Towards Cubical Type Theory for Synthetic Stable ∞-Category Theory

Reuben Hillyard

Trinity Term 2025

Contents

Introduction

Spaces are an important object of study in mathematics. The class of spaces we will focus on is homotopy types: topological spaces considered up to weak homotopy equivalence. Homotopy types are significant to higher category theorists since the homotopy hypothesis identifies homotopy types with ∞-groupoids. (Deflorin 2019, Theorem 9.2)

Many models of ∞-groupoids involve complicated combinatorics (Lurie 2009, Subsection A.2.7 and Definition 1.1.2.1), and whatever model of homotopy types or ∞-groupoids we choose, we must take care that all our constructions respect equivalences appropriately.

We can escape the bureaucracy associated with having chosen a specific model by working synthetically in Homotopy Type Theory (HoTT) (Univalent Foundations Program 2013). That is, we axiomatise the properties that homotopy types enjoy rather than working from an explicit description. In HoTT, the equality types naturally endow every type with the structure of an ∞-groupoid, and every construction we can perform internally automatically preserves this structure (Riley, Finster and Licata 2021, p. 2). Whatever model of ∞-groupoids we choose, we establish once and for all that it is a model of HoTT, whereafter we can prove theorems in HoTT and they will apply to our model. Not only is it hopefully simpler to work in HoTT, but theorems proven in HoTT apply to all models at once; so we do not need to prove the same results over and over for different models.

An advantage of working in type theory is that terms have computational content; so, many equalities are judgmental and can be verified by a proof assistant, rather than requiring proof terms. In HoTT, however, we have the univalence axiom. Axioms in type theory obstruct computation: they are terms without rules specifying their computational behavior. Thus, while univalence is useful for synthetic homotopy theory, in HoTT it comes at the cost of losing desirable computational properties such as canonicity. In (Cohen et al. 2018), they introduce Cubical Type Theory (CuTT), where we regain canonicity (Huber 2019), and univalence becomes a theorem, provable from the new primitives we are given.

In algebraic topology, we study spaces by means of their algebraic invariants. In HoTT, we can define the homotopy groups and work with them synthetically. The definitions of homology and cohomology groups in terms of simplices, however, do

not work, but they can be defined by the spectra that represent them. (Riley, Finster and Licata 2021, p. 2) In (Riley, Finster and Licata 2021), they present an extension of HoTT to handle spectra. Their type theory has axioms, and these, like univalence, prevent computation. In this dissertation, we will work towards Stable Type Theory (StabTT), which seeks to give computational content to the type theory of (Riley, Finster and Licata 2021) as CuTT does for HoTT.

In Sections 1 and 2, we introduce HoTT and CuTT. In Section 3, we familiarise ourselves with working in CuTT by making definitions and proving a lemma that we need later on. In Section 4, we discuss stable ∞-categories. In Section 5, we work towards presenting StabTT. In Section 6, we adapt a method that (Riley, Finster and Licata 2021) use to obtain stability. In Section 7, we discuss what work is needed in the future.

1 Homotopy Type Theory

Homotopy Type Theory is Martin-Löf Type Theory with intensional equality types, univalence, and higher inductive types. (Univalent Foundations Program 2013)

1.1 Martin-Löf Type Theory

In Martin-Löf Type Theory (Martin-Löf 1975), there are terms, which denote pieces of data, every term has a type, which tells us what we can do with the data, and terms and types exist in contexts, which list the resources available for us to use.

The presentation of Martin-Löf Type Theory can be broken into two pieces. First, we give rules that explain how the judgments are related, and, second, we define our type-formers.

1.1.1 Judgments

In Martin-Löf Type Theory, we have the following judgments:

- Γ `ctx` – well-formed context;
- $\Gamma \vdash A$ `type` – well-formed type in a context;
- $\Gamma \vdash A \equiv B$ `type` – judgmentally equal types;
- $\Gamma \vdash a : A$ – well-typed term in a context;
- $\Gamma \vdash a \equiv b : A$ – judgmentally equal terms.

And we have inference rules to derive these judgments from each other.

Contexts are lists of typed variables. The types of the later variables may depend on the values of the earlier variables. We have two context-forming operations.

$$\textsc{Empty Context}\quad \frac{}{\cdot\ \texttt{ctx}} \qquad\qquad \textsc{Extended Context}\quad \frac{\Gamma \vdash A\ \texttt{type}}{\Gamma, a : A\ \texttt{ctx}}$$

We have one special term-forming operation – the variable rule – and the rest come from the universal properties of the types we will add to the system. The variable rule says that if we have a resource in our context, we may use it as data of the type we assumed it to have.

$$\textsc{Variable}\quad \frac{\Gamma, a : A, \Gamma'\ \texttt{ctx}}{\Gamma, a : A, \Gamma' \vdash a : A}$$

Judgmental equalities of types and terms arise in three ways:

- we have rules expressing that judgmental equality is reflexive, symmetric and transitive;
- for each type- or term-forming operation, we have a corresponding rule expressing that the operation preserves judgmental equality;
- types will typically have computation and uniqueness rules in the form of judgmental equalities.

1.1.2 Type-Formers

Each type-former represents some universal construction we want to perform in the category of types.

To present a type-former, we must give the following rules:

- formation – What data is the input to the type-former?
- introduction – How can we make values of the types?
- elimination – How can we use values of the types?
- computation – What happens when we use a value we just made?
- uniqueness – Some types have a uniqueness rule, which corresponds to the uniqueness part of their universal property. This rule tells us that taking a value apart and putting it back together again gets us back to where we started.

Often a type-former will have a mapping-in universal property, and we call them *negative*, or a mapping-out universal property, *positive*. Positive types tend to have eliminators that resemble the `match` construct from programming languages such as

Rust (Klabnik and Nichols 2022, Section 6.2), whereas negative types have eliminators that simply extract data, similar to accessing a field of an object. Some types, such as the unit type and dependent sum types, admit presentations as positive and negative types.

Example 1.1 (Negative Dependent Sum Types)**.** The elements of dependent sum – Σ – types are pairs where the type of the second component may depend on the value of the first. The data required to form a Σ type are the type A of the first component together with the type $B(a)$ of the second component depending on the value $a : A$ of the first.

$$\Sigma\text{-FORM}\quad \frac{\Gamma, a : A \vdash B\ \mathtt{type}}{\Gamma \vdash \sum_{a:A} B\ \mathtt{type}}$$

Dependent sum types have a mapping-in universal property: a map from Γ to $\sum_{a:A} B$ is given by a map s from Γ to A together with a map t from Γ to $B[s/a]$, where '$[s/a]$' denotes capture-avoiding substitution (Ker 2009). That is, a map is determined by the values it produces for each component. Then the introduction rule lets us assemble a term of type $\sum_{a:A} B$ from parts, the elimination rule lets us access the components, and the computation and uniqueness rules assert that these operations are mutually inverse.

$$\Sigma\text{-INTRO}\quad \frac{\Gamma, a : A \vdash B\ \mathtt{type} \qquad \Gamma \vdash s : A \qquad \Gamma \vdash t : B[s/a]}{\Gamma \vdash (s, t) : \sum_{a:A} B}$$

$$\Sigma\text{-ELIM}\quad \frac{\Gamma \vdash p : \sum_{a:A} B}{\Gamma \vdash \pi_1(p) : A \qquad \Gamma \vdash \pi_2(p) : B[\pi_1(p)/a]}$$

$$\Sigma\text{-COMP}\quad \frac{\Gamma, a : A \vdash B\ \mathtt{type} \qquad \Gamma \vdash s : A \qquad \Gamma \vdash t : B[s/a]}{\Gamma \vdash \pi_1((s, t)) \equiv s : A \qquad \Gamma \vdash \pi_2((s, t)) \equiv t : B[s/a]}$$

$$\Sigma\text{-UNIQ}\quad \frac{\Gamma \vdash p : \sum_{a:A} B}{\Gamma \vdash (\pi_1(p), \pi_2(p)) \equiv p : \sum_{a:A} B}$$

When B does not depend on $a : A$, we get the pair type $A \times B$.

Example 1.2 (Negative Unit Type)**.** The unit type represents the terminal object. It has a mapping-in universal property: there is a unique map into the terminal object. It takes no data to form the unit type $\mathbf{1}$. It takes no data to introduce a term $* : \mathbf{1}$. If we have $p : \mathbf{1}$, then all we can do with it is discard it. The computation rule says that if we start out with nothing, introduce $* : \mathbf{1}$ and discard it, then we get back to where we started, which is vacuous. The uniqueness rule says that if we have $p : \mathbf{1}$, discard it, and introduce $* : \mathbf{1}$, we are back where we started, that is, every term of type $\mathbf{1}$ is judgmentally equal to $* : \mathbf{1}$.

$$\mathbf{1}\text{-FORM}\quad \frac{\Gamma\ \mathtt{ctx}}{\Gamma \vdash \mathbf{1}\ \mathtt{type}} \qquad \mathbf{1}\text{-INTRO}\quad \frac{\Gamma\ \mathtt{ctx}}{\Gamma \vdash * : \mathbf{1}} \qquad \mathbf{1}\text{-ELIM}\quad \frac{\Gamma \vdash p : \mathbf{1}}{} \qquad \mathbf{1}\text{-COMP}\quad \frac{\Gamma\ \mathtt{ctx}}{} \qquad \mathbf{1}\text{-UNIQ}\quad \frac{\Gamma \vdash p : \mathbf{1}}{\Gamma \vdash * \equiv p : \mathbf{1}}$$

The **1**-ELIM and **1**-COMP rules are vacuous; so we can omit them.

