begin{array}{ccccc}
 K \times X & \longrightarrow & A \times X & \longrightarrow & A \times Y \\
 \downarrow & & \downarrow & & \downarrow \\
 L \times X & \longrightarrow & B \times X & \longrightarrow & \text{pt } fh
 \end{array}$$

we are done. □

Lemma 4.3.10. $\hookrightarrow_{\text{JM}}$ $\hookrightarrow_{\text{JM}}$ *If f is a transfinite composition of shape J , then $f \square h$ is a transfinite composition of shape J .*

Proof. The proof is omitted here for the sake of brevity. The statement is formalized as follows:

```

instance [T.IsStableUnderCobaseChange]
  [IsStableUnderTransfiniteComposition.{w} T] :
  IsStableUnderTransfiniteComposition.{w} (T.pushoutProduct  $\iota$ )
  where
    isStableUnderTransfiniteCompositionOfShape J _ _ _ _ := ...
  
```

Here the ellipsis signifies the omission of code for clarity. `T.pushoutProduct ι` denotes the collection of morphisms $T_{\square \iota}$ to be defined in Theorem 4.3.12. □

Lemma 4.3.11. \hookrightarrow_{JM} *If f is a coproduct of shape J , then $f \square h$ is a coproduct of shape J .*

Proof. `instance [IsStableUnderCoproducts.{w} T] :`
`IsStableUnderCoproducts.{w} (T.pushoutProduct ι) where`
`isStableUnderCoproductsOfShape J := ...`

Here the ellipsis signifies the omission of code for clarity. `T.pushoutProduct ι` denotes the collection of morphisms $T_{\square \iota}$ to be defined in Theorem 4.3.12. \square

Theorem 4.3.12. $\hookrightarrow_{JM} \hookrightarrow_{JM}$ *Let T be a saturated collection. Fix a morphism ι and let $T_{\square \iota}$ be the collection of morphisms f such that $f \square \iota \in T$. Then $T_{\square \iota}$ is saturated.*

Proof. We define $T_{\square \iota}$ as

```
def pushoutProduct (T : MorphismProperty C) {A B : C} ( $\iota$  : A  $\longrightarrow$ 
  B) : MorphismProperty C := fun _ _ i  $\mapsto$  T ( $\iota \square i$ )
```

and then we combine Lemmas 4.3.8 to 4.3.11. \square

Chapter 5

Monomorphisms of Simplicial Sets

In the proof of Theorem 7.1.1, we need to consider the saturation of $\{\partial\Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$, the collection of all boundary inclusions. In Corollary 5.1.6, we prove that this is exactly the collection of monomorphisms of simplicial sets, which involves a brief investigation into *non-degenerate* simplices.

To formalize Corollary 5.1.6, we define the collection $\{\partial\Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$ using the inductive type approach described in Section 2.4 [↗_{JM}](#):

```
inductive BoundaryInclusion : {X Y : SSet} → (X → Y) → Prop
  | mk n : BoundaryInclusion ∂Δ[n].ι
```

```
def boundaryInclusions : MorphismProperty SSet := fun _ _ p ↦
  BoundaryInclusion p
```

Here $\partial\Delta[n].\iota$ is the n -th boundary inclusion defined via the canonical subset morphism described in Section 3.2. As we expect, a term of `boundaryInclusions f` reduces `f` to $\partial\Delta[n].\iota$ for some n .

5.1 Non-Degenerate Simplices

The non-degenerate simplices of a simplicial set can be thought of as its “building blocks”. In fact, a simplicial set can be recovered up to isomorphism if you know enough about its non-degenerate simplices [21, Rem. 19.20].

Definition 5.1.1. \hookrightarrow An n -simplex of a simplicial set X is *degenerate* if it is in the range of $Xf : X_m \rightarrow X_n$ for some morphism $f : [n] \rightarrow [m]$ with $m < n$. An n -simplex is *non-degenerate* if it is not degenerate.

So an n -simplex is non-degenerate if it doesn’t come from any “lesser simplex”. Furthermore, any simplex is the image of some non-degenerate simplex:

Theorem 5.1.2. \hookrightarrow Let $n \in \mathbb{N}$ and let $x \in X_n$. There exists $m \in \mathbb{N}$, $y \in X_m^{\text{nd}}$, and an epimorphism $f : [n] \rightarrow [m]$ such that $x = (Xf)y$.

The relationship between monomorphisms and boundary inclusions is given by the skeleton construction, which is a simplicial subset built up from non-degenerate simplices:

Definition 5.1.3. \hookrightarrow The n -skeleton of a simplicial set Y is the subset generated by the non-degenerate simplices of dimension $< n$:

$$\text{Sk}_n Y := \bigcup_{\substack{0 \leq i < n \\ x \in Y_i^{\text{nd}}}} \text{range}(x : \Delta^i \rightarrow Y).$$

Remark 5.1.4. In [15, 21], the n -skeleton as defined in Definition 5.1.3 is called the $(n - 1)$ -skeleton.

Note that $\text{Sk}_0 Y = \emptyset$,

$$\text{Sk}_n Y \subset \text{Sk}_n Y \cup \bigcup_{x \in Y_n^{\text{nd}}} \text{range } x = \text{Sk}_{n+1} Y,$$

and $\bigcup_{n \in \mathbb{N}} \text{Sk}_n Y = Y$ $\hookrightarrow \hookrightarrow \hookrightarrow$. This defines a filtration

$$\emptyset = \text{Sk}_0 Y \hookrightarrow \text{Sk}_1 Y \hookrightarrow \dots \hookrightarrow Y.$$

Given a simplicial subset $A \subseteq Y$, we obtain a similar filtration

$$A \cong A \cup \text{Sk}_0 Y \hookrightarrow A \cup \text{Sk}_1 Y \hookrightarrow \dots \hookrightarrow A \cup Y \cong Y,$$

which allows us to decompose $A \hookrightarrow Y$ as a composition of “nice” monomorphisms (see Theorem 5.1.5). Note that

$$A \cup \text{Sk}_{n+1} Y = A \cup \text{Sk}_n Y \cup \bigcup_{x \in Y_n^{\text{nd}} \setminus A_n} \text{range } x,$$

which leads us to the following theorem.

Theorem 5.1.5. *Let $A \subseteq Y$ be a simplicial subset, and let $n \in \mathbb{N}$. The following diagram is a pushout square.*

$$\begin{array}{ccc} \coprod_{Y_n^{\text{nd}} \setminus A_n} \partial \Delta^n & \longrightarrow & A \cup \text{Sk}_n Y \\ \downarrow & & \downarrow \\ \coprod_{Y_n^{\text{nd}} \setminus A_n} \Delta^n & \longrightarrow & A \cup \text{Sk}_{n+1} Y \end{array}$$

Proof. The horizontal maps are induced by $x : \Delta^n \rightarrow Y$ given by each $x \in Y_n^{\text{nd}} \setminus A_n$, and the vertical maps are the obvious inclusions. This theorem is formalized by Joël Riou in the case where $A = \text{range } i$ for some monomorphism $i : X \hookrightarrow Y$ \square , and a more general proof can be found in [21]. We omit the proof here. \square

Corollary 5.1.6. \square_{JM} *The collection of monomorphisms of simplicial sets is equal to the saturation of $\{\partial \Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$.*

Proof. Because the collection of monomorphisms is saturated and every boundary inclusion is a monomorphism, it is clear that every morphism in the saturation of $\{\partial \Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$ is a monomorphism.

To show that every monomorphism $i : X \hookrightarrow Y$ belongs to the saturation of $\{\partial \Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$, we apply Theorem 5.1.5 with $A = \text{range } i \subseteq Y$ \square . This gives us a filtration

$$(\text{range } i) \cup \text{Sk}_0 Y \hookrightarrow (\text{range } i) \cup \text{Sk}_1 Y \hookrightarrow \dots \hookrightarrow (\text{range } i) \cup Y,$$

such that each intermediate inclusion presents as a cobase change of a coproduct of boundary inclusions. Therefore, the inclusion $(\text{range } i) \cup \text{Sk}_0 Y \hookrightarrow (\text{range } i) \cup Y$ is in the saturation of $\{\partial\Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$. We have an isomorphism of arrows

$$(X \hookrightarrow Y) \cong ((\text{range } i) \cup \text{Sk}_0 Y \hookrightarrow (\text{range } i) \cup Y),$$

from which it follows that i is in the saturation of $\{\partial\Delta^n \hookrightarrow \Delta^n\}_{n \in \mathbb{N}}$. \square

Our formalization of Corollary 5.1.6 relies entirely on formalizations by Joël Riou; in fact, our formalization is effectively a restatement of his \hookrightarrow : without using the inductive `SaturatedClass` approach that we take, he shows that if you define a `MorphismProperty` by taking all transfinite compositions of cobase changes of coproducts of boundary inclusions (note that retracts aren't necessary here), then this is equal to the collection of monomorphisms in `SSet`.

The following theorem will be relevant for a proof in Chapter 6:

Theorem 5.1.7. \hookrightarrow *Let $A, B \subseteq X$ be subsets of a simplicial set X . Then $A \subseteq B$ if and only if every non-degenerate n -simplex of X contained in A_n is also in B_n .*

Proof. One direction is clear. Conversely, let B_n contain every non-degenerate n -simplex of X which is contained in A_n . We will show $A \subseteq B$.

Let $x \in A_n$. By Theorem 5.1.2, there exists $m \in \mathbb{N}$, $y \in A_m^{\text{nd}}$, and an epimorphism $f : [n] \rightarrow [m]$ such that $x = (Af)y$. Since A is a subset, this means $x = (Xf)y$. By assumption, $y \in B_m$, and it follows that $(Bf)y = x \in B_n$. \square

Corollary 5.1.8. \hookrightarrow *Let $A \subseteq X$ be a subset of a simplicial set X . Then $A = X$ if and only if A_n contains every non-degenerate n -simplex of X , for all $n \in \mathbb{N}$.*

We have established everything we need to know about monomorphisms to complete the proof of Theorem 7.1.1. Next, we turn to another distinguished collection of morphisms.

Chapter 6

Inner Anodyne Morphisms

Recall Remark 4.2.20. Given that we are interested in lifting properties and how they pertain to quasi-categories, it makes sense to consider the saturation of all inner horn inclusions $\Lambda_i^n \hookrightarrow \Delta^n$. In this chapter, we investigate that saturation. The major outcome is Theorem 6.0.3, which is essential to prove Theorem 7.1.1. The proof of Theorem 6.0.3 that we formalize requires extensive combinatorial detail—most of this chapter is devoted to those details.

Definition 6.0.1. $\hookrightarrow_{\text{JM}}$ The saturation of

$$\{\Lambda_i^{n+2} \hookrightarrow \Delta^{n+2} : 0 < i < n + 2\}_{n \in \mathbb{N}}$$

is called the collection of *inner anodyne* morphisms. Here we index by $n + 2$ because a horn Λ_i^n can only be inner for $n \geq 2$.