Example 1.3 (Type of Booleans)**.** The type $\mathbb{B}$ of booleans is an inductive type with two constructors $\texttt{true}, \texttt{false} : \mathbb{B}$.

$$\mathbb{B}\text{-FORM}\ \frac{\Gamma\ \texttt{ctx}}{\Gamma \vdash \mathbb{B}\ \texttt{type}} \qquad \mathbb{B}\text{-INTRO}\ \frac{\Gamma\ \texttt{ctx}}{\Gamma \vdash \texttt{true} : \mathbb{B} \qquad \Gamma \vdash \texttt{false} : \mathbb{B}}$$

Inductive types have mapping-out universal properties and `match`-style eliminators. In the world of dependent types, we do not just eliminate into a type, but a type family – the *motive* – that depends on the scrutinee.

$$\mathbb{B}\text{-ELIM}\ \frac{\Gamma, b : \mathbb{B} \vdash Z\ \texttt{type} \qquad \Gamma \vdash s : Z[\texttt{true}/b] \qquad \Gamma \vdash t : Z[\texttt{false}/b] \qquad \Gamma \vdash p : \mathbb{B}}{\Gamma \vdash \texttt{if}_{b.Z}\ p\ \texttt{then}\ s\ \texttt{else}\ t : Z[p/b]}$$

$$\mathbb{B}\text{-COMP}\ \frac{\Gamma, b : \mathbb{B} \vdash Z\ \texttt{type} \qquad \Gamma \vdash s : Z[\texttt{true}/b] \qquad \Gamma \vdash t : Z[\texttt{false}/b]}{\begin{array}{c}\Gamma \vdash \texttt{if}_{b.Z}\ \texttt{true}\ \texttt{then}\ s\ \texttt{else}\ t \equiv s : Z[\texttt{true}/b] \\ \Gamma \vdash \texttt{if}_{b.Z}\ \texttt{false}\ \texttt{then}\ s\ \texttt{else}\ t \equiv t : Z[\texttt{false}/b]\end{array}}$$

We could add a uniqueness rule for $\mathbb{B}$, but we tend to omit uniqueness rules for inductive types since they make type-checking much harder and we will be able to prove propositional uniqueness rules. (Angiuli and Gratzer 2024)

Example 1.4 (Dependent Product Types)**.** The elements of dependent product – Π – types are functions where the type of the result may depend on the value of the argument. Dependent product types simultaneously generalise function types and universal quantification.

$$\Pi\text{-FORM}\ \frac{\Gamma, a : A \vdash B\ \texttt{type}}{\Gamma \vdash \prod_{a:A} B\ \texttt{type}} \qquad \Pi\text{-INTRO}\ \frac{\Gamma, a : A \vdash s : B}{\Gamma \vdash \lambda a.s : \prod_{a:A} B} \qquad \Pi\text{-ELIM}\ \frac{\Gamma \vdash s : \prod_{a:A} B \qquad \Gamma \vdash t : A}{\Gamma \vdash st : B[t/a]}$$

$$\Pi\text{-COMP}\ \frac{\Gamma, a : A \vdash s : B \qquad \Gamma \vdash t : A}{\Gamma \vdash (\lambda a.s)t \equiv s[t/a] : B[t/a]} \qquad \Pi\text{-UNIQ}\ \frac{\Gamma \vdash s : \prod_{a:A} B}{\Gamma \vdash \lambda a.sa \equiv s : \prod_{a:A} B}$$

When B does not depend on $a : A$, we get the function type $A \to B$.

Example 1.5 (Universe Types)**.** The elements of the universe types $\mathcal{U}_i$ are themselves types. For size reasons, we cannot have a type of all types (Angiuli and Gratzer 2024, Section 2.7); so we have a sequence of universe types, with each containing the previous.

1.2 Intensional Equality Types

Inductive types are freely generated by constructors. For example, the natural numbers are generated by `zero` and `succ`. We can then define a map out of an inductive type by pattern-matching on each of its constructors. With indexed inductive types, we define a family of types by mutual induction. Then, to define maps out, we must deal with all values of the indices together.

The textbook example of an indexed inductively defined family is vectors – lists of a given length. For a fixed type A, we have a family of types $\texttt{Vec}\ A\ n$ for $n : \mathbb{N}$. There are two constructors for vectors: `nil`, which makes an empty vector with type $\texttt{Vec}\ A\ \texttt{zero}$, and `cons`, which takes a $\texttt{Vec}\ A\ n$ and an A, and appends the element to the end of the list to make a $\texttt{Vec}\ A\ (\texttt{succ}\ n)$. To eliminate a vector, we must pattern-match on n and our value of type $\texttt{Vec}\ A\ n$ together.

Intensional equality types can also be seen as indexed inductive types. For a fixed type A, we have a family of types $a =_A b$ for $a, b : A$. We have a single constructor `refl`, and, for $a : A$, $\texttt{refl}_a$ has type $a =_A a$. The intuition for this is that the ways of being equal are freely generated by the rule that equality is reflexive.

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b\ \texttt{type}}\ =\text{-FORM} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \texttt{refl}_a : a =_A a}\ =\text{-INTRO}$$

To eliminate an equality, we need a type family C depending on our indices $a, b : A$ and a variable $p : a =_A b$, and we must pattern-match on a, b and our value of type $a =_A b$ together, with the single case that they are a, a and $\texttt{refl}_a$. Sometimes the elimination rule for equality types is called *the J rule*. (Warren 2008)

$$\frac{\begin{array}{c}\Gamma, a : A, b : A, p : a =_A b \vdash C\ \texttt{type} \\ \Gamma, a : A \vdash c : C[a/b, \texttt{refl}_a/p] \\ \Gamma \vdash u : s =_A t\end{array}}{\begin{array}{c}\Gamma \vdash \texttt{match}_{abp.C}\ (s, t, u)\ \{ \\ (a, a, \texttt{refl}_a) \mapsto c, \\ \} : C[s/a, t/b, u/p]\end{array}}\ =\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, a : A, b : A, p : a =_A b \vdash C\ \texttt{type} \\ \Gamma, a : A \vdash c : C[a/b, \texttt{refl}_a/p] \\ \Gamma \vdash s : A\end{array}}{\begin{array}{c}\Gamma \vdash \texttt{match}_{abp.C}\ (s, s, \texttt{refl}_s)\ \{ \\ (a, a, \texttt{refl}_a) \mapsto c, \\ \} \equiv c[s/a] : C[s/a, s/b, \texttt{refl}_s/p]\end{array}}\ =\text{-COMP}$$

As with $\mathbb{B}$, we will not have a uniqueness rule; in particular we do not have *uniqueness of identity proofs*, which would say, for $p, q : a =_A b$, it must be that p and q are equal. It may seem odd that there could be more than one way for two values to be equal, but this is just like two objects being isomorphic by more than one isomorphism.

The homotopical perspective on intensional equality is to imagine types as spaces, terms as points, and equalities as paths between them. Then unequal equality proofs correspond to non-homotopic paths, whose existence is not so surprising. The J rule allows us to concatenate and reverse equalities as coherently as we could for paths

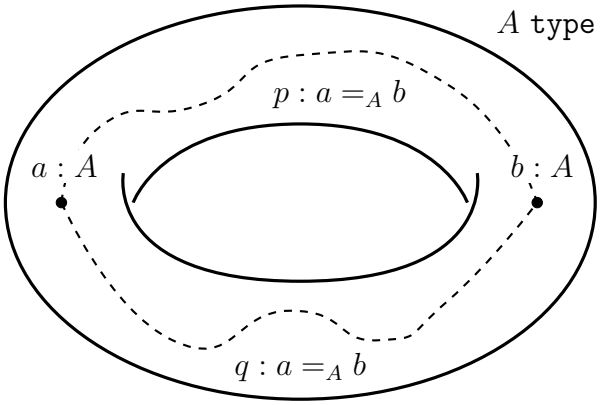


Figure 1: Unequal terms $p, q : a =_A b$

in a space; so every type has an ∞-groupoid structure (Berg and Garner 2011), and we can think of types as homotopy types.

1.3 Univalence

For most of our type-formers, we can relate equality in the formed type to equality in the input types. For example, two pairs are equal when their components are equal. Intensional type theory is not sufficient to do this for the universe type; so we will add an axiom. Then the question is: what do we want equality in $\mathcal{U}_i$ to be? In set theory, two sets are equal when they have all the same elements, but this doesn't reflect the way we do mathematics: we might say $\mathbb{N} \subseteq \mathbb{Q}$ even if the elements of $\mathbb{N}$ are not literally elements of $\mathbb{Q}$. In category theory, we know that the appropriate notion of 'sameness' for objects is isomorphism. We will add the univalence axiom, which asserts that equality in $\mathcal{U}_i$ is isomorphism of types, but in line with the homotopical perspective, we will call isomorphisms *equivalences*, as in 'homotopy equivalences'.