Remark 6.0.2. In fact, we formalize the collection of inner anodyne morphisms as

$$\text{llp}(\text{rlp}(\{\Lambda_i^{n+2} \hookrightarrow \Delta^{n+2} : 0 < i < n + 2\}_{n \in \mathbb{N}})), \hookrightarrow_{\text{JM}}$$

which is equivalent because the collection of inner horn inclusions admits the *small object argument* $\hookrightarrow_{\text{JM}}$. As formalized in Mathlib \hookrightarrow , the small object argument implies that if a collection of morphisms I satisfies certain conditions, then $\text{llp}(\text{rlp}(I))$ coincides with collection of morphisms given by taking all retracts of transfinite compositions of cobase changes of coproducts of morphisms in I . In this case, $\text{llp}(\text{rlp}(I))$ coincides with the saturation of I $\hookrightarrow_{\text{JM}}$.

For our needs it would suffice to define inner anodyne morphisms in terms of a saturation, but thanks to the small object argument, this doesn't matter.

Our formalization that inner horn inclusions admit the small object argument is adapted from a similar formalization by Joël Riou [↗](#).

To formalize Definition 6.0.1, we define the collection $\{\Lambda_i^{n+2} \hookrightarrow \Delta^n : 0 < i < n + 2\}_{n \in \mathbb{N}}$ using the inductive type approach described in Section 2.4 [↗](#)_{JM}:

```

inductive InnerHornInclusion : {X Y : SSet} → (X → Y) → Prop
  | mk {n : ℕ} {i : Fin (n+3)} (h0 : 0 < i) (hn : i < Fin.last
    (n+2)) :
    InnerHornInclusion Λ[n + 2, i].ι

```

```

def innerHornInclusions : MorphismProperty SSet := fun _ _ p ↦
  InnerHornInclusion p

```

As we expect, a term of `innerHornInclusions` `f` reduces `f` to $\Lambda[n + 2, i].\iota$ for some `n` and some `i`.

In this section we prove the following important theorem about inner anodyne morphisms:

Theorem 6.0.3. [15, Tag 007F] [↗](#)_{JM}

(a) *For every monomorphism $i : A \hookrightarrow B$ of simplicial sets, the map*

$$i \square (\Lambda_1^2 \hookrightarrow \Delta^2)$$

is inner anodyne.

(b) *The collection of inner anodyne morphisms is equal to the saturation of*

$$\{(\partial \Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2)\}_{n \in \mathbb{N}}.$$

Proof. Let $T = \{(\partial \Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2)\}_{n \in \mathbb{N}}$ and let $\overline{T}_{\square(\Lambda_1^2 \hookrightarrow \Delta^2)}$ be defined as in Theorem 4.3.12. It follows from Theorem 4.3.12 that $\overline{T}_{\square(\Lambda_1^2 \hookrightarrow \Delta^2)}$ is saturated.

In Lean, we define T using another inductive type [↗](#)_{JM}:

inductive HornBoundaryPushout : {X Y : SSet} → (X → Y) →

Prop

| mk (m : ℕ) : HornBoundaryPushout (Λ[2, 1].ι □ ∂Δ[m].ι)

def hornBoundaryPushouts : MorphismProperty SSet := fun _ _ p ↦
HornBoundaryPushout p

By definition, $\overline{T}_{\square(\Lambda_1^2 \hookrightarrow \Delta^2)}$ contains all boundary inclusions $\partial\Delta^n \hookrightarrow \Delta^n$, so $\overline{T}_{\square(\Lambda_1^2 \hookrightarrow \Delta^2)}$ contains all monomorphisms (Corollary 5.1.6). Therefore, to prove (a) it suffices to prove (b). In Theorem 6.1.1 we show that \overline{T} contains all inner anodyne morphisms, and in Theorem 6.2.17 we show that every morphism in \overline{T} is inner anodyne. \square

The rest of this chapter is devoted to the proofs of Theorem 6.1.1 and Theorem 6.2.17.

Remark 6.0.4. Theorem 6.0.3 appears as stated in [14, 15] and similarly in [21, Lem. 78.5]. The following formalization is derived from sketches of proofs which appear in the aforementioned texts, but includes far more detail. For example, in [15] the proof that $\overline{T}_{\square(\Lambda_1^2 \hookrightarrow \Delta^2)}$ is saturated is skipped, and in the equivalent proof of Theorem 6.2.17 many non-trivial details are omitted. [21] presents a more complete sketch of the proof of Theorem 6.2.17, but is still not rigorous enough for a formalization.

6.1 Generating Inner Horn Inclusions

Theorem 6.1.1. $\hookrightarrow_{\text{JM}}$ *The saturation of $\{(\partial\Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2)\}_{n \in \mathbb{N}}$ contains all inner anodyne morphisms.*

Proof. As above, let $T = \{(\partial\Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2)\}_{n \in \mathbb{N}}$. It suffices (Definition 4.2.19) to show that every inner horn inclusion $\Lambda_i^n \hookrightarrow \Delta^n$ belongs to \overline{T} . $\overline{T}_{\square(\Lambda_1^2 \hookrightarrow \Delta^2)}$ contains all inner horn inclusions, because it contains all monomorphisms, so \overline{T} contains all pushout-products $(\Lambda_i^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2)$. We show that each inner horn inclusion $\iota_i^n : \Lambda_i^n \hookrightarrow \Delta^n$ belongs to \overline{T} by showing that each is a retract of $f := \iota_i^n \square (\Lambda_1^2 \hookrightarrow \Delta^2)$.

Define maps $s : [n] \rightarrow [n] \times [2]$ and $r : [n] \times [2] \rightarrow [n]$ by

$$s(j) = \begin{cases} (j, 0) & \text{if } j < i \\ (j, 1) & \text{if } j = i \\ (j, 2) & \text{if } j > i \end{cases}$$

and

$$r(j, k) = \begin{cases} j & \text{if } j < i, k = 0 \\ j & \text{if } j > i, k = 2 \\ i & \text{otherwise.} \end{cases}$$

These determine maps $s : \Delta^n \rightarrow \Delta^n \times \Delta^2$ and $r : \Delta^n \times \Delta^2 \rightarrow \Delta^n \hookrightarrow_{\text{JM}} \hookrightarrow_{\text{JM}}$. We claim that the following is a retract diagram $\hookrightarrow_{\text{JM}}$:

$$\begin{array}{ccccc} \Lambda_i^n & \xrightarrow{\sigma} & (\Delta^n \times \Lambda_1^2) \amalg_{\Lambda_i^n \times \Lambda_1^2} (\Lambda_i^n \times \Delta^2) & \xrightarrow{\rho} & \Lambda_i^n \\ \iota_i^n \downarrow & & \downarrow f & & \downarrow \iota_i^n \\ \Delta^n & \xrightarrow{s} & \Delta^n \times \Delta^2 & \xrightarrow{r} & \Delta^n \end{array}$$

σ is defined by observing that s restricted to Λ_i^n lands in $\Lambda_i^n \times \Delta^2$, so σ is given by

$$\Lambda_i^n \xrightarrow{\iota_i^n} \Delta^n \xrightarrow{s} \Lambda_i^n \times \Delta^2 \xrightarrow{i_2} (\Delta^n \times \Lambda_1^2) \amalg_{\Lambda_i^n \times \Lambda_1^2} (\Lambda_i^n \times \Delta^2).$$

ρ is defined by observing that r restricted to $\Delta^n \times \Lambda_1^2$ and r restricted to $\Lambda_i^n \times \Delta^2$ both land in Λ_i^n , so ρ is induced by the two maps

$$\Delta^n \times \Lambda_1^2 \xrightarrow{\Delta^n \triangleleft \iota_1^2} \Delta^n \times \Delta^2 \xrightarrow{r} \Lambda_i^n$$

and

$$\Lambda_i^n \times \Delta^2 \xrightarrow{\iota_i^n \triangleright \Delta^2} \Delta^n \times \Delta^2 \xrightarrow{r} \Lambda_i^n.$$

Observe that $f \circ \sigma = f \circ (i_2 \circ s \circ \iota_i^n) = (\iota_i^n \triangleright \Delta^2) \circ s \circ \iota_i^n = s \circ \iota_i^n$, so the left square commutes. To check that the right square commutes ($\iota_i^n \circ \rho = r \circ f$),

it suffices to observe the following equalities:

$$(\iota_i^n \circ \rho) \circ i_1 = \iota_i^n \circ (r \circ (\Delta^n \triangleleft \iota_1^2)) = \iota_i^n \circ r \circ (f \circ i_1) = (r \circ f) \circ i_1$$

$$(\iota_i^n \circ \rho) \circ i_2 = \iota_i^n \circ (r \circ (\iota_i^n \triangleright \Delta^2)) = \iota_i^n \circ r \circ (f \circ i_2) = (r \circ f) \circ i_2$$

It is straightforward to check that $r \circ s = \text{id}$ and that $\rho \circ \sigma = \rho \circ (i_2 \circ s \circ \iota_i^n) = (r \circ (\iota_i^n \triangleright \Delta^2)) \circ (s \circ \iota_i^n) = \text{id}$. Therefore, $\iota_i^n : \Lambda_i^n \rightarrow \Delta^n$ presents as a retract of $(\Lambda_i^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2) \in \overline{T}$, so every inner anodyne morphism belongs to \overline{T} . \square

6.2 Generating Pushout-Products

Next, we show that every morphism belonging to \overline{T} is inner anodyne. Since the collection of inner anodyne morphisms is saturated it suffices (Definition 4.2.19) to show that every pushout-product

$$(\partial\Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2) : (\Delta^n \times \Lambda_1^2) \coprod_{\partial\Delta^n \times \Lambda_1^2} (\partial\Delta^n \times \Delta^2) \rightarrow \Delta^n \times \Delta^2$$

is inner anodyne. This is shown in Theorem 6.2.17, after some lengthy, combinatorial results are established. The following is a sketch of the proof:

For $n = 0$, $(\partial\Delta^0 \hookrightarrow \Delta^0) \square (\Lambda_1^2 \hookrightarrow \Delta^2)$ becomes $(\Lambda_1^2 \hookrightarrow \Delta^2)$ (Example 4.3.2), so we consider $n + 1$ for $n \in \mathbb{N}$. Observing the isomorphism

$$(\Delta^{n+1} \times \Lambda_1^2) \coprod_{\partial\Delta^{n+1} \times \Lambda_1^2} (\partial\Delta^{n+1} \times \Delta^2) \cong (\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2),$$

as subsets of $\Delta^{n+1} \times \Delta^2$ (Theorem 3.2.2), we will construct a filtration of subsets of $\Delta^{n+1} \times \Delta^2$,

$$(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \subset \dots \subset \Delta^{n+1} \times \Delta^2,$$

such that each intermediate inclusion presents as a cobase change of a morphism isomorphic to an inner horn inclusion. We will conclude that the

pushout-product is inner anodyne as a composition of inner anodyne morphisms. To construct the filtration, we will attach specific non-degenerate simplices of $\Delta^{n+1} \times \Delta^2$ to $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)$ in a particular order until the filtration terminates.