Defining what it means for a function to be an equivalence in HoTT is more subtle than defining bijections in set theory; we want to ensure that 'being an equivalence' is a property, and not additional structure. (Univalent Foundations Program 2013, Chapter 4) We take as our definition 'admitting a left- and right-inverse'. This leads to the definitions

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

$$\mathsf{isEquiv}(f) \equiv \left(\sum_{g:B\to A} f \circ g \sim \mathrm{id}_B\right) \times \left(\sum_{h:B\to A} h \circ f \sim \mathrm{id}_A\right)$$

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

Whenever A, B are equal types, they are equivalent; so we have a term $\mathsf{idtoeqv}$ that converts equalities to equivalences.

$$\begin{aligned}
&\mathsf{idtoeqv} : (A =_{\mathcal{U}_i} B) \to (A \simeq B)\\
&\mathsf{idtoeqv}\ p \equiv \mathtt{match}\ (A, B, p)\ \{\\
&\qquad\qquad (C, C, \mathtt{refl}_C) \mapsto (\mathrm{id}_C, ((\mathrm{id}_C, \lambda c.\mathtt{refl}_c), (\mathrm{id}_C, \lambda c.\mathtt{refl}_c))),\\
&\qquad \}
\end{aligned}$$

We want equalities to be the same as equivalences; so we add an axiom ua of type $\mathsf{isEquiv}(\mathsf{idtoeqv})$ asserting that $\mathsf{idtoeqv}$ is an equivalence. (Univalent Foundations Program 2013)

1.4 Higher Inductive Types

Having adopted the homotopical perspective, we would like a way to construct types with non-trivial higher homotopies. Higher inductive types generalise inductive types by allowing constructors to produce equalities in addition to points.

Example 1.6 (Suspension)**.** Fix a type A; then the suspension type ΣA – not to be confused with dependent sum – can be defined by

$$\begin{aligned}
&\mathtt{inductive}\ \Sigma A\ \mathtt{where}\ \{\\
&\quad \mathtt{N} : \Sigma A,\\
&\quad \mathtt{S} : \Sigma A,\\
&\quad \mathtt{merid} : A \to \mathtt{N} =_{\Sigma A} \mathtt{S},\\
&\}
\end{aligned}$$

To define a map out of ΣA, we must choose its values at $\mathtt{N}$ and $\mathtt{S}$, and, for $a : A$, we must choose an equality between those values. In HoTT, the case of the computation rule for $\mathtt{merid}$ is not a judgmental equality but an axiomatised propositional equality (Univalent Foundations Program 2013, Chapter 6), leading to stuck terms. We do not give ΣA a judgmental uniqueness rule, since it would not hold some of our intended models, and a propositional uniqueness rule is provable.

2 Cubical Type Theory

In Cubical Type Theory, we reify the paths that make up equality proofs with the use of a formal interval $\mathbb{I}$, which plays the role that the closed unit interval does in ordinary homotopy theory.

A naïve approach to this would make the interval an ordinary type. We would then want to write

$$\mathrm{Path}_A(a_0, a_1) \equiv \sum_{p:\mathbb{I}\to A} \big(p(\mathsf{i0}) =_A a_0\big) \times \big(p(\mathsf{i1}) =_A a_1\big)$$

but then we are using the equality type to define the path type which is supposed take the place of the equality type. A step closer is the ill-formed definition

$$\mathrm{Path}_A(a_0, a_1) \equiv \sum_{p:\mathbb{I}\to A} \big(p(\mathsf{i0}) \equiv a_0\big) \times \big(p(\mathsf{i1}) \equiv a_1\big)$$

where we've eliminated the equality type in favour of a non-existent judgmental equality type. (Riehl and Shulman 2023, Section 1)

Introducing the interval as a 'pre-type' and not a first-class type gives us the flexibility to axiomatise a path type with the universal property that we would expect the ill-formed definition to have. As a pre-type, the interval will behave as a type in many ways, but not all.

2.1 Dimension Layer

The first way that the interval differs from a first-class type is that it gets its own section of the context, given by the rules

$$\textsc{Empty dtx}\ \frac{}{\cdot\ \mathtt{dtx}} \qquad \textsc{Extended dtx}\ \frac{\Xi\ \mathtt{dtx}}{\Xi, x : \mathbb{I}\ \mathtt{dtx}}$$

In a dimension context, we can form dimension terms as follows.

$$\textsc{Dimension Variable}\ \frac{\Xi, x : \mathbb{I}, \Xi'\ \mathtt{dtx}}{\Xi, x : \mathbb{I}, \Xi' \vdash x : \mathbb{I}} \qquad \textsc{Dimension Begin}\ \frac{\Xi\ \mathtt{dtx}}{\Xi \vdash \mathsf{i0} : \mathbb{I}} \qquad \textsc{Dimension End}\ \frac{\Xi\ \mathtt{dtx}}{\Xi \vdash \mathsf{i1} : \mathbb{I}}$$

In some presentations of CuTT, such as (Angiuli, Brunerie et al. 2021), these are the only rules that create interval terms. These correspond to the morphisms in the cartesian cube category. We, like (Cohen et al. 2018), will have more interval terms – arising from the additional morphisms in the de Morgan cube category. These terms express that the interval is a de Morgan algebra.

$$\textsc{Dimension Conjunction}\ \frac{\Xi \vdash r : \mathbb{I} \qquad \Xi \vdash s : \mathbb{I}}{\Xi \vdash r \wedge s : \mathbb{I}} \qquad \textsc{Dimension Disjunction}\ \frac{\Xi \vdash r : \mathbb{I} \qquad \Xi \vdash s : \mathbb{I}}{\Xi \vdash r \vee s : \mathbb{I}} \qquad \textsc{Dimension Negation}\ \frac{\Xi \vdash r : \mathbb{I}}{\Xi \vdash \neg r : \mathbb{I}}$$

We will have the expected equations governing these as part of the next layer.

2.2 Formula Layer

Next, we have a section of the context consisting of formulas relating the interval terms. This section is what allows us to make sense of the judgmental equalities in our ill-formed definition of the path type.

For two interval terms $r, r' : \mathbb{I}$, we get a formula $r = r'$ asserting their equality, and formulas are closed under disjunction. This generates all the formulas we will use,

but to define Kan composition for `Glue` types, formulas also need to be closed under universal quantification. (Angiuli, Brunerie et al. 2021)

A formula context is simply a list of fomulas; we have a judgment $\Xi \vdash \Phi$ `ftx` that Φ is a well-formed formula context over Ξ. We have a proof system for a judgment $\Xi|\Phi \vdash \phi$ that ϕ is provable from Φ, including the usual rules for the connectives, and the axioms that $\mathbb{I}$ is a de Morgan algebra. If we can prove an equality formula, we can reflect it as a judgmental equality. This does not threaten computability since this logic is decidable.

2.3 Ordinary Layer

On top of these two layers, we have ordinary Martin-Löf Type Theory with two additional features. First, we can define terms by pattern-matching on disjunction formulas, and, second, we have Kan composition.

2.3.1 Disjunction

The rules for eliminating disjunction formulas are those we would expect from considering them as the pushout of the disjuncts against their conjunction. If we have two partially-defined terms s, t and they agree judgmentally when they are both defined, we can define a term on the disjunction.

$$\frac{\Xi|\Phi \vdash \alpha \vee \beta \qquad \Xi|\Phi, \alpha|\Gamma \vdash s : A \qquad \Xi|\Phi, \beta|\Gamma \vdash t : A \qquad \Xi|\Phi, \alpha, \beta|\Gamma \vdash s \equiv t : A}{\Xi|\Phi|\Gamma \vdash [\alpha \mapsto s, \beta \mapsto t] : A} \vee\text{-ELIM}$$

If Φ proves α then $[\alpha \mapsto s, \beta \mapsto t] \equiv s$, and symmetrically for β. Also, if u is defined on $\alpha \vee \beta$, then $[\alpha \mapsto u, \beta \mapsto u] \equiv u$.

We have similar rules expressing initiality of $\mathsf{i0} = \mathsf{i1}$: when $\mathsf{i0} = \mathsf{i1}$ holds, we get a term $\mathtt{abort} : A$, and $\mathtt{abort} \equiv a : A$ for any term $a : A$.

2.3.2 Kan Composition

In HoTT, types acquire ∞-groupoid structure from the J rule. In CuTT, paths are real maps out of $\mathbb{I}$, and we get ∞-groupoid structure from *Kan composition*, inspired by the model of HoTT in cubical sets from (Bezem, Coquand and Huber 2014).

To perform a Kan composition, we need a box missing its lid. That is, we have a term b at the base of the box, a partial term t at the sides of the box, and they agree judgmentally where they meet. Then we get a term filling the box.

In (Angiuli, Brunerie et al. 2021), we have diagonal Kan compositions, where we might not compose from the bottom of the box to the top, but from any diagonal to any other.

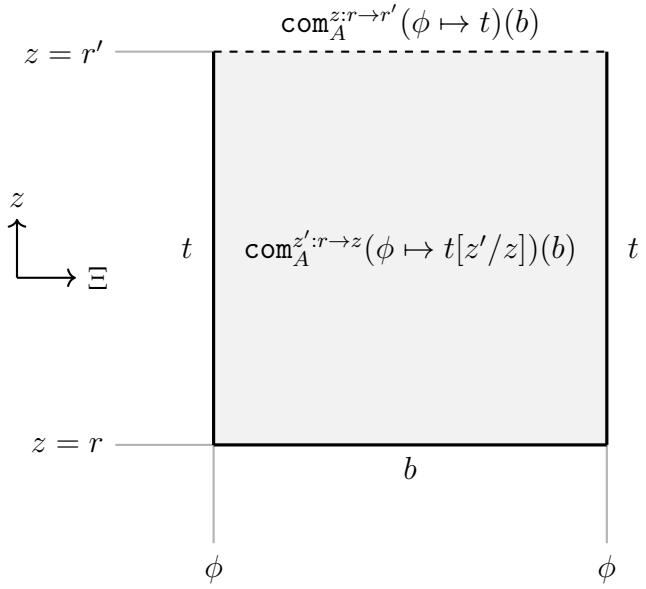


Figure 2: The data involved in a Kan composition

Kan Composition

$$\frac{\Xi \vdash r : \mathbb{I} \qquad \Xi \vdash r' : \mathbb{I} \qquad \Xi, z : \mathbb{I} | \Phi | \Gamma \vdash A\ \mathtt{type} \qquad \Xi, z : \mathbb{I} | \Phi, \phi | \Gamma \vdash t : A \qquad \Xi | \Phi | \Gamma \vdash b : A[r/z] \qquad \Xi | \Phi, \phi | \Gamma \vdash t[r/z] \equiv b : A[r/z]}{\Xi | \Phi | \Gamma \vdash \mathtt{com}_A^{z:r\to r'}(\phi \mapsto t)(b) : A[r'/z]}$$

This computes judgmentally to the partial term where it is defined, and to the base if we composed from an interval term to itself.