The rest of this section concerns the details of this proof and the details of their implementation in Lean.

6.2.1 Non-Degenerate Simplices of $\Delta^{n+1} \times \Delta^2$

For all $0 \leq a \leq b \leq n$, define maps $s_{b,a} : [n+2] \rightarrow [n+1] \times [2]$ by

$$s_{b,a}(j) = \begin{cases} (j, 0) & \text{if } 0 \leq j \leq a \\ (j-1, 1) & \text{if } a+1 \leq j \leq b+1 \\ (j-1, 2) & \text{if } b+2 \leq j \end{cases}$$

or

$$s_{b,a} = \langle (0, 0), \dots, (a, 0), (a, 1), \dots, (b, 1), (b+1, 2), \dots, (n+1, 2) \rangle.$$

For all $0 \leq a \leq b \leq n+1$, define maps $t_{b,a} : [n+3] \rightarrow [n+1] \times [2]$ by

$$t_{b,a}(j) = \begin{cases} (j, 0) & \text{if } 0 \leq j \leq a \\ (j-1, 1) & \text{if } a+1 \leq j \leq b+1 \\ (j-2, 2) & \text{if } b+2 \leq j \end{cases}$$

or

$$t_{b,a} = \langle (0, 0), \dots, (a, 0), (a, 1), \dots, (b, 1), (b, 2), \dots, (n+1, 2) \rangle.$$

$s_{b,a}$ and $t_{b,a}$ correspond to morphisms $\Delta^{n+2} \rightarrow \Delta^{n+1} \times \Delta^2$ and $\Delta^{n+3} \rightarrow \Delta^{n+1} \times \Delta^2$ which we will also refer to as $s_{b,a}$ and $t_{b,a}$ $\hookrightarrow_{\text{JM}}$ $\hookrightarrow_{\text{JM}}$. We let $\sigma_{b,a}$ and $\tau_{b,a}$ denote the subsets of $\Delta^{n+1} \times \Delta^2$ given by the range of $s_{b,a}$ and $t_{b,a}$ respectively $\hookrightarrow_{\text{JM}}$.

In Lean, we index s , t , σ , and τ by the dependent products established in Section 2.3.2:

abbrev $\sigma.s$ ($i : \Sigma_l (b : \text{Fin } n), \text{Fin } b.\text{succ}$) :
 $\Delta[n + 1] \rightarrow \Delta[n] \otimes \Delta[2] :=$
 $\text{yonedaEquiv.symm } (\sigma.\text{simplex } i)$

abbrev $\tau.t$ ($i : \Sigma_l (b : \text{Fin } (n + 1)), \text{Fin } b.\text{succ}$) :
 $\Delta[n + 2] \rightarrow \Delta[n] \otimes \Delta[2] :=$
 $\text{yonedaEquiv.symm } (\tau.\text{simplex } i)$

def $\sigma.\text{subcomplex}$ ($i : \Sigma_l (b : \text{Fin } n), \text{Fin } b.\text{succ}$) :=
 $\text{Subcomplex.ofSimplex } (\sigma.\text{simplex } i)$

def $\tau.\text{subcomplex}$ ($i : \Sigma_l (b : \text{Fin } (n + 1)), \text{Fin } b.\text{succ}$) :=
 $\text{Subcomplex.ofSimplex } (\tau.\text{simplex } i)$

Lemma 6.2.1. $\hookrightarrow_{\text{JM}}$ $s_{b,a} \in (\Delta^{n+1} \times \Delta^2)_{n+2}$ and $t_{b,a} \in (\Delta^{n+1} \times \Delta^2)_{n+3}$ are non-degenerate.

Proof. The proof is omitted here, but the formalized equivalent can be found at the above link. \square

Furthermore, there is a bijection between non-degenerate $(n + 3)$ -simplices of $\Delta^{n+1} \times \Delta^2$ and pairs $0 \leq a \leq b \leq n + 1$, via $(b, a) \mapsto t_{b,a}$. We will express this in terms of the dependent product defined in Section 2.3.2.

Theorem 6.2.2. $\hookrightarrow_{\text{JM}}$ *There is a bijection*

$$(b : \text{Fin } (n + 2)) \times (\text{Fin } (b + 1)) \simeq (\Delta^{n+1} \times \Delta^2)_{n+3}^{\text{nd}}.$$

Proof. The following is a sketch of the proof. We map a pair (b, a) to $t_{b,a}$. It is straightforward to show that this mapping is injective. To show that the mapping is surjective, let $y \in (\Delta^{n+1} \times \Delta^2)_{n+3}^{\text{nd}}$ and let

$$S = \{i \in [n + 3] : (yi)_2 = 1\}$$

(the set of $i \in [n + 3]$ such that the second component of yi is 1). If S is empty, we reach a contradiction. Let a' be the minimum of S and let b' be

the maximum of S . Let $a = (ya')_1$ and let $b = (yb')_1$. It can be checked that $y = t_{b,a}$. \square

In the proof of Theorem 6.2.17, we need two general results about non-degenerate simplices of simplicial sets of the form $\Delta^m \times \Delta^n$. Like Δ^n , these non-degenerate simplices are fairly easy to describe but the results we need are tedious to develop and prove. We omit their proofs, but include links to their associated formalizations by Joël Riou:

Theorem 6.2.3. \hookrightarrow *Let $m, n \in \mathbb{N}$, and let $A \subseteq \Delta^m \times \Delta^n$ be a subset. Then $A = \Delta^m \times \Delta^n$ if and only if A contains all non-degenerate $(m+n)$ -simplices of $\Delta^m \times \Delta^n$.*

Proof. This reduces to Corollary 5.1.8 by showing that every non-degenerate simplex of $\Delta^m \times \Delta^n$ is contained in a simplicial subset generated by a non-degenerate $(m+n)$ -simplex. \square

Theorem 6.2.4. \hookrightarrow *$x \in (\Delta^m \times \Delta^n)_k$ is non-degenerate if and only if $x : \Delta^k \rightarrow \Delta^m \times \Delta^n$ is a monomorphism.*

From Theorem 6.2.4 it follows that $s_{b,a}$ and $t_{b,a}$ are monomorphisms. From Theorem 6.2.3 and Lemma 3.2.5, it follows that any subset of $\Delta^{n+1} \times \Delta^2$ which contains all $\tau_{a,b}$'s must be $\Delta^{n+1} \times \Delta^2$ itself. This is how we ensure that our filtration terminates.

With Definition 3.2.12 in mind, we record some basic observations about $\sigma_{b,a}$ and $\tau_{b,a}$:

Lemma 6.2.5. $\hookrightarrow_{\text{JM}}$ *Let $0 \leq a < b \leq n$. The $(a+1)$ -th face of $\sigma_{b,a+1}$ is the $(a+1)$ -th face of $\sigma_{b,a}$.*

Proof. It suffices to show that $s_{b,a+1} \circ \delta^{a+1} = s_{b,a} \circ \delta^{a+1}$. We can observe this by representing $s_{b,a+1} \circ \delta^{a+1}$ as

$$\langle (0, 0), \dots, (a, 0), (\widehat{a+1, 0}), (a+1, 1), \dots, (b, 1), (b+1, 2), \dots, (n+1, 2) \rangle$$

and $s_{b,a} \circ \delta^{a+1}$ as

$$\langle (0, 0), \dots, (a, 0), (\widehat{a, 1}), (a+1, 1), \dots, (b, 1), (b+1, 2), \dots, (n+1, 2) \rangle.$$

□

Similarly, we obtain the following lemma.

Lemma 6.2.6. \square_{JM} *Let $0 \leq a < b \leq n + 1$. The $(a + 1)$ -th face of $\tau_{b,a+1}$ is the $(a + 1)$ -th face of $\tau_{b,a}$.*

Proof. It suffices to show that $t_{b,a+1} \circ \delta^{a+1} = t_{b,a} \circ \delta^{a+1}$. We can observe this by representing $t_{b,a+1} \circ \delta^{a+1}$ as

$$\langle (0, 0), \dots, (a, 0), (\widehat{a+1, 0}), (a+1, 1), \dots, (b, 1), (b, 2), \dots, (n+1, 2) \rangle$$

and $t_{b,a} \circ \delta^{a+1}$ as

$$\langle (0, 0), \dots, (a, 0), (\widehat{a, 1}), (a+1, 1), \dots, (b, 1), (b, 2), \dots, (n+1, 2) \rangle.$$

□

Lemma 6.2.7. \square_{JM} \square_{JM} *Let $0 \leq a \leq b \leq n$. The $(b + 2)$ -th face of $\tau_{b,a}$, the $(b + 2)$ -th face of $\tau_{b+1,a}$, and $\sigma_{b,a}$ are all equal.*

Proof. It suffices to show that $t_{b,a} \circ \delta^{b+2} = t_{b+1,a} \circ \delta^{b+2} = s_{b,a}$. We can observe this by representing $t_{b,a} \circ \delta^{b+2}$ as

$$\langle (0, 0), \dots, (a, 0), (a, 1), \dots, (b, 1), (\widehat{b, 2}), (b+1, 2), \dots, (n+1, 2) \rangle,$$

$t_{b+1,a} \circ \delta^{b+2}$ as

$$\langle (0, 0), \dots, (a, 0), (a, 1), \dots, (b, 1), (\widehat{b+1, 1}), (b+1, 2), \dots, (n+1, 2) \rangle,$$

and $s_{b,a}$ as

$$\langle (0, 0), \dots, (a, 0), (a, 1), \dots, (b, 1), (b+1, 2), \dots, (n+1, 2) \rangle.$$

□

Now that we have defined the non-degenerate simplices of interest, we define a filtration.

6.2.2 Constructing a Filtration

We construct a filtration by attaching the following subsets of $\Delta^{n+1} \times \Delta^2$ to $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)$ in the following order:

$$\sigma_{0,0}, \sigma_{1,0}, \sigma_{1,1}, \sigma_{2,0}, \sigma_{2,1}, \sigma_{2,2}, \dots, \sigma_{n-1,0}, \dots, \sigma_{n-1,n-1},$$

followed by

$$\tau_{0,0}, \tau_{1,0}, \tau_{1,1}, \tau_{2,0}, \tau_{2,1}, \tau_{2,2}, \dots, \tau_{n,0}, \dots, \tau_{n,n}.$$

The $t_{b,a}$'s range through all the non-degenerate $(n+3)$ -simplices of $\Delta^{n+1} \times \Delta^2$ (Theorem 6.2.2), so by Lemma 3.2.5 and Theorem 6.2.3,

$$((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cup \bigcup \sigma_{b,a} \cup \bigcup \tau_{b,a} = \Delta^{n+1} \times \Delta^2.$$

We split the filtration into two parts, one for attaching the σ 's and one for attaching the τ 's. We define the σ filtration as such $\boxrightarrow_{\text{JM}}$:

```
def  $\sigma$ .filtration {n : ℕ} (i :  $\Sigma_l$  (b : Fin (n + 1)), Fin (b + 1)) :
  ( $\Delta$ [n + 1]  $\otimes$   $\Delta$ [2]).Subcomplex :=
   $\partial\Delta$ [n + 1].unionProd  $\Lambda$ [2, 1]  $\sqcup$  ( $\bigsqcup$  (j) ( $\_ : j \leq i$ ),  $\sigma$ .subcomplex j)
```

which translates to the following definition for $0 \leq a \leq b \leq n$:

$$S_{b,a} := ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cup \left(\bigcup_{(y,x) \leq (b,a)} \sigma_{y,x} \right),$$

where $(y, x) \leq (b, a)$ is given by the *lexicographic* order defined in Definition 2.3.1. We define the τ filtration similarly $\boxrightarrow_{\text{JM}}$:

```
def  $\tau$ .filtration {n : ℕ} (i :  $\Sigma_l$  (b : Fin (n + 2)), Fin b.succ) :
  :
  ( $\Delta$ [n + 1]  $\otimes$   $\Delta$ [2]).Subcomplex :=
  ( $\sigma$ .filtration  $\top$ )  $\sqcup$  ( $\bigsqcup$  (j) ( $\_ : j \leq i$ ),  $\tau$ .subcomplex j)
```

which translates to the following definition for $0 \leq a \leq b \leq n + 1$:

$$T_{b,a} := S_{n,n} \cup \left(\bigcup_{(y,x) \leq (b,a)} \tau_{y,x} \right).$$

The filtration is thus given by $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \subset S_{0,0} \subset \dots \subset S_{n,n} \subset T_{0,0} \subset \dots \subset T_{n+1,n+1} \cong \Delta^{n+1} \times \Delta^2$.

6.2.3 Properties of the Filtration

We will show that every inclusion in the filtration presents as a cobase change of (a morphism which is isomorphic to) an inner horn inclusion and is therefore inner anodyne. We do this in the following order:

1. $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \subset S_{0,0}$,
2. $S_{0,0} \subset S_{n,n}$,
3. $S_{n,n} \subset T_{0,0}$,
4. $T_{0,0} \subset T_{n+1,n+1}$.

1. $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \subset S_{0,0}$

Lemma 6.2.8. $\hookrightarrow_{\text{JM}}$ *The inclusion $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \hookrightarrow S_{0,0}$ is inner anodyne.*

Proof. We claim that the following diagram of subsets of $\Delta^{n+1} \times \Delta^2$ is a pushout square $\hookrightarrow_{\text{JM}}$:

$$\begin{array}{ccc} s_{0,0}(\Lambda_1^{n+2}) & \longrightarrow & (\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \\ \downarrow & & \downarrow \\ \sigma_{0,0} & \longrightarrow & S_{0,0} \end{array}$$

First, we show that the diagram commutes. Note that $s_{0,0}(\Lambda_1^{n+2}) = \bigcup_{i \neq 1} s_{0,0}(\Delta^{[n+2] \setminus i})$ is the union of all the faces of $\sigma_{0,0}$ except the first. To show that $s_{0,0}(\Lambda_1^{n+2})$ is

contained in $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)$, we make the following observations about the faces of $\sigma_{0,0}$:

- $\boxrightarrow_{\text{JM}}$ Every face $s_{0,0}(\Delta^{[n+2]\setminus i})$ is contained in $\partial\Delta^{n+1} \times \Delta^2 \subset (\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)$ except the 0-th and first face. The 0-th face is generated by $s_{0,0} \circ \delta^0$, which looks like

$$\langle \widehat{(0,0)}, (0,1), (1,2), \dots, (n+1,2) \rangle.$$

and the first face is generated by $s_{0,0} \circ \delta^1$, which looks like

$$\langle (0,0), \widehat{(0,1)}, (1,2), \dots, (n+1,2) \rangle.$$

In particular, $s_{0,0} \circ \delta^0$ and $s_{0,0} \circ \delta^1$ are surjective in the first component. By the same “visual” argument, it is clear that any other $s_{0,0} \circ \delta^i$ is not surjective in the first component.

- $\boxrightarrow_{\text{JM}}$ The 0-th face is contained in $\Delta^{n+1} \times \Lambda_1^2 \subset (\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)$, because the second component of $s_{0,0} \circ \delta^0$ does not have 0 in its image.

It follows that $s_{0,0}(\Lambda_1^{n+2})$ is contained in $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \boxrightarrow_{\text{JM}}$. The rest of the morphisms in the diagram are the obvious inclusions, so the diagram commutes. To show that the diagram is a pushout square, it suffices (Theorem 3.2.2) to show that $S_{0,0} = (\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \cup \sigma_{0,0}$ (which is true by construction) and that $s_{0,0}(\Lambda_1^{n+2}) = ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cap \sigma_{0,0}$, i.e.

$$((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cap \sigma_{0,0} \subseteq s_{0,0}(\Lambda_1^{n+2}).$$

Applying Corollary 3.2.14 with the monomorphism $s_{0,0}$ (Theorem 6.2.4), we must show that $s_{0,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cap \sigma_{0,0}$. Since $s_{0,0}(\Delta^{[n+2]\setminus 1}) \subset \sigma_{0,0}$, we must show that

$$s_{0,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq (\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2). \boxrightarrow_{\text{JM}}$$

We split this into the following observations about $s_{0,0}(\Delta^{[n+2]\setminus 1})$, the first face of $\sigma_{0,0}$:

- $s_{0,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq \Delta^{n+1} \times \Lambda_1^2$, because the image of the second component of $s_{0,0} \circ \delta^1$ is $\{0, 2\}$.
- $s_{0,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq \partial\Delta^{n+1} \times \Delta^2$, as observed above.

Therefore, $s_{0,0}(\Lambda_1^{n+2}) = ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cap \sigma_{0,0}$ and the diagram is a pushout square.

$s_{0,0}$ is a monomorphism (Theorem 6.2.4), so we have an isomorphism of arrows

$$(s_{0,0}(\Lambda_1^{n+2}) \rightarrow \sigma_{0,0}) \cong (\Lambda_1^{n+2} \rightarrow \Delta^{n+2})$$

and the pushout square above implies the inclusion $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \hookrightarrow S_{0,0}$ is a cobase change of a morphism isomorphic to an inner horn inclusion. Therefore, the inclusion $(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \hookrightarrow S_{0,0}$ is inner anodyne. \square

Remark 6.2.9. The rest of the lemmas in this section are structured similarly, with very similar arguments. In particular, many diagrams must be shown to be pushout squares, each with slightly different indices and slightly different arguments. Most of these arguments are adapted from [21], modulo a few typos and missing cases.

2. $S_{0,0} \subset S_{n,n}$

To prove that the inclusion $S_{0,0} \hookrightarrow S_{n,n}$ is inner anodyne, it suffices $\boxrightarrow_{\text{JM}}$ $\boxrightarrow_{\text{JM}}$ to prove the following lemma:

Lemma 6.2.10. $\boxrightarrow_{\text{JM}}$ For each pair $0 \leq a \leq b \leq n$ with $(b, a) < (n, n)$, the inclusion $S_{b,a} \hookrightarrow S_{\text{succ}(b,a)}$ is inner anodyne.

Proof. We claim that the following diagram of subsets of $\Delta^{n+1} \times \Delta^2$ is a pushout square for all $(b, a) < (n, n)$ $\boxrightarrow_{\text{JM}}$:

$$\begin{array}{ccc} s_{\text{succ}(b,a)}(\Lambda_{(\text{succ}(b,a))_2+1}^{n+2}) & \longrightarrow & S_{b,a} \\ \downarrow & & \downarrow \\ \sigma_{\text{succ}(b,a)} & \longrightarrow & S_{\text{succ}(b,a)} \end{array}$$

Since $(b, a) < (n, n)$, we have two cases:

1. (b, a) with $0 \leq a < b \leq n$ and $\text{succ}(b, a) = (b, a + 1)$,
2. or (b, b) with $0 \leq b < n$ and $\text{succ}(b, b) = (b + 1, 0)$.

Thus, we have to prove that the following diagrams are pushout squares:

$$\begin{array}{ccc} s_{b,a+1}(\Lambda_{a+2}^{n+2}) & \longrightarrow & S_{b,a} \\ \downarrow & & \downarrow \\ \sigma_{b,a+1} & \longrightarrow & S_{b,a+1} \end{array}$$

for all $0 \leq a < b \leq n$, and

$$\begin{array}{ccc} s_{b+1,0}(\Lambda_1^{n+2}) & \longrightarrow & S_{b,b} \\ \downarrow & & \downarrow \\ \sigma_{b+1,0} & \longrightarrow & S_{b+1,0} \end{array}$$

for all $0 \leq b < n$. The key advantage of the lexicographic successor defined in Definition 2.3.2 is that it allows us to bundle these two cases into one. We defer these proofs to Lemma 6.2.11 and Lemma 6.2.12. Note that Λ_{a+2}^{n+2} and Λ_1^{n+2} are both inner horns (the former because $a + 2 \leq b + 1 < n + 2$). Since $s_{b,a+1}, s_{b+1,0}$ are monomorphisms (Theorem 6.2.4), we have isomorphisms of arrows

$$(s_{b,a+1}(\Lambda_{a+2}^{n+2}) \rightarrow \sigma_{b,a+1}) \cong (\Lambda_{a+2}^{n+2} \rightarrow \Delta^{n+2})$$

and

$$(s_{b+1,0}(\Lambda_1^{n+2}) \rightarrow \sigma_{b+1,0}) \cong (\Lambda_1^{n+2} \rightarrow \Delta^{n+2}),$$

so the above pushout squares imply the inclusions $S_{b,a} \hookrightarrow S_{b,a+1}$ and $S_{b,b} \hookrightarrow S_{b+1,0}$ are cobase changes of morphisms isomorphic to inner horn inclusions. Therefore, the inclusion $S_{b,a} \hookrightarrow S_{\text{succ}(b,a)}$ is inner anodyne for all $(b, a) < (n, n)$. \square

Lemma 6.2.11. \square_{JM} For all $0 \leq a < b \leq n$, the following diagram is a pushout square:

$$\begin{array}{ccc} s_{b,a+1}(\Lambda_{a+2}^{n+2}) & \longrightarrow & S_{b,a} \\ \downarrow & & \downarrow \\ \sigma_{b,a+1} & \longrightarrow & S_{b,a+1} \end{array}$$

Proof. First, we show that the diagram commutes. Note that $s_{b,a+1}(\Lambda_{a+2}^{n+2}) = \bigcup_{i \neq a+2} s_{b,a+1}(\Delta^{[n+2] \setminus i})$ is the union of all the faces of $\sigma_{b,a+1}$ except the $(a+2)$ -th. To show that $s_{b,a+1}(\Lambda_{a+2}^{n+2})$ is contained in $S_{b,a}$ we make the following observations about the faces of $\sigma_{b,a+1}$:

- \square_{JM} Every face $s_{b,a+1}(\Delta^{[n+2] \setminus i})$ is contained in $\partial\Delta^{n+1} \times \Delta^2 \subset S_{b,a}$ except the $(a+1)$ -th and $(a+2)$ -th face. The $(a+1)$ -th face is generated by $s_{b,a+1} \circ \delta^{a+1}$, which looks like

$$\langle (0,0), \dots, (a,0), (\widehat{a+1,0}), (a+1,1), \dots, (b,1), (b+1,2), \dots, (n+1,2) \rangle$$

and the $(a+2)$ -th face is generated by $s_{b,a+1} \circ \delta^{a+2}$, which looks like

$$\langle (0,0), \dots, (a+1,0), (\widehat{a+1,1}), (a+2,1), \dots, (b,1), (b+1,2), \dots, (n+1,2) \rangle.$$

In particular, $s_{b,a+1} \circ \delta^{a+1}$ and $s_{b,a+1} \circ \delta^{a+2}$ are surjective in the first component. By the same “visual” argument, it is clear that any other $s_{b,a+1} \circ \delta^i$ is not surjective in the first component.