To maintain computational content, we need to know how to compute with Kan compositions. For negative types, this means defining Kan composition in terms of Kan composition for the inputs. For positive types, we add a constructor `vhcom` that produces formal Kan compositions, and a computation rule stating that `match` sends `vhcom`'s to Kan composition in the codomain. Another perspective is that Kan composition is defined by external induction on the structure of the type where the composition takes place.

Example 2.1 (Kan Rule for Negative Dependent Sum)**.** To compose in $\sum_{a:A} B$, we compose in A and B.

$$\frac{\begin{array}{c}\Xi \vdash r : \mathbb{I} \qquad \Xi \vdash r' : \mathbb{I} \qquad \Xi, z : \mathbb{I} | \Phi | \Gamma, a : A \vdash B \ \mathtt{type} \qquad \Xi, z : \mathbb{I} | \Phi, \phi | \Gamma \vdash t : \sum_{a:A} B \\ \Xi | \Phi | \Gamma \vdash b : \sum_{a:A[r/z]} B[r/z] \qquad \Xi | \Phi, \phi | \Gamma \vdash t[r/z] \equiv b : \sum_{a:A[r/z]} B[r/z]\end{array}}{\begin{array}{l}\Xi | \Phi | \Gamma \vdash \mathtt{com}^{z:r \to r'}_{\sum_{a:A} B}(\phi \mapsto t)(b) \equiv \big(\mathtt{com}^{z:r \to r'}_{A}(\phi \mapsto \pi_1(t))(\pi_1(b)), \\ \qquad \mathtt{com}^{z':r \to r'}_{B[z'/z,\ \mathtt{com}^{z:r \to z'}_{A}(\phi \mapsto \pi_1(t))(\pi_1(b))/a]}(\\ \qquad\quad \phi \mapsto \pi_2(t[z'/z]), \\ \qquad)(\pi_2(b))\big) : \sum_{a:A[r'/z]} B[r'/z]\end{array}}$$

2.4 Path Types

Now we can define the path type. (Angiuli, Brunerie et al. 2021, Section 2.10)

$$\frac{\Xi, x : \mathbb{I} | \Phi | \Gamma \vdash A \ \mathtt{type} \qquad \Xi | \Phi | \Gamma \vdash a_0 : A[\mathsf{i0}/x] \qquad \Xi | \Phi | \Gamma \vdash a_1 : A[\mathsf{i1}/x]}{\Xi | \Phi | \Gamma \vdash \mathrm{Path}_{x.A}(a_0, a_1) \ \mathtt{type}}$$

$$\frac{\Xi, x : \mathbb{I} | \Phi | \Gamma \vdash a : A}{\Xi | \Phi | \Gamma \vdash (x \mapsto a) : \mathrm{Path}_{x.A}(a[\mathsf{i0}/x], a[\mathsf{i1}/x])}$$

$$\frac{\begin{array}{c}\Xi | \Phi | \Gamma \vdash p : \mathrm{Path}_{x.A}(a_0, a_1) \\ \Xi \vdash r : \mathbb{I}\end{array}}{\begin{array}{l}\Xi | \Phi | \Gamma \vdash p\ r : A[r/x] \\ \Xi | \Phi | \Gamma \vdash p\ \mathsf{i0} \equiv a_0 : A[\mathsf{i0}/x] \\ \Xi | \Phi | \Gamma \vdash p\ \mathsf{i1} \equiv a_1 : A[\mathsf{i1}/x]\end{array}}$$

$$\frac{\Xi, x : \mathbb{I} | \Phi | \Gamma \vdash a : A \qquad \Xi \vdash r : \mathbb{I}}{\Xi | \Phi | \Gamma \vdash (x \mapsto a)\ r \equiv a[r/x] : A[r/x]}$$

$$\frac{\Xi | \Phi | \Gamma \vdash p : \mathrm{Path}_{x.A}(a_0, a_1)}{\Xi | \Phi | \Gamma \vdash (x \mapsto p\ x) \equiv p : \mathrm{Path}_{x.A}(a_0, a_1)}$$

Path-KAN

$$\frac{\begin{array}{c}\Xi \vdash r : \mathbb{I} \qquad \Xi \vdash r' : \mathbb{I} \qquad \Xi, z : \mathbb{I}, x : \mathbb{I} | \Phi | \Gamma \vdash A \ \mathtt{type} \\ \Xi, z : \mathbb{I} | \Phi | \Gamma \vdash a_0 : A[\mathsf{i0}/x] \qquad \Xi, z : \mathbb{I} | \Phi | \Gamma \vdash a_1 : A[\mathsf{i1}/x] \\ \Xi, z : \mathbb{I} | \Phi, \phi | \Gamma \vdash t : \mathrm{Path}_{x.A}(a_0, a_1) \qquad \Xi | \Phi | \Gamma \vdash b : \mathrm{Path}_{x.A[r/z]}(a_0[r/z], a_1[r/z]) \\ \Xi | \Phi, \phi | \Gamma \vdash t[r/z] \equiv b : \mathrm{Path}_{x.A[r/z]}(a_0[r/z], a_1[r/z])\end{array}}{\begin{array}{l}\Xi | \Phi | \Gamma \vdash \mathtt{com}^{z:r \to r'}_{\mathrm{Path}_{x.A}(a_0,a_1)}(\phi \mapsto t)(b) \equiv \big(x \mapsto \mathtt{com}^{z:r \to r'}_{A}(\\ \qquad \phi \mapsto t\ x, \\ \qquad x = \mathsf{i0} \mapsto a_0, \\ \qquad x = \mathsf{i1} \mapsto a_1, \\ \qquad)(b\ x)\big) : \mathrm{Path}_{x.A[r'/z]}(a_0[r'/z], a_1[r'/z])\end{array}}$$

2.5 Higher Inductive Types

In HoTT, higher inductive types have constructors producing equalities, but, in CuTT, they simply have constructors with interval parameters. This lets us give judgmental computation rules.

Example 2.2 (Suspension)**.** Now, we define suspension by

$$\begin{aligned}&\texttt{inductive}\ \Sigma A\ \texttt{where}\ \{\\ &\quad \texttt{N} : \Sigma A,\\ &\quad \texttt{S} : \Sigma A,\\ &\quad \texttt{merid}^r : A \to \Sigma A, \quad \texttt{merid}^{\mathsf{i0}}(a) \equiv \texttt{N}, \quad \texttt{merid}^{\mathsf{i1}}(a) \equiv \texttt{S},\\ &\}\end{aligned}$$

With corresponding formation and introduction rules

$$\Sigma\text{-FORM}\ \frac{\Xi|\Phi|\Gamma \vdash A\ \texttt{type}}{\Xi|\Phi|\Gamma \vdash \Sigma A\ \texttt{type}} \qquad \Sigma\text{-INTRO}_1\ \frac{\Xi|\Phi|\Gamma \vdash A\ \texttt{type}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \texttt{N} : \Sigma A\\ \Xi|\Phi|\Gamma \vdash \texttt{S} : \Sigma A\end{array}}$$

$$\Sigma\text{-INTRO}_2\ \frac{\Xi|\Phi|\Gamma \vdash a : A \qquad \Xi \vdash r : \mathbb{I}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \texttt{merid}^r(a) : \Sigma A\\ \Xi|\Phi|\Gamma \vdash \texttt{merid}^{\mathsf{i0}}(a) \equiv \texttt{N} : \Sigma A\\ \Xi|\Phi|\Gamma \vdash \texttt{merid}^{\mathsf{i1}}(a) \equiv \texttt{S} : \Sigma A\end{array}}$$

Again, we define maps out by cases.

$$\Sigma\text{-ELIM}\ \frac{\begin{array}{c}\Xi|\Phi|\Gamma, p : \Sigma A \vdash C\ \texttt{type}\\ \Xi|\Phi|\Gamma \vdash s : C[\texttt{N}/p] \qquad \Xi|\Phi|\Gamma \vdash t : C[\texttt{S}/p]\\ \Xi, x : \mathbb{I}|\Phi|\Gamma, a : A \vdash u : C[\texttt{merid}^x(a)/p]\\ \Xi|\Phi|\Gamma, a : A \vdash u[\mathsf{i0}/x] \equiv s : C[\texttt{N}/p] \qquad \Xi|\Phi|\Gamma, a : A \vdash u[\mathsf{i1}/x] \equiv t : C[\texttt{S}/p]\\ \Xi|\Phi|\Gamma \vdash q : \Sigma A\end{array}}{\Xi|\Phi|\Gamma \vdash \texttt{match}_{p.C}\ q\ \{\texttt{N} \mapsto s,\ \texttt{S} \mapsto t,\ \texttt{merid}^x(a) \mapsto u\} : C[q/p]}$$

Abbreviate $M(q) \equiv \texttt{match}_{p.C}\ q\ \{\texttt{N} \mapsto s,\ \texttt{S} \mapsto t,\ \texttt{merid}^x(a) \mapsto u\}$; then we have judgmental computation rules $M(\texttt{N}) \equiv s$, $M(\texttt{S}) \equiv t$, and $M(\texttt{merid}^r(e)) \equiv u[r/x, e/a]$. As in HoTT, the judgmental uniqueness rule would be too strong; so we omit it.