- \square_{JM} The $(a+1)$ -th face of $\sigma_{b,a+1}$ is the $(a+1)$ -th face of $\sigma_{b,a}$ (Lemma 6.2.5), so is contained in $\sigma_{b,a} \subset S_{b,a}$.

It follows that $s_{b,a+1}(\Lambda_{a+2}^{n+2})$ is contained in $S_{b,a}$ \square_{JM} . The rest of the morphisms in the diagram are the obvious inclusions, so the diagram commutes. To show that the diagram is a pushout square, it suffices (Theorem 3.2.2) to show that $S_{b+1,a} = S_{b,a} \cup \sigma_{b,a+1}$ (which is true by construction) and that $s_{b,a+1}(\Lambda_{a+2}^{n+2}) = S_{b,a} \cap \sigma_{b,a+1}$. Applying Corollary 3.2.14 with the monomorphism $s_{b,a+1}$ (Theorem 6.2.4), it suffices (by the same argument given in Lemma 6.2.8) to show that

$s_{b,a+1} (\Delta^{[n+2]\setminus a+2}) \not\subseteq ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cup \left(\bigcup_{(y,x) \leq (b,a)} \sigma_{y,x} \right)$. \square_{JM}

We split this into the following observations about $s_{b,a+1} (\Delta^{[n+2]\setminus a+2})$, the $(a+2)$ -th face of $\sigma_{b,a+1}$:

- $\square_{\text{JM}} s_{b,a+1} (\Delta^{[n+2]\setminus a+2}) \not\subseteq \Delta^{n+1} \times \Lambda_1^2$, because the second component of $s_{b,a+1} \circ \delta^{a+2}$ contains $\{0, 2\}$.
- $\square_{\text{JM}} s_{b,a+1} (\Delta^{[n+2]\setminus a+2}) \not\subseteq \partial\Delta^{n+1} \times \Delta^2$, as shown above
- $\square_{\text{JM}} s_{b,a+1} (\Delta^{[n+2]\setminus a+2}) \not\subseteq \sigma_{y,x}$ for any $(y, x) \leq (b, a)$:
 - if $y = b$, then $x \leq a$ and $(a+1, 0)$ is in the image of $s_{b,a+1} \circ \delta^{a+2}$, but not in the image of $s_{b,x}$.
 - if $y < b$, then $x < b$ and
 - * if $a+1 < b$, then $(b, 1)$ is in the image of $s_{b,a+1} \circ \delta^{a+2}$, but not in the image of $s_{y,x}$.
 - * if $a+1 = b$, then $(b, 0)$ is in the image of $s_{b,b} \circ \delta^{b+1}$, but not in the image of $s_{y,x}$.

Therefore, $s_{b,a+1} (\Lambda_{a+2}^{n+2}) = S_{b,a} \cap \sigma_{b,a+1}$ and the diagram is a pushout square. \square

Lemma 6.2.12. \square_{JM} For all $0 \leq b < n$, the following diagram is a pushout square:

$$\begin{array}{ccc} s_{b+1,0}(\Lambda_1^{n+2}) & \longrightarrow & S_{b,b} \\ \downarrow & & \downarrow \\ \sigma_{b+1,0} & \longrightarrow & S_{b+1,0} \end{array}$$

Proof. First, we show that the diagram commutes. Note that $s_{b+1,0} (\Lambda_1^{n+2}) = \bigcup_{i \neq 1} s_{b+1,0} (\Delta^{[n+2]\setminus i})$ is the union of all the faces of $\sigma_{b+1,0}$ except the first. To show that $s_{b+1,0} (\Lambda_1^{n+2})$ is contained in $S_{b,b}$ we make the following observations about the faces of $\sigma_{b+1,0}$:

- \square_{JM} Every face $s_{b+1,0} (\Delta^{[n+2]\setminus i})$ is contained in $\partial\Delta^{n+1} \times \Delta^2 \subset S_{b,b}$ except the 0-th and first face. The 0-th face is generated by $s_{b+1,0} \circ \delta^0$, which

looks like

$$\langle (\widehat{(0,0)}, (0,1), \dots, (b+1,1), (b+2,2), \dots, (n+1,2) \rangle$$

and the first face is generated by $s_{b+1,0} \circ \delta^1$, which looks like

$$\langle (0,0), (\widehat{(0,1)}, (1,1), \dots, (b+1,1), (b+2,2), \dots, (n+1,2) \rangle$$

In particular, $s_{b+1,0} \circ \delta^0$ and $s_{b+1,0} \circ \delta^1$ are surjective in the first component. By the same “visual” argument, it is clear that any other $s_{b+1,0} \circ \delta^i$ is not surjective in the first component.

- $\hookrightarrow_{\text{JM}}$ The 0-th face is contained in $\Delta^{n+1} \times \Lambda_1^2 \subset S_{b,b}$, because 0 is not in the image of the second component of $s_{b+1,0} \circ \delta^0$.

It follows that $s_{b+1,0}(\Lambda_1^{n+2})$ is contained in $S_{b,b} \hookrightarrow_{\text{JM}}$. The rest of the morphisms in the diagram are the obvious inclusions, so the diagram commutes. To show that the diagram is a pushout square, it suffices (Theorem 3.2.2) to show that $S_{b+1,0} = S_{b,b} \cup \sigma_{b+1,0}$ (which is true by construction) and that $s_{b+1,0}(\Lambda_1^{n+2}) = S_{b,b} \cap \sigma_{b+1,0}$. Applying Corollary 3.2.14 with the monomorphism $s_{b+1,0}$ (Theorem 6.2.4), it suffices (by the same argument given in Lemma 6.2.8) to show that

$$s_{b+1,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cup \left(\bigcup_{(y,x) \leq (b,b)} \sigma_{y,x} \right). \hookrightarrow_{\text{JM}}$$

We split this into the following observations about $s_{b+1,0}(\Delta^{[n+2]\setminus 1})$, the first face of $\sigma_{b+1,0}$:

- $\hookrightarrow_{\text{JM}}$ $s_{b+1,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq \Delta^{n+1} \times \Lambda_1^2$, because the second component of $s_{b+1,0} \circ \delta^1$ is surjective.
- $\hookrightarrow_{\text{JM}}$ $s_{b+1,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq \partial\Delta^{n+1} \times \Delta^2$, as shown above
- $\hookrightarrow_{\text{JM}}$ $s_{b+1,0}(\Delta^{[n+2]\setminus 1}) \not\subseteq \sigma_{y,x}$ for any $(y,x) \leq (b,b)$, because $(b+1,0)$ is in the image of $s_{b+1,0} \circ \delta^1$, but not in the image of any such $\sigma_{y,x}$.

Therefore, $s_{b+1,0}(\Lambda_1^{n+2}) = S_{b,b} \cap \sigma_{b+1,0}$ and the diagram is a pushout square. \square

3. $S_{n,n} \subset T_{0,0}$

Lemma 6.2.13. $\hookrightarrow_{\text{JM}}$ *The inclusion $S_{n,n} \hookrightarrow T_{0,0}$ is inner anodyne.*

Proof. We claim that the following diagram of subsets of $\Delta^{n+1} \times \Delta^2$ is a pushout square $\hookrightarrow_{\text{JM}}$:

$$\begin{array}{ccc} t_{0,0}(\Lambda_1^{n+3}) & \longrightarrow & S_{n,n} \\ \downarrow & & \downarrow \\ \tau_{0,0} & \longrightarrow & T_{0,0} \end{array}$$

First, we show that the diagram commutes. Note that $t_{0,0}(\Lambda_1^{n+3}) = \bigcup_{i \neq 1} t_{0,0}(\Delta^{[n+3] \setminus i})$ is the union of all faces of $\tau_{0,0}$ except the first. To show that $t_{0,0}(\Lambda_1^{n+3})$ is contained in $S_{n,n}$ we make the following observations about the faces of $\tau_{0,0}$:

- $\hookrightarrow_{\text{JM}}$ Every face $t_{0,0}(\Delta^{[n+3] \setminus i})$ is contained in $\partial\Delta^{n+1} \times \Delta^2 \subset S_{n,n}$ except the 0-th, first, and second. The 0-th face is generated by $t_{0,0} \circ \delta^0$, which looks like

$$\langle \widehat{(0,0)}, (0,1), (0,2), \dots, (n+1,2) \rangle,$$

the first face is generated by $t_{0,0} \circ \delta^1$, which looks like

$$\langle (0,0), \widehat{(0,1)}, (0,2), \dots, (n+1,2) \rangle,$$

the second face is generated by $t_{0,0} \circ \delta^2$, which looks like

$$\langle (0,0), (0,1), \widehat{(0,2)}, \dots, (n+1,2) \rangle,$$

In particular, $t_{0,0} \circ \delta^0$, $t_{0,0} \circ \delta^1$, and $t_{0,0} \circ \delta^2$ are surjective in the first component. By the same “visual” argument, it is clear that any other $t_{0,0} \circ \delta^i$ is not surjective in the first component.

- The 0-th face is contained in $\Delta^{n+1} \times \Lambda_1^2 \subset S_{n,n}$, because the second component of $t_{0,0} \circ \delta^0$ does not have 0 in its image.
- $\hookrightarrow_{\text{JM}}$ The second face of $\tau_{0,0}$ is $\sigma_{0,0} \subset S_{n,n}$ (Lemma 6.2.7).