Kan composition can be decomposed into *homogeneous Kan composition* – when the type family is independent of the interval variable – and *weak coercion* – when the partial term is defined nowhere. (Angiuli, Brunerie et al. 2021, Section 2.7) To define Kan composition for positive types, such as ΣA, we add a constructor `vhcom` that computes formal homogeneous Kan compositions, we define weak coercion by external induction, and these assemble into a Kan rule. (Angiuli, Brunerie et al. 2021, Section 2.15)

We need a case of the computation rule for `vhcom`:

$$M(\texttt{vhcom}_{\Sigma A}^{z:r\to r'}(\phi \mapsto t)(b)) \equiv \texttt{com}^{z':r\to r'}_{C[\texttt{vhcom}^{z:r\to z'}_{\Sigma A}(\phi\mapsto t)(b)/p]}(\phi \mapsto M(t[z'/z]))(M(b))$$

2.6 Univalence

We achieve univalence by a new type-former – `Glue` – that reifies the path whose existence the standard univalence axiom demands. The Kan rule for `Glue` gives computational content to univalence. We also use `Glue` types to define Kan composition in the universe. (Angiuli, Brunerie et al. 2021, Section 2.12)

3 Definitions

Definition 3.1 (Morphism)**.** A morphism, or map, $f : A \to B$ is a method for producing terms of type B from terms of type A. If we have a term $a : A \vdash s : B$, then we get a map $a.s : A \to B$, and every morphism arises in this way uniquely up to change of dummy variable. If we have $f : A \to B$, and $t : A$, then we get a term $f(t) : B$. We define this by $(a.s)(t) \equiv s[t/a]$.

Definition 3.2 (Identity and Composition)**.** Given a type A, we have $a : A \vdash a : A$; so we get a map $\mathrm{id}_A \equiv (a.a) : A \to A$. Given maps $f : A \to B$ and $g : B \to C$, we get a map $g \circ f \equiv (a.g(f(a))) : A \to C$.

Definition 3.3 (Homotopy)**.** A homotopy is a map that depends on an interval variable.

3.1 Equivalences

Definition 3.4 (Equivalence)**.** A map $f : A \to B$ is an equivalence when it is equipped with the following data:

- maps $g, h : B \to A$;
- homotopies $M^r : B \to B$, $N^r : A \to A$, such that

$$M^{\mathsf{i0}} \equiv \mathrm{id}_B \qquad M^{\mathsf{i1}} \equiv f \circ g \qquad N^{\mathsf{i0}} \equiv \mathrm{id}_A \qquad N^{\mathsf{i1}} \equiv h \circ f$$

Definition 3.5 (Half-adjoint Equivalence)**.** A map $f : A \to B$ is a half-adjoint equivalence when it is equipped with the following data:

- a map $g : B \to A$;
- homotopies $M^r : B \to B$, $N^r : A \to A$, such that

$$M^{\mathsf{i0}} \equiv \mathrm{id}_B \qquad M^{\mathsf{i1}} \equiv f \circ g \qquad N^{\mathsf{i0}} \equiv \mathrm{id}_A \qquad N^{\mathsf{i1}} \equiv g \circ f$$

- a homotopy between homotopies $\tau^{r,s} : A \to B$, such that

$$\tau^{\mathsf{i0},s} \equiv f \qquad \tau^{\mathsf{i1},s} \equiv f \circ g \circ f \qquad \tau^{r,\mathsf{i0}} \equiv f \circ N^r \qquad \tau^{r,\mathsf{i1}} \equiv M^r \circ f$$

Lemma 3.6 (Equivalence Promotion)**.** *For a map $f : A \to B$, if it is an equivalence, then it is a half-adjoint equivalence.*

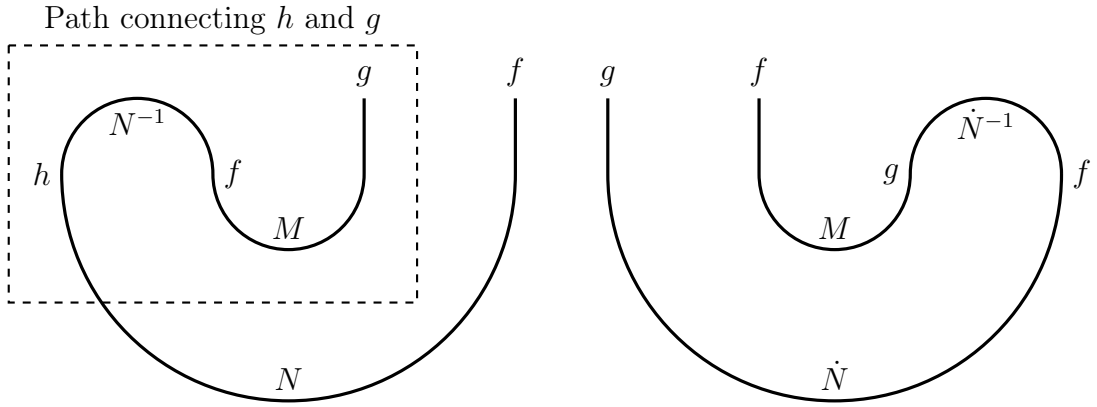
Proof. We are given data exhibiting f as an equivalence, and must construct data exhibiting it as a half-adjoint equivalence. Our first step is to eliminate h in favour of g. To do this, we use the path we have that connects h and g, and concatenate it with N.

$$\dot{N}^r(a) \equiv \mathtt{com}_A^{z:\mathsf{i1}\to\mathsf{i0}}(\quad r = \mathsf{i0} \mapsto N^z(a),\quad r = \mathsf{i1} \mapsto N^z(g(f(a))),\quad)(h(M^r(f(a))))$$

$$\dot{N}^{\mathsf{i0}} \equiv \mathrm{id}_A \qquad \dot{N}^{\mathsf{i1}} \equiv g \circ f$$

Next, we adjust $\dot{N}$ to relate to M as τ requires.

$$\ddot{N}^r(a) \equiv \mathtt{com}_A^{z:\mathsf{i1}\to\mathsf{i0}}(\quad r = \mathsf{i0} \mapsto \dot{N}^z(a),\quad r = \mathsf{i1} \mapsto g(f(\dot{N}^z(a))),\quad)(g(M^r(f(a))))$$



(a) Definition of $\dot{N}$ (b) Definition of $\ddot{N}$

Figure 3: String diagrams for intuition

Finally, we define τ by

$$
\begin{aligned}
\tau^{r,s}(a) \equiv \mathtt{com}_B^{z':\mathsf{i}1\to\mathsf{i}0}(& \\
& r = \mathsf{i}0 \mapsto f(\dot{N}^{z'}(a)), \\
& r = \mathsf{i}1 \mapsto f(g(f(\dot{N}^{z'}(a)))), \\
& s = \mathsf{i}0 \mapsto f(\mathtt{com}_A^{z:\mathsf{i}1\to z'}(\\
& \qquad\qquad r = \mathsf{i}0 \mapsto \dot{N}^{z}(a), \\
& \qquad\qquad r = \mathsf{i}1 \mapsto g(f(\dot{N}^{z}(a))), \\
& \qquad\qquad)(g(M^r(f(a))))), \\
& s = \mathsf{i}1 \mapsto M^r(f(\dot{N}^{z'}(a))), \\
&)(\mathtt{com}_B^{y:\mathsf{i}0\to\mathsf{i}1}(\\
& \quad r = \mathsf{i}0 \mapsto M^y(f(a)), \\
& \quad r = \mathsf{i}1 \mapsto f(g(f(g(f(a))))), \\
& \quad s = \mathsf{i}0 \mapsto M^{y\vee r}(M^r(f(a))), \\
& \quad s = \mathsf{i}1 \mapsto M^r(M^{y\vee r}(f(a))), \\
&)(M^r(M^r(f(a))))
\end{aligned}
$$

□

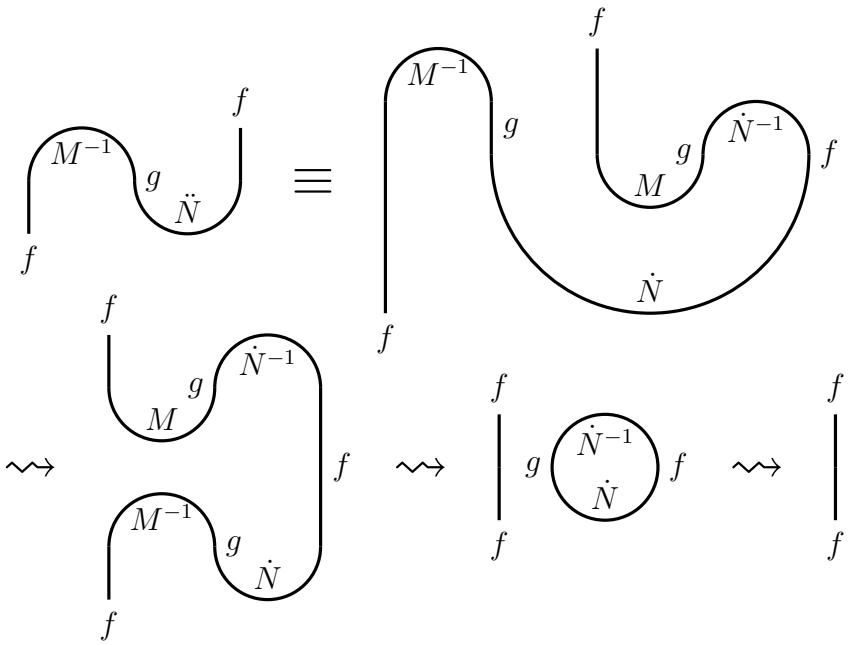
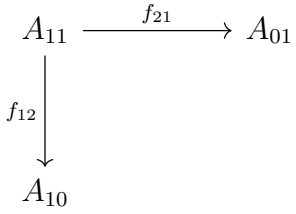


Figure 4: Intuition for τ

3.2 Diagrams

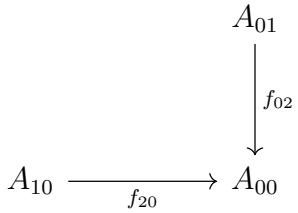
Definition 3.7 (Span)**.** A span is a pair of morphisms with a common domain.



The data of a span consists of:

- objects A_{11}, A_{01}, A_{10};
- morphisms $f_{21} : A_{11} \to A_{01}$, $f_{12} : A_{11} \to A_{10}$.

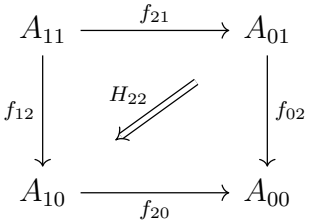
Definition 3.8 (Cospan)**.** Dually, a cospan is a pair of morphisms with a common codomain.



The data of a cospan consists of:

- objects A_{01}, A_{10}, A_{00};
- morphisms $f_{02} : A_{01} \to A_{00}$, $f_{20} : A_{10} \to A_{00}$.

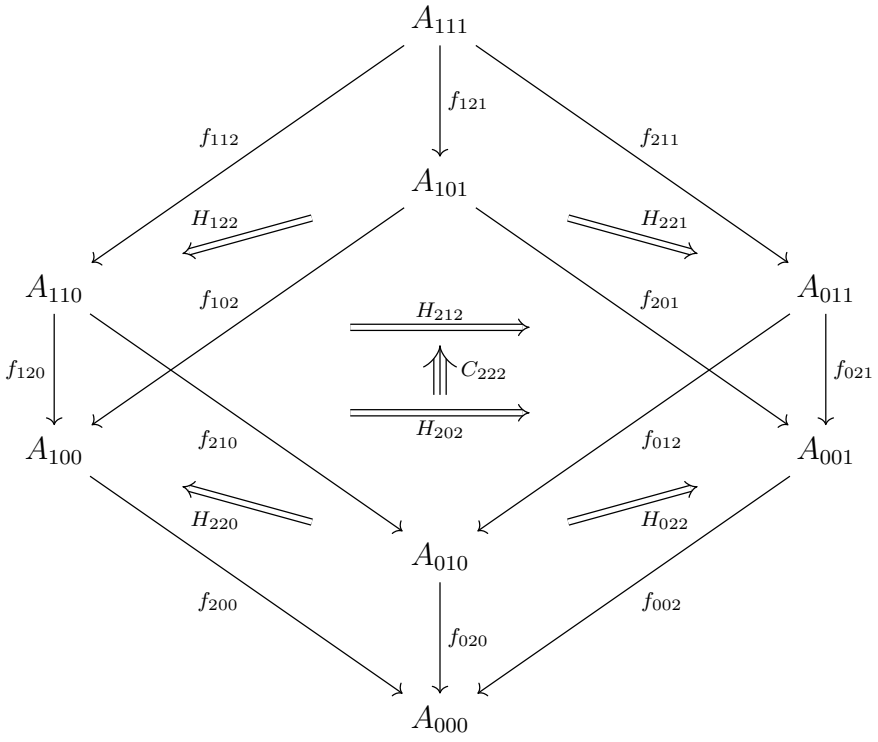
Definition 3.9 (Commuting Square)**.** A commuting square is a pair of a span and a cospan with common feet, together with a homotopy between the two composites.



The data of a commuting square consists of:

- objects A_{11}, A_{01}, A_{10}, A_{00};
- morphisms $f_{21} : A_{11} \to A_{01}$, $f_{12} : A_{11} \to A_{10}$, $f_{02} : A_{01} \to A_{00}$, $f_{20} : A_{10} \to A_{00}$.
- a homotopy $H^r_{22} : A_{11} \to A_{00}$, such that $H^{\mathsf{i0}}_{22} \equiv f_{02} \circ f_{21}$ and $H^{\mathsf{i1}}_{22} \equiv f_{20} \circ f_{12}$.

Definition 3.10 (Commuting Cube)**.** A commuting cube is six compatible commuting squares, together with a homotopy between homotopies that fills the cube.



The data of a commuting cube consists of:

- objects A_{111}, A_{011}, A_{101}, A_{110}, A_{100}, A_{010}, A_{001}, A_{000};
- morphisms

$$f_{112} : A_{111} \to A_{110} \qquad f_{121} : A_{111} \to A_{101} \qquad f_{211} : A_{111} \to A_{011}$$
$$f_{210} : A_{110} \to A_{010} \qquad f_{102} : A_{101} \to A_{100} \qquad f_{021} : A_{011} \to A_{001}$$
$$f_{012} : A_{011} \to A_{010} \qquad f_{201} : A_{101} \to A_{001} \qquad f_{120} : A_{110} \to A_{100}$$
$$f_{200} : A_{100} \to A_{000} \qquad f_{020} : A_{010} \to A_{000} \qquad f_{002} : A_{001} \to A_{000}$$

- homotopies

$$H^r_{122} : A_{111} \to A_{100} \qquad H^r_{212} : A_{111} \to A_{010} \qquad H^r_{221} : A_{111} \to A_{001}$$
$$H^r_{220} : A_{110} \to A_{000} \qquad H^r_{202} : A_{101} \to A_{000} \qquad H^r_{022} : A_{011} \to A_{000}$$

such that

$$H^{\mathsf{i0}}_{122} \equiv f_{102} \circ f_{121} \qquad H^{\mathsf{i0}}_{212} \equiv f_{210} \circ f_{112} \qquad H^{\mathsf{i0}}_{221} \equiv f_{201} \circ f_{121}$$

$$H^{\mathsf{i1}}_{122} \equiv f_{120} \circ f_{112} \qquad H^{\mathsf{i1}}_{212} \equiv f_{012} \circ f_{211} \qquad H^{\mathsf{i1}}_{221} \equiv f_{021} \circ f_{211}$$

$$H^{\mathsf{i0}}_{220} \equiv f_{020} \circ f_{210} \qquad H^{\mathsf{i0}}_{202} \equiv f_{200} \circ f_{102} \qquad H^{\mathsf{i0}}_{022} \equiv f_{020} \circ f_{012}$$

$$H^{\mathsf{i1}}_{220} \equiv f_{200} \circ f_{120} \qquad H^{\mathsf{i1}}_{202} \equiv f_{002} \circ f_{201} \qquad H^{\mathsf{i1}}_{022} \equiv f_{002} \circ f_{021}$$

- a homotopy between homotopies $C^{r,s}_{222} : A_{111} \to A_{000}$, such that

$$C^{\mathsf{i0},s}_{222}(a) \equiv f_{200}(f_{120}(f_{112}(a))) \qquad C^{\mathsf{i1},s}_{222}(a) \equiv f_{002}(f_{021}(f_{211}(a)))$$

$$C^{r,\mathsf{i0}}_{222}(a) \equiv \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{000}}(\; r = \mathsf{i0} \mapsto H^z_{220}(f_{112}(a)),\; r = \mathsf{i1} \mapsto H^z_{022}(f_{211}(a)),\;)(f_{020}(H^r_{212}(a)))$$

$$C^{r,\mathsf{i1}}_{222}(a) \equiv \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{000}}(\; r = \mathsf{i0} \mapsto f_{200}(H^z_{122}(a)),\; r = \mathsf{i1} \mapsto f_{002}(H^z_{221}(a)),\;)(H^r_{202}(f_{121}(a)))$$

4 Stability

Sometimes, while doing ∞-category theory, people will speak of "homotopy limits", and "homotopy colimits". In an ∞-category, we will only use these and never their strict counterparts; so we will refer to them simply as "limits" and "colimits".

4.1 Pointedness

Definition 4.1 (Zero Object (Lurie 2017, Definition 1.1.1.1))**.** A zero object is an object that is both initial and terminal. That is, an object $\mathbf{0}$ such that, for any object X, the spaces $\mathbf{Hom}(\mathbf{0}, X)$ and $\mathbf{Hom}(X, \mathbf{0})$ are contractible. We name these maps, and their composites, 0.

Definition 4.2 (Pointed ∞-Category (Lurie 2017, Definition 1.1.1.1))**.** A pointed ∞-category is an ∞-category with a zero object.

Example 4.3 ($\boldsymbol{\infty}\mathbf{Grpd}_\bullet$)**.** The ∞-category $\boldsymbol{\infty}\mathbf{Grpd}_\bullet$ of pointed spaces has a zero object given by the one-point pointed space.

Remark 4.4 (Enrichment from Pointedness)**.** Let $\mathcal{C}$ be a pointed ∞-category. Then we can canonically enrich it over $\boldsymbol{\infty}\mathbf{Grpd}_\bullet$ by equipping $\mathcal{C}(X, Y)$ with the point $X \to \mathbf{0} \to Y$.

Remark 4.5 (Pointedness from Enrichment)**.** Let $\mathcal{C}$ be a $\boldsymbol{\infty}\mathbf{Grpd}_\bullet$-enriched ∞-category with a terminal object. Then the terminal object $\mathbf{1}$ is also initial with inital morphisms $0 : \mathbf{1} \to Z$.