It follows that $t_{0,0}(\Lambda_1^{n+3})$ is contained in $S_{n,n}$ $\hookrightarrow_{\text{JM}}$. The rest of the morphisms in the diagram are the obvious inclusions, so the diagram commutes. To show

that the diagram is a pushout square, it suffices (Theorem 3.2.2) to show that $T_{0,0} = S_{n,n} \cup \tau_{0,0}$ (which is true by construction) and that $t_{0,0}(\Lambda_1^{n+3}) = S_{n,n} \cap \tau_{0,0}$. Applying Corollary 3.2.14 with the monomorphism $t_{0,0}$ (Theorem 6.2.4), it suffices (by the same argument given in Lemma 6.2.8) to show that

$$t_{0,0}(\Delta^{[n+3]\setminus 1}) \not\subseteq ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2)) \cup \left(\bigcup_{(y,x) \leq (n,n)} \sigma_{y,x} \right). \quad \square_{\text{JM}}$$

We split this into the following observations about $t_{0,0}(\Delta^{[n+3]\setminus 1})$, the first face of $\tau_{0,0}$:

- $t_{0,0}(\Delta^{[n+3]\setminus 1}) \not\subseteq \Delta^{n+1} \times \Lambda_1^2$, because the image of the second component of $t_{0,0} \circ \delta^1$ is $\{0, 2\}$.
- $t_{0,0}(\Delta^{[n+3]\setminus 1}) \not\subseteq \partial\Delta^{n+1} \times \Delta^2$, as observed above.
- $t_{0,0}(\Delta^{[n+3]\setminus 1}) \not\subseteq \sigma_{y,x}$ for any $0 \leq x \leq y \leq n$, because $(0, 2)$ is in the image of $t_{0,0} \circ \delta^1$ but not in the image of any $s_{y,x}$.

Therefore, $t_{0,0}(\Lambda_1^{n+3}) = S_{n,n} \cap \tau_{0,0}$ and the diagram is a pushout square. It follows that the inclusion $S_{n,n} \hookrightarrow T_{0,0}$ is inner anodyne by the same argument given in Lemma 6.2.8. \square

4. $T_{0,0} \subset T_{n+1,n+1}$

To prove that the inclusion $T_{0,0} \hookrightarrow T_{n+1,n+1}$ is inner anodyne, it suffices \square_{JM} \square_{JM} to prove the following lemma:

Lemma 6.2.14. \square_{JM} *For each pair $0 \leq a \leq b \leq n+1$ with $(b, a) < (n+1, n+1)$, the inclusion $T_{b,a} \hookrightarrow T_{\text{succ}(b,a)}$ is inner anodyne.*

Proof. By the same argument given in Lemma 6.2.10, the proof reduces to Lemma 6.2.15 and Lemma 6.2.16. \square

Lemma 6.2.15. \square_{JM} *For all $0 \leq a < b \leq n+1$, the following diagram is a pushout square:*

$$\begin{array}{ccc} t_{b,a+1}(\Lambda_{a+2}^{n+3}) & \longrightarrow & T_{b,a} \\ \downarrow & & \downarrow \\ \tau_{b,a+1} & \longrightarrow & T_{b,a+1} \end{array}$$

Proof. First, we show that the diagram commutes. Note that $t_{b,a+1}(\Lambda_{a+2}^{n+3}) = \bigcup_{i \neq a+2} t_{b,a+1}(\Delta^{[n+3] \setminus i})$ is the union of all the faces of $\tau_{b,a+1}$ except the $(a+2)$ -th. To show that $t_{b,a+1}(\Lambda_{a+2}^{n+3})$ is contained in $T_{b,a}$ we make the following observations about the faces of $\tau_{b,a+1}$:

- $\boxrightarrow_{\text{JM}}$ Every face $t_{b,a+1}(\Delta^{[n+3] \setminus i})$ of $\tau_{b,a+1}$ is contained in $\partial\Delta^{n+1} \times \Delta^2 \subset T_{b,a}$ except the $(a+1)$ -th, $(a+2)$ -th, $(b+1)$ -th, and $(b+2)$ -th face. The $(a+1)$ -th face is generated by $t_{b,a+1} \circ \delta^{a+1}$, which looks like

$$\langle (0, 0), \dots, (a, 0), (\widehat{a+1, 0}), (a+1, 1), \dots, (b, 1), (b, 2), \dots, (n+1, 2) \rangle,$$

the $(a+2)$ -th face is generated by $t_{b,a+1} \circ \delta^{a+2}$, which looks like

$$\langle (0, 0), \dots, (a+1, 0), (\widehat{a+1, 1}), (a+2, 1), \dots, (b, 1), (b, 2), \dots, (n+1, 2) \rangle,$$

the $(b+1)$ -th face is generated by $t_{b,a+1} \circ \delta^{b+1}$, which looks like

$$\langle (0, 0), \dots, (a+1, 0), (a+1, 1), \dots, (b-1, 1), (\widehat{b, 1}), (b, 2), \dots, (n+1, 2) \rangle,$$

and the $(b+2)$ -th face is generated by $t_{b,a+1} \circ \delta^{b+2}$, which looks like

$$\langle (0, 0), \dots, (a+1, 0), (a+1, 1), \dots, (b, 1), (\widehat{b, 2}), (b+1, 2), \dots, (n+1, 2) \rangle.$$

In particular, $t_{b,a+1} \circ \delta^{a+1}$, $t_{b,a+1} \circ \delta^{a+2}$, $t_{b,a+1} \circ \delta^{b+1}$, and $t_{b,a+1} \circ \delta^{b+2}$ are surjective in the first component. By the same “visual” argument, it is clear that any other $t_{b,a+1} \circ \delta^i$ is not surjective in the first component.

- $\boxrightarrow_{\text{JM}}$ The $(a+1)$ -th face of $\tau_{b,a+1}$ is the $(a+1)$ -th face of $\tau_{b,a} \subset T_{b,a}$ (Lemma 6.2.6).
- $\boxrightarrow_{\text{JM}}$ The $(b+1)$ -th face of $\tau_{b,a+1}$ is $\sigma_{b-1,a+1} \subset T_{b,a}$ (Lemma 6.2.7).
- $\boxrightarrow_{\text{JM}} \boxrightarrow_{\text{JM}}$ If $b < n+1$, then the $(b+2)$ -th face of $\tau_{b,a+1}$ is $\sigma_{b,a+1} \subset T_{b,a}$ (Lemma 6.2.7). Otherwise, $b = n+1$ and the $(n+3)$ -th face of $\tau_{n+1,a+1}$ is generated by $t_{n+1,a+1} \circ \delta^{n+3}$, which looks like

$$\langle (0, 0), \dots, (a+1, 0), (a+1, 1), \dots, (n+1, 1), (\widehat{n+1, 2}) \rangle.$$

In this case, the $(n + 3)$ -th face is contained in $\Delta^{n+1} \times \Lambda_1^2 \subset T_{n+1,a}$, because 2 is not in the image of the second component of $t_{n+1,a+1} \circ \delta^{n+3}$.

It follows that $t_{b,a+1} (\Lambda_{a+2}^{n+3})$ is contained in $T_{b,a} \hookrightarrow_{\text{JM}}$. The rest of the morphisms in the diagram are the obvious inclusions, so the diagram commutes. To show that the diagram is a pushout square, it suffices (Theorem 3.2.2) to show that $T_{b+1,a} = T_{b,a} \cup \tau_{b,a+1}$ (which is true by construction) and that $t_{b,a+1} (\Lambda_{a+2}^{n+3}) = T_{b,a} \cap \tau_{b,a+1}$. Applying Corollary 3.2.14 with the monomorphism $t_{b,a+1}$ (Theorem 6.2.4), it suffices (by the same argument given in Lemma 6.2.8) to show that

$$t_{b,a+1} (\Delta^{[n+3] \setminus a+2}) \not\subseteq ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial \Delta^{n+1} \times \Delta^2)) \cup \left(\bigcup_{(y,x) \leq (n,n)} \sigma_{y,x} \right) \cup \left(\bigcup_{(y,x) \leq (b,a)} \tau_{y,x} \right). \hookrightarrow_{\text{JM}}$$

We split this into the following observations about $t_{b,a+1} (\Delta^{[n+3] \setminus a+2})$, the $(a + 2)$ -th face of $\tau_{b,a+1}$:

- $\hookrightarrow_{\text{JM}} t_{b,a+1} (\Delta^{[n+3] \setminus a+2}) \not\subseteq \Delta^{n+1} \times \Lambda_1^2$, because the image of the second component of $t_{b,a+1} \circ \delta^{a+2}$ contains $\{0, 2\}$.
- $\hookrightarrow_{\text{JM}} t_{b,a+1} (\Delta^{[n+3] \setminus a+2}) \not\subseteq \partial \Delta^{n+1} \times \Delta^2$, as observed above.
- $\hookrightarrow_{\text{JM}} t_{b,a+1} (\Delta^{[n+3] \setminus a+2}) \not\subseteq \sigma_{y,x}$ for any $0 \leq x \leq y \leq n$:
 - if $y < b$ and
 - * if $a + 1 < b$, then $(b, 1)$ is in the image of $t_{b,a+1} \circ \delta^{a+2}$, but not in the image of $s_{y,x}$.
 - * if $a + 1 = b$, then $(b, 0)$ is in the image of $t_{b,b} \circ \delta^{b+1}$, but not in the image of $s_{y,x}$.
 - if $b \leq y$, then $(b, 2)$ is in the image of $t_{b,a+1} \circ \delta^{a+2}$, but not in the image of $s_{y,x}$.
- $\hookrightarrow_{\text{JM}} t_{b,a+1} (\Delta^{[n+3] \setminus a+2}) \not\subseteq \tau_{y,x}$ for any $(y, x) \leq (b, a)$:
 - if $y = b$, then $x \leq a$ and $(a + 1, 0)$ is in the image of $t_{b,a+1} \circ \delta^{a+2}$, but not in the image of $t_{b,x}$.
 - if $y < b$, then $x < b$ and

- * if $a + 1 < b$, then $(b, 1)$ is in the image of $t_{b,a+1} \circ \delta^{a+2}$, but not in the image of $t_{y,x}$.
- * if $a + 1 = b$, then $(b, 0)$ is in the image of $t_{b,b} \circ \delta^{b+1}$, but not in the image of $s_{y,x}$.