Remark 4.6 (Pointedness Adjunction)**.** The forgetful functor $U : \mathbf{Pointed}\boldsymbol{\infty}\mathbf{Cat} \to \mathbf{t.o.}\boldsymbol{\infty}\mathbf{Cat}$ mapping a pointed ∞-category to its underlying terminal object-having ∞-category has a right adjoint $\mathbf{Pointed} : \mathbf{t.o.}\boldsymbol{\infty}\mathbf{Cat} \to \mathbf{Pointed}\boldsymbol{\infty}\mathbf{Cat}$ given by the coslice at the terminal object $\mathbf{Pointed}(\mathcal{C}) \equiv \mathcal{C}_{\mathbf{1}/}$. That is, $\mathbf{Pointed}(\mathcal{C})$ is the ∞-category of pointed objects in $\mathcal{C}$. In particular, $\boldsymbol{\infty}\mathbf{Grpd}_\bullet \equiv \mathbf{Pointed}(\boldsymbol{\infty}\mathbf{Grpd})$.

The above equally applies to ordinary categories if we restrict any spaces to be sets.

4.2 Semi-Additivity

Definition 4.7 (Binary Biproduct)**.** Let $\mathcal{C}$ be an ordinary category. A biproduct of two objects A and B of $\mathcal{C}$ is an object that is both the product and coproduct of A and B.

Remark 4.8 (Finite Biproducts)**.** A zero object is a nullary biproduct, and from a zero object and binary biproducts we can build all finite biproducts.

Definition 4.9 (Semi-additive Category)**.** A semi-additive category is a category with finite biproducts.

Example 4.10 (**CMon**)**.** The category of commutative monoids has finite biproducts given by the cartesian product of the underlying sets with pointwise operations.

Remark 4.11 (Enrichment from Semi-additivity)**.** Let $\mathcal{C}$ be a semi-additive category. Then we can canonically enrich it over **CMon**, giving $\mathcal{C}(X, Y)$ the structure of a commutative monoid by

$$0 \equiv X \to \mathbf{0} \to Y \qquad\qquad f + g \equiv X \xrightarrow{\Delta} X \oplus X \xrightarrow{f \oplus g} Y \oplus Y \xrightarrow{\nabla} Y$$

Remark 4.12 (Semi-additivity from Enrichment)**.** Let $\mathcal{C}$ be a **CMon**-enriched category with finite products. Then these products are biproducts, with initial morphisms $0 : \mathbf{1} \to Z$, and copairing given by $[f, g] \equiv f \circ \pi_1 + g \circ \pi_2 : X \times Y \to Z$.

Definition 4.13 (Commutative Monoid Object)**.** Let $\mathcal{C}$ be a category with finite products. Then the data of a commutative monoid object in $\mathcal{C}$ are

- an object C of $\mathcal{C}$;
- morphisms $m : C \times C \to C$ and $e : \mathbf{1} \to C$;
- satisfying

$$m \circ (m \times \mathrm{id}_C) \equiv m \circ (\mathrm{id}_C \times m) \qquad m \circ (e \times \mathrm{id}_C) \equiv \mathrm{id}_C \equiv m \circ (\mathrm{id}_C \times e)$$

$$m \circ (\pi_2, \pi_1) \equiv m$$

Remark 4.14 (Semi-additivity Adjunction)**.** The forgetful functor $U : \mathbf{SemiAdditiveCat} \to \mathbf{CartCat}$ mapping a semi-additive category to its underlying cartesian category has

a right adjoint $\mathbf{CMon} : \mathbf{CartCat} \to \mathbf{SemiAdditiveCat}$ given by taking the category of internal commutative monoids. In particular $\mathbf{CMon} \equiv \mathbf{CMon}(\mathbf{Set})$.

4.3 Stability

Definition 4.15 (Reduced Suspension and Loop Space)**.** Let $\mathcal{C}$ be a pointed ∞-category with pushouts and pullbacks. Then the reduced suspension and loop space functors $\Sigma, \Omega : \mathcal{C} \to \mathcal{C}$ are given by

$$\begin{array}{ccc} A & \longrightarrow & \mathbf{0} \\ \downarrow & \ulcorner & \downarrow \\ \mathbf{0} & \longrightarrow & \Sigma A \end{array} \qquad \begin{array}{ccc} \Omega A & \longrightarrow & \mathbf{0} \\ \downarrow & \lrcorner & \downarrow \\ \mathbf{0} & \longrightarrow & A \end{array}$$

Lemma 4.16 ($\Sigma \dashv \Omega$)**.** *The reduced suspension functor is left adjoint to the loop space functor.*

Proof. For objects $X, Y : \mathcal{C}$, we have $\mathcal{C}(\Sigma X, Y) \cong \Omega(\mathcal{C}(X, Y)) \cong \mathcal{C}(X, \Omega Y)$ where the central Ω is calculated in $\boldsymbol{\infty}\mathbf{Grpd}_\bullet$. □

Definition 4.17 (Stable ∞-Category (Lurie 2017, Proposition 1.4.2.11))**.** A stable ∞-category is a pointed ∞-category with pushouts and pullbacks, where this adjunction is an adjoint equivalence. That is, where the unit and co-unit of the adjunction are equivalences.

Example 4.18 (**Spec**)**.** The ∞-category **Spec** of spectra is stable.

Definition 4.19 (Spectrum Object)**.** Let $\mathcal{C}$ be an ∞-category with finite limits. Then the data of a spectrum object in $\mathcal{C}$ are

- pointed objects C_n of $\mathcal{C}$ for $n \geq 0$;
- with chosen equivalences $C_n \cong \Omega C_{n+1}$ for $n \geq 0$, that preserve the base points.

Definition 4.20 (Stabilisation)**.** The forgetful functor $U : \mathbf{Stab}\boldsymbol{\infty}\mathbf{Cat} \to \mathbf{Lex}\boldsymbol{\infty}\mathbf{Cat}$ mapping a stable ∞-category to its underlying finite limits ∞-category has a right adjoint $\mathbf{Stab} : \mathbf{Lex}\boldsymbol{\infty}\mathbf{Cat} \to \mathbf{Stab}\boldsymbol{\infty}\mathbf{Cat}$ giving by taking the category of spectrum objects. (Lurie 2017, Corollary 1.4.2.17) This is the stabilisation functor, and $\mathbf{Spec} \equiv \mathbf{Stab}(\boldsymbol{\infty}\mathbf{Grpd})$.

Remark 4.21 (Enrichment from Stability)**.** Let $\mathcal{C}$ be a stable ∞-category. Then we can canonically enrich it over **Spec**, giving $\mathcal{C}(X, Y)$ the structure of a spectrum by

$$C_n \equiv \mathcal{C}(\Omega^n X, Y)$$

$$C_n \equiv \mathcal{C}(\Omega^n X, Y) \cong \mathcal{C}(\Omega^{n+1} X, \Omega Y) \cong \Omega(\mathcal{C}(\Omega^{n+1} X, Y)) \equiv \Omega C_{n+1}$$

where we make use of the fact that Ω is fully faithful since $\mathcal{C}$ is stable.

Remark 4.22 (Stability from Enrichment)**.** Let $\mathcal{C}$ be a **Spec**-enriched finite limits ∞-category. Then, given the pattern we have established, we might expect that $\mathcal{C}$ is necessarily stable, but the ∞-category $\mathbf{Spec}_{\geq 0}$ of *connective spectra* is a counterexample, since it is a **Spec**-enriched finite limits ∞-category yet it fails even to have all finite colimits.

This is, however, the only obstruction; so if we also assume $\mathcal{C}$ has finite colimits, then $\mathcal{C}$ is stable. To see this, by (Lurie 2017, Propositions 1.4.2.11 and 1.1.3.4), it suffices to show that Σ and Ω are fully faithful.

$$\begin{aligned}
\mathcal{C}(\Omega X, \Omega Y) &\cong \int_Z \mathbf{Spec}(\mathcal{C}(Z, \Omega X), \mathcal{C}(Z, \Omega Y)) && \text{Yoneda Lemma.} \\
&\cong \int_Z \mathbf{Spec}(\Omega(\mathcal{C}(Z, X)), \Omega(\mathcal{C}(Z, Y))) && \mathcal{C}(Z, -) \text{ preserves limits.} \\
&\cong \int_Z \mathbf{Spec}(\mathcal{C}(Z, X), \mathcal{C}(Z, Y)) && \Omega \text{ is fully faithful on } \mathbf{Spec}. \\
&\cong \mathcal{C}(X, Y) && \text{Yoneda Lemma.}
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}(\Sigma X, \Sigma Y) &\cong \int_Z \mathbf{Spec}(\mathcal{C}(\Sigma Y, Z), \mathcal{C}(\Sigma X, Z)) && \text{Yoneda Lemma.} \\
&\cong \int_Z \mathbf{Spec}(\Omega(\mathcal{C}(Y, Z)), \Omega(\mathcal{C}(X, Z))) && \mathcal{C}(-, Z) \text{ maps colimits to limits.} \\
&\cong \int_Z \mathbf{Spec}(\mathcal{C}(Y, Z), \mathcal{C}(X, Z)) && \Omega \text{ is fully faithful on } \mathbf{Spec}. \\
&\cong \mathcal{C}(X, Y) && \text{Yoneda Lemma.}
\end{aligned}$$

The ends we used exist when $\mathcal{C}$ is small, since **Spec** is complete, and otherwise we can compute them in some extension **SPEC** of large spectra, as in (Kelly 2005).