Therefore, $t_{b,a+1}(\Lambda_{a+2}^{n+3}) = T_{b,a} \cap \tau_{b,a+1}$ and the diagram is a pushout square. \square

Lemma 6.2.16. \square_{JM} For all $0 \leq b \leq n$, the following diagram is a pushout square:

$$\begin{array}{ccc} t_{b+1,0}(\Lambda_1^{n+3}) & \longrightarrow & T_{b,b} \\ \downarrow & & \downarrow \\ \tau_{b+1,0} & \longrightarrow & T_{b+1,0} \end{array}$$

Proof. First, we show that the diagram commutes. Note that $t_{b+1,0}(\Lambda_1^{n+3}) = \bigcup_{i \neq 1} t_{b+1,0}(\Delta^{[n+3] \setminus i})$ is the union of all the faces of $\tau_{b+1,0}$ except the first. To show that $t_{b+1,0}(\Lambda_1^{n+3})$ is contained in $T_{b,b}$ we make the following observations about the faces of $\tau_{b+1,0}$:

- \square_{JM} Every face $t_{b+1,0}(\Delta^{[n+3] \setminus i})$ of $\tau_{b+1,0}$ is contained in $\partial \Delta^{n+1} \times \Delta^2 \subset T_{b,b}$ except the 0-th, first, $(b+2)$ -th, and $(b+3)$ -th face. The 0-th face is generated by $t_{b+1,0} \circ \delta^0$, which looks like

$$\langle (\widehat{(0,0)}, (0,1), \dots, (b+1,1), (b+1,2), \dots, (n+1,2)) \rangle,$$

the first face is generated by $t_{b+1,0} \circ \delta^1$, which looks like

$$\langle (0,0), (\widehat{(0,1)}, (1,1), \dots, (b+1,1), (b+1,2), \dots, (n+1,2)) \rangle,$$

the $(b+2)$ -th face is generated by $t_{b+1,0} \circ \delta^{b+2}$, which looks like

$$\langle (0,0), (0,1), \dots, (b,1), (\widehat{(b+1,1)}, (b+1,2), \dots, (n+1,2)) \rangle,$$

and the $(b+3)$ -th face is generated by $t_{b+1,0} \circ \delta^{b+3}$, which looks like

$$\langle (0,0), (0,1), \dots, (b+1,1), (\widehat{(b+1,2)}, (b+2,2), \dots, (n+1,2)) \rangle.$$

In particular, $t_{b+1,0} \circ \delta^0$, $t_{b+1,0} \circ \delta^1$, $t_{b+1,0} \circ \delta^{b+2}$, and $t_{b+1,0} \circ \delta^{b+3}$ are surjective in the first component. By the same “visual” argument, it is clear that any other $t_{b+1,0} \circ \delta^i$ is not surjective in the first component.

- $\boxrightarrow_{\text{JM}}$ The 0-th face is contained in $\Delta^{n+1} \times \Lambda_1^2 \subset T_{b,b}$, because 0 is not in the image of the second component of $t_{b+1,0} \circ \delta^0$.
- $\boxrightarrow_{\text{JM}}$ The $(b+2)$ -th face of $\tau_{b+1,0}$ is $\sigma_{b,0} \subset T_{b,b}$ (Lemma 6.2.7).
- $\boxrightarrow_{\text{JM}} \boxrightarrow_{\text{JM}}$ If $b < n$, the $(b+3)$ -th face of $\tau_{b+1,0}$ is $\sigma_{b+1,0}$ (Lemma 6.2.7). Otherwise, $b = n$ and the $(n+3)$ -th face of $\tau_{n+1,0}$ is generated by $t_{n+1,0} \circ \delta^{n+3}$, which looks like

$$\langle (0,0), (0,1), \dots, (n+1,1), \widehat{(n+1,2)} \rangle.$$

In this case, the $(n+3)$ -th face of $\tau_{n+1,0}$ is contained in $\Delta^{n+1} \times \Lambda_1^2$ because the image of the second component of $t_{n+1,0} \circ \delta^{n+3}$ does not contain 2.

It follows that $t_{b+1,0}(\Lambda_1^{n+3})$ is contained in $T_{b,b} \boxrightarrow_{\text{JM}}$. The rest of the morphisms in the diagram are the obvious inclusions, so the diagram commutes. To show that the diagram is a pushout square, it suffices (Theorem 3.2.2) to show that $T_{b+1,0} = T_{b,b} \cup \tau_{b+1,0}$ (which is true by construction) and that $t_{b+1,0}(\Lambda_1^{n+3}) = T_{b,b} \cap \tau_{b+1,0}$. Applying Corollary 3.2.14 with the monomorphism $t_{b+1,0}$ (Theorem 6.2.4), it suffices (by the same argument given in Lemma 6.2.8) to show that

$$t_{b+1,0}(\Delta^{[n+3] \setminus 1}) \not\subseteq ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial \Delta^{n+1} \times \Delta^2)) \cup \left(\bigcup_{(y,x) \leq (n,n)} \sigma_{y,x} \right) \cup \left(\bigcup_{(y,x) \leq (b,b)} \tau_{y,x} \right). \boxrightarrow_{\text{JM}}$$

We split this into four observations about $t_{b+1,0}(\Delta^{[n+3] \setminus 1})$, the first face of $\tau_{b+1,0}$:

- $\boxrightarrow_{\text{JM}}$ $t_{b+1,0}(\Delta^{[n+3] \setminus 1}) \not\subseteq \Delta^{n+1} \times \Lambda_1^2$, because the second component of $t_{b+1,0} \circ \delta^1$ is surjective.
- $\boxrightarrow_{\text{JM}}$ $t_{b+1,0}(\Delta^{[n+3] \setminus 1}) \not\subseteq \partial \Delta^{n+1} \times \Delta^2$, as shown above.
- $\boxrightarrow_{\text{JM}}$ $t_{b+1,0}(\Delta^{[n+3] \setminus 1}) \not\subseteq \sigma_{y,x}$ for any $0 \leq x \leq y \leq n$:

- if $y < b + 1$, then $(b + 1, 1)$ is in the image of $t_{b+1,0} \circ \delta^1$, but not in the image of $s_{y,x}$.
- if $b + 1 \leq y$, then $(b + 1, 2)$ is in the image of $t_{b+1,0} \circ \delta^1$, but not in the image of $s_{y,x}$.
- $\boxrightarrow_{\text{JM}} t_{b+1,0} (\Delta^{[n+3] \setminus 1}) \not\subseteq \tau_{y,x}$ for any $(y, x) \leq (b, b)$, because $(b + 1, 0)$ is in the image of $t_{b+1,0} \circ \delta^1$, but not in the image of any such $t_{y,x}$.

Therefore, $t_{b+1,0} (\Lambda_1^{n+3}) = T_{b,b} \cap \tau_{b+1,0}$ and the diagram is a pushout square. \square

Conclusion

We can finally prove the main result of this section:

Theorem 6.2.17. $\boxrightarrow_{\text{JM}}$ *The pushout-product*

$$(\partial\Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2) : (\Delta^n \times \Lambda_1^2) \coprod_{\partial\Delta^n \times \Lambda_1^2} (\partial\Delta^n \times \Delta^2) \rightarrow \Delta^n \times \Delta^2$$

is inner anodyne for all $n \in \mathbb{N}$.

Proof. If $n = 0$, then we have an isomorphism of arrows

$$(\partial\Delta^0 \hookrightarrow \Delta^0) \square (\Lambda_1^2 \hookrightarrow \Delta^2) \cong (\Lambda_1^2 \hookrightarrow \Delta^2), \quad (4.3.2)$$

so $(\partial\Delta^0 \hookrightarrow \Delta^0) \square (\Lambda_1^2 \hookrightarrow \Delta^2)$ is inner anodyne. Otherwise, we consider $(\partial\Delta^{n+1} \hookrightarrow \Delta^{n+1}) \square (\Lambda_1^2 \hookrightarrow \Delta^2)$. Combining Lemma 6.2.8, Lemma 6.2.10, Lemma 6.2.13, and Lemma 6.2.14, the inclusion

$$(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \hookrightarrow T_{n+1, n+1}$$

is inner anodyne $\boxrightarrow_{\text{JM}}$. We have isomorphisms

$$(\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \cong (\Delta^n \times \Lambda_1^2) \coprod_{\partial\Delta^n \times \Lambda_1^2} (\partial\Delta^n \times \Delta^2), \quad (3.2.2)$$

and

$$T_{n+1,n+1} \cong \Delta^{n+1} \times \Delta^2, \quad (3.2.5, 6.2.3)$$

from which an isomorphism of arrows follows

$$(\partial\Delta^{n+1} \hookrightarrow \Delta^{n+1}) \square (\Lambda_1^2 \hookrightarrow \Delta^2) \cong ((\Delta^{n+1} \times \Lambda_1^2) \cup (\partial\Delta^{n+1} \times \Delta^2) \hookrightarrow \Delta^n \times \Delta^2).$$

Therefore, $(\partial\Delta^n \hookrightarrow \Delta^n) \square (\Lambda_1^2 \hookrightarrow \Delta^2)$ is inner anodyne for all $n \in \mathbb{N}$. \square

Remark 6.2.18. After this project was completed, Joël Riou formalized a different (more general) statement of Theorem 6.2.17 [↗](#), following an approach derived by Sean Moss in [19, Lem. 17].

Chapter 7

Conclusion

In this chapter, we apply the theory developed in the preceding chapters to prove Theorem 7.1.1. Throughout this thesis, each concept and construction has been introduced with this goal in mind, providing the necessary motivation at every stage. Following the proof of Theorem 7.1.1, we present immediate consequences and outline possible directions for further work.