Intuition 4.23 ($\mathbb{E}_n$-Spaces)**.** For $1 \leq n \leq \infty$, $\mathbb{E}_n$-spaces are ∞-categorical generalisations of monoids and commutative monoids. For a definition, see (Lurie 2017, Chapter 5). Intuitively, for $1 \leq n < \infty$, an $\mathbb{E}_n$-space is a space with a binary operation that is as coherently associative, commutative and unital as concatenation in an n-fold loop space, but possibly without inverses. In particular, for a pointed space A, the n-fold loop space $\Omega^n A$ is an $\mathbb{E}_n$-space. An $\mathbb{E}_\infty$-space must satisfy all the coherence conditions of an $\mathbb{E}_n$-space for $1 \leq n < \infty$.

Remark 4.24 (Relation of Spectra to $\mathbb{E}_n$-Spaces)**.** Given a spectrum $C_\bullet$, we have $C_0 \cong \Omega^n(C_n)$; so C_0 is an $\mathbb{E}_n$-space, for $1 \leq n < \infty$; then C_0 is an $\mathbb{E}_\infty$-space. This gives a functor $U : \mathbf{Spec} \to \mathbb{E}_\infty\text{-}\mathbf{Space}$. This functor is lax monoidal (Gepner, Groth and Nikolaus 2015); so it induces a functor $U\text{-}\boldsymbol{\infty}\mathbf{Cat} : \mathbf{Spec}\text{-}\boldsymbol{\infty}\mathbf{Cat} \to \mathbb{E}_\infty\text{-}\mathbf{Space}\text{-}\boldsymbol{\infty}\mathbf{Cat}$ sending spectra-enriched ∞-categories to $\mathbb{E}_\infty$-space-enriched categories. In particular, every stable ∞-category is enriched over $\mathbb{E}_\infty$-spaces.

To define a type theory for semi-additive categories, we could axiomatise finite products and give every type the structure of a commutative monoid. Correspondingly, we might hope to obtain a type theory for stable ∞-categories by axiomatising

finite limits and colimits and giving every type the structure of a spectrum. Giving a commutative monoid structure is convenient since commutative monoids are presented by operations and equations, but it is not so simple to give the structure of a spectrum, which includes additional objects. We know stable ∞-categories are also enriched over $\mathbb{E}_\infty$-spaces; so we will, instead, endow our types with $\mathbb{E}_\infty$-space structure.

Enrichment over $\mathbb{E}_\infty$-spaces is not sufficient to ensure stability of a finitely bicomplete ∞-category, but the rules of our type theory will be stronger due to the context and we conjecture that they are enough for stability. We expect this to work since $\mathbb{E}_\infty$-space-enrichment is sufficient for the existence of biproducts, and (Riley, Finster and Licata 2021) axiomatise biproducts and derive stability. Then our plan is to present StabTT, show that we have biproducts, and adapt the argument in (Riley, Finster and Licata 2021, Section 4) to derive stability.

5 Stable Type Theory

Stable Type Theory is cubical type theory with type-formers for finite limits and colimits, and $\mathbb{E}_\infty$-space structure for every type. StabTT has dimension and formula layers as in CuTT, but the top layer will differ to accommodate the enrichment.

As with Kan composition, the $\mathbb{E}_\infty$-space structure is defined by external induction. For negative types, we defer to the operations on the components. For positive types, we freely add terms for each operation, and the eliminator maps these to the corresponding operations in the codomain.

If StabTT were to have negative Σ types, we would have to implement their $\mathbb{E}_\infty$ operations in terms of those for the components. If we try to define $p + p'$ for $p, p' : \sum_{a:A} B$, we can choose $\pi_1(p)+\pi_1(p') : A$ for the first component, then we need a second component of type $B[\pi_1(p)+\pi_1(p')/a]$, but all we have are $\pi_2(p) : B[\pi_1(p)/a]$ and $\pi_2(p') : B[\pi_1(p')/a]$, which we cannot add since they have different types. Then we cannot implement addition for $\sum_{a:A} B$.

One solution would be to define addition not on each type but on each type family. Then we would be able to add $\pi_2(p) : B[\pi_1(p)/a]$ and $\pi_2(p') : B[\pi_1(p')/a]$ to get a value of type $B[\pi_1(p) + \pi_1(p')/a]$, as we needed. This could be interesting to investigate in the future.

A simpler solution, however, is available to us: we will disallow types in StabTT from depending on variables in the context. Then, in place of the judgments $\Xi|\Phi \vdash \Gamma$ `ctx` and $\Xi|\Phi|\Gamma \vdash A$ `type`, we will have Γ `ctx` and A `obj`.

5.1 Enrichment

For the $\mathbb{E}_\infty$-space structure, we have term-forming operations 0 and $+$.

$$\frac{\Xi \vdash \Phi\ \mathtt{ftx} \qquad \Gamma\ \mathtt{ctx} \qquad A\ \mathtt{obj}}{\Xi|\Phi|\Gamma \vdash 0 : A}\ 0 \qquad \frac{\Xi|\Phi|\Gamma \vdash a : A \qquad \Xi|\Phi|\Gamma \vdash b : A}{\Xi|\Phi|\Gamma \vdash a + b : A}\ +$$

These operations are not strictly associative, commutative or unital, but they are up to paths; so we have operations forming these paths. These resemble the unitors, associator, and braiding of a braided monoidal category.

$$\frac{\Xi|\Phi|\Gamma \vdash a : A \qquad \Xi \vdash r : \mathbb{I}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \lambda^r(a) : A \\ \Xi|\Phi|\Gamma \vdash \lambda^{\mathsf{i0}}(a) \equiv a : A \\ \Xi|\Phi|\Gamma \vdash \lambda^{\mathsf{i1}}(a) \equiv 0 + a : A\end{array}}\ \lambda \qquad \frac{\Xi|\Phi|\Gamma \vdash a : A \qquad \Xi \vdash r : \mathbb{I}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \rho^r(a) : A \\ \Xi|\Phi|\Gamma \vdash \rho^{\mathsf{i0}}(a) \equiv a : A \\ \Xi|\Phi|\Gamma \vdash \rho^{\mathsf{i1}}(a) \equiv a + 0 : A\end{array}}\ \rho$$

$$\frac{\begin{array}{c}\Xi|\Phi|\Gamma \vdash a : A \\ \Xi|\Phi|\Gamma \vdash b : A \\ \Xi \vdash r : \mathbb{I}\end{array}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \sigma^r(a,b) : A \\ \Xi|\Phi|\Gamma \vdash \sigma^{\mathsf{i0}}(a,b) \equiv a + b : A \\ \Xi|\Phi|\Gamma \vdash \sigma^{\mathsf{i1}}(a,b) \equiv b + a : A\end{array}}\ \sigma \qquad \frac{\begin{array}{c}\Xi|\Phi|\Gamma \vdash a : A \\ \Xi|\Phi|\Gamma \vdash b : A \\ \Xi|\Phi|\Gamma \vdash c : A \\ \Xi \vdash r : \mathbb{I}\end{array}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \alpha^r(a,b,c) : A \\ \Xi|\Phi|\Gamma \vdash \alpha^{\mathsf{i0}}(a,b,c) \equiv (a + b) + c : A \\ \Xi|\Phi|\Gamma \vdash \alpha^{\mathsf{i1}}(a,b,c) \equiv a + (b + c) : A\end{array}}\ \alpha$$

Similarly, these do not satisfy the triangle, pentagon and hexagon identities judgmentally, but we have rules that introduce paths between paths that correspond to these identities. And these have higher coherators forever.

We also have paths exhibiting that every map respects the $\mathbb{E}_\infty$ operations. There are, again, infinitely many of these beginning with

$$\frac{\begin{array}{c}\Xi \vdash \Phi\ \mathtt{ftx} \qquad \Gamma\ \mathtt{ctx} \\ f : A \to B \qquad \Xi \vdash r : \mathbb{I}\end{array}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \delta_0^r(f) : B \\ \Xi|\Phi|\Gamma \vdash \delta_0^{\mathsf{i0}}(f) \equiv 0 : B \\ \Xi|\Phi|\Gamma \vdash \delta_0^{\mathsf{i1}}(f) \equiv f(0) : B\end{array}}\ \delta_0 \qquad \frac{\begin{array}{c}\Xi|\Phi|\Gamma \vdash a : A \qquad \Xi|\Phi|\Gamma \vdash a' : A \\ f : A \to B \qquad \Xi \vdash r : \mathbb{I}\end{array}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \delta_+^r(f;a,a') : B \\ \Xi|\Phi|\Gamma \vdash \delta_+^{\mathsf{i0}}(f;a,a') \equiv f(a) + f(a') : B \\ \Xi|\Phi|\Gamma \vdash \delta_+^{\mathsf{i1}}(f;a,a') \equiv f(a + a') : B\end{array}}\ \delta_+$$

We also have δ_{δ_0} and so on, showing every map respects the distributivity structure.

It would be good if we could capture all these coherators in finitely many rules, as with Kan composition. We discuss this in Subsection 7.6.

5.2 Zero Type

The zero type is very simple to present. We start with the rules for a negative unit type.

$$\frac{}{\mathbf{0}\ \mathtt{obj}}\ \mathbf{0}\text{-FORM} \qquad \frac{\Xi \vdash \Phi\ \mathtt{ftx} \qquad \Gamma\ \mathtt{ctx}}{\Xi|\Phi|\Gamma \vdash * : \mathbf{0}}\ \mathbf{0}\text{-INTRO} \qquad \frac{\Xi|\Phi|\Gamma \vdash s : \mathbf{0}}{\Xi|\Phi|\Gamma \vdash s \equiv * : \mathbf{0}}\ \mathbf{0}\text{-UNIQ}$$

We define Kan composition and the $\mathbb{E}_\infty$ operations to all always return $*$, which