7.1 Functor Quasi-Categories are Quasi-Categories

Recall Definitions 3.3.2 and 3.4.4. In light of the language and theory established in preceding chapters, we prove the following theorem:

Theorem 7.1.1. \square_{JM} *If S is a simplicial set and X is a quasi-category, then $[S, X]$ is a quasi-category.*

Proof. We must show that every lifting problem of the following form has a solution:

$$\begin{array}{ccc} \Lambda_i^n & \longrightarrow & [S, X] \\ \downarrow & \nearrow \text{dashed} & \downarrow \\ \Delta^n & \longrightarrow & \Delta^0 \end{array}$$

Here Λ_i^n is an inner horn. X is a quasi-category, so every lifting problem of

the following form has a solution:

$$\begin{array}{ccc} \Lambda_i^n & \longrightarrow & X \\ \downarrow & \nearrow & \downarrow \\ \Delta^n & \longrightarrow & \Delta^0 \end{array}$$

Equivalently, every inner horn inclusion belongs to $\text{lp}(X \rightarrow \Delta^0)$. It follows that $\text{lp}(X \rightarrow \Delta^0)$ contains all inner anodyne morphisms (Definition 4.2.19) and, therefore, the collection $\{(\partial\Delta^m \hookrightarrow \Delta^m) \square (\Lambda_1^2 \hookrightarrow \Delta^2)\}_{m \in \mathbb{N}}$ (Theorem 6.0.3). Thus, every lifting problem of the following form has a solution:

$$\begin{array}{ccc} (\Delta^m \times \Lambda_1^2) \amalg_{\partial\Delta^m \times \Lambda_1^2} (\partial\Delta^m \times \Delta^2) & \longrightarrow & X \\ \downarrow & \nearrow & \downarrow \\ \Delta^m \times \Delta^2 & \longrightarrow & \Delta^0 \end{array}$$

By Corollary 4.3.6, these are equivalent to the following lifting problems \boxtimes_{JM} :

$$\begin{array}{ccc} \partial\Delta^m & \longrightarrow & [\Delta^2, X] \\ \downarrow & \nearrow & \downarrow \\ \Delta^m & \longrightarrow & [\Lambda_1^2, X] \end{array}$$

Equivalently, every boundary inclusion belongs to $\text{lp}([\Delta^2, X] \rightarrow [\Lambda_1^2, X])$. It follows that $\text{lp}([\Delta^2, X] \rightarrow [\Lambda_1^2, X])$ contains all monomorphisms (Corollary 5.1.6) and, therefore, the collection $\{\partial\Delta^m \times S \hookrightarrow \Delta^m \times S\}_{m \in \mathbb{N}}$. Thus, every lifting problem of the following form has a solution:

$$\begin{array}{ccc} \partial\Delta^m \times S & \longrightarrow & [\Delta^2, X] \\ \downarrow & \nearrow & \downarrow \\ \Delta^m \times S & \longrightarrow & [\Lambda_1^2, X] \end{array}$$

By Corollary 4.3.7, these are equivalent to the following lifting problems:

$$\begin{array}{ccc} \partial\Delta^m & \longrightarrow & [S, [\Delta^2, X]] \\ \downarrow & \nearrow & \downarrow \\ \Delta^m & \longrightarrow & [S, [\Lambda_1^2, X]] \end{array}$$

By observing the isomorphisms \mathcal{C}_{JM}

$$[S, [\Delta^2, X]] \cong [\Delta^2, [S, X]] \quad \text{and} \quad [S, [\Lambda_1^2, X]] \cong [\Lambda_1^2, [S, X]],$$

these are equivalent (Remark 4.2.16) to the following lifting problems \mathcal{C}_{JM} :

$$\begin{array}{ccc} \partial\Delta^m & \longrightarrow & [\Delta^2, [S, X]] \\ \downarrow & \nearrow \text{dashed} & \downarrow \\ \Delta^m & \longrightarrow & [\Lambda_1^2, [S, X]] \end{array}$$

By Corollary 4.3.6, these are equivalent to the following lifting problems:

$$\begin{array}{ccc} (\Delta^m \times \Lambda_1^2) \amalg_{\partial\Delta^m \times \Lambda_1^2} (\partial\Delta^m \times \Delta^2) & \longrightarrow & [S, X] \\ \downarrow & \nearrow \text{dashed} & \downarrow \\ \Delta^m \times \Delta^2 & \longrightarrow & \Delta^0 \end{array}$$

Equivalently, $\text{lp}([S, X] \rightarrow \Delta^0)$ contains the collection $\{(\partial\Delta^m \hookrightarrow \Delta^m) \square (\Lambda_1^2 \hookrightarrow \Delta^2)\}_{m \in \mathbb{N}}$. It follows that $\text{lp}([S, X] \rightarrow \Delta^0)$ contains all inner anodyne morphisms (Theorem 6.0.3) and, therefore, all inner horn inclusions (Definition 4.2.19). Thus, every lifting problem of following form admits a solution:

$$\begin{array}{ccc} \Lambda_i^n & \longrightarrow & [S, X] \\ \downarrow & \nearrow \text{dashed} & \downarrow \\ \Delta^n & \longrightarrow & \Delta^0 \end{array}$$

Here Λ_i^n is an inner horn. We conclude that $[S, X]$ is a quasi-category. \square

We have the following corollary:

Corollary 7.1.2. $\mathcal{C}_{\text{JM}} \mathbf{QCat}$, the category of quasi-categories, is cartesian closed.

Proof. Because \mathbf{QCat} is a full subcategory of \mathbf{SSet} , it suffices to check that \mathbf{QCat} is compatible with the cartesian closed structure on \mathbf{SSet} \mathcal{C}_{JM} \mathcal{C}_{JM} . In particular, it suffices to prove that

- (a) \mathcal{C}_{JM} if X and Y are quasi-categories, then $X \times Y$ is a quasi-category,

(b) $\mathcal{C}_{\text{JM}} \Delta^0$ (the monoidal unit) is a quasi-category, and

(c) \mathcal{C}_{JM} if X and Y are quasi-categories, then $[X, Y]$ is a quasi-category.

(a) is true because inner horn lifts for $X \times Y$ are induced by those for X and Y , and (b) is true because Δ^0 is terminal and hence trivially satisfies the inner horn lifting condition. (c) is true by Theorem 7.1.1. \square

7.2 Further Work

The immediate next step is to contribute the results of this thesis to Mathlib. This involves identifying results that are both reusable and sufficiently general, and generalizing them where necessary. For example:

- Many constructions from Chapter 4, such as pushout-products, pullback-powers, and the adjunction of lifting properties, must be generalized to fit Mathlib’s notion of parameterized adjunctions \mathcal{C}_{JM} .
- Theorem 6.2.2 should be generalized to apply to more general products of standard simplices. Furthermore, the bespoke proof of Theorem 6.2.17 should be examined for results that can be extracted and reused.

Once related code is organized and modularized, pull requests can be made and subjected to the careful Mathlib review process, piece by piece.

Various unrelated results were formalized during the course of this project, and should also make it into Mathlib. For example:

- For simplicial sets, miscellaneous results such as [15, Tag 050J, Tag 01BJ, Tag 006Z, Tag 00J8, Tag 01BT] are fully or nearly formalized.
- For simplicial subsets, certain order embeddings and order isomorphisms are of general utility \mathcal{C}_{JM} \mathcal{C}_{JM} .

Beyond Mathlib, another goal is to contribute to the ongoing ∞ -cosmos formalization project [23]. Theorem 7.1.1 is a major step toward formalizing the ∞ -cosmos of quasi-categories [24, Prop. 1.2.10], which will be the author’s next concrete formalization goal.

Bibliography

- [1] Anne Baanen, *Formalizing fundamental algebraic number theory*, PhD-Thesis - Research and graduation internal, 2024.
- [2] J. M. Boardman and R. M. Vogt, *Homotopy invariant algebraic structures on topological spaces*, Lecture Notes in Mathematics, Vol. 347, Springer-Verlag, Berlin-New York, 1973. MR0420609
- [3] Kevin Buzzard, *Fermat's last theorem — how it's going* (2024). Accessed: October 2025, <https://xenaproject.wordpress.com/2024/12/11/fermats-last-theorem-how-its-going/>.
- [4] Omar Antolín Camarena, *A whirlwind tour of the world of $(\infty, 1)$ -categories*, 2013.
- [5] Denis-Charles Cisinski, *Higher Categories and Homotopical Algebra*, 1st ed., Cambridge University Press, 2019.
- [6] Lean Prover Community, *Mathlib statistics*, 2025. Accessed: October 2025, https://leanprover-community.github.io/mathlib_stats.html.
- [7] The Lean Developers, *The lean language reference*, 2025. Accessed: November 2025, <https://lean-lang.org/doc/reference/4.25.0-rc2/>.
- [8] W.T. Gowers, Ben Green, Freddie Manners, and Terence Tao, *On a conjecture of Marston*, *Annals of Mathematics* **201** (2025), no. 2, 515–549.
- [9] Kevin Carlson ([https://math.stackexchange.com/users/31228/kevin carlson](https://math.stackexchange.com/users/31228/kevin%20carlson)), *composition and invertibility of 2-morphisms in a quasi-category*, 2020. Accessed: October 2025, <https://math.stackexchange.com/q/3827627>.
- [10] Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich, *Theorem proving in lean 4*, 2025. Accessed: October 2025, https://leanprover.github.io/theorem_proving_in_lean4/.
- [11] André Joyal, *Quasi-categories and Kan complexes*, *J. Pure Appl. Algebra* **175** (2002), no. 1-3, 207–222. Special volume celebrating the 70th birthday of Professor Max Kelly. MR1935979
- [12] ———, *Notes on quasi-categories*. (2008). Accessed: October 2025, <https://www.math.uchicago.edu/~may/IMA/Joyal.pdf>.
- [13] ———, *The theory of quasi-categories and its applications*. (2008). Accessed: October 2025, <https://mat.uab.cat/~kock/crm/hocat/advanced-course/Quadern45-2.pdf>.

- [14] Jacob Lurie, *Higher topos theory*, Annals of Mathematics Studies, vol. 170, Princeton University Press, Princeton, NJ, 2009. MR2522659
- [15] ———, *Kerodon*, 2025. <https://kerodon.net>.
- [16] Patrick Massot, *Why formalize mathematics?*, 2021. Accessed: October 2025, https://www.imo.universite-paris-saclay.fr/~patrick.massot/files/exposition/why_formalize.pdf.
- [17] The mathlib Community, *The lean mathematical library*, Proceedings of the 9th acm sigplan international conference on certified programs and proofs, 2020, pp. 367–381.
- [18] Bhavik Mehta, *Formalising modern research mathematics in real time* (2023). Accessed: October 2025, <https://xenaproject.wordpress.com/2023/11/04/formalising-modern-research-mathematics-in-real-time/>.
- [19] Sean Moss, *Another approach to the kan–quillen model structure*, Journal of Homotopy and Related Structures **15** (September 2019), no. 1, 143–165.
- [20] Leonardo de Moura and Sebastian Ullrich, *The lean 4 theorem prover and programming language*, Automated deduction – cade 28, 2021, pp. 625–635.
- [21] Charles Rezk, *Introduction to Quasicategories*, 2022. Accessed: October 2025, <https://rezk.web.illinois.edu/quasicats.pdf>.
- [22] Emily Riehl, *A LEISURELY INTRODUCTION TO SIMPLICIAL SETS* (2011). Accessed: October 2025, <https://math.jhu.edu/~eriehl/ssets.pdf>.
- [23] ———, *∞ -cosmos, a project to formalize ∞ -category theory in lean.*, 2025. Accessed: October 2025, <https://emilyriehl.github.io/infinity-cosmos/>.
- [24] Emily Riehl and Dominic Verity, *Elements of ∞ -Category Theory*, 1st ed., Cambridge University Press, 2022.
- [25] Peter Scholze, *Liquid tensor experiment*, Experimental Mathematics **31** (2022), no. 2, 349–354, available at <https://doi.org/10.1080/10586458.2021.1926016>.
- [26] Terence Tao, *Equational theories project, mapping out the relations between different equational theories of magmas*, 2024. Accessed: October 2025, https://teorth.github.io/equational_theories/.
- [27] The Univalent Foundations Program, *Homotopy type theory: Univalent foundations of mathematics*, Institute for Advanced Study, 2013. Accessed: October 2025, <https://homotopytypetheory.org/book>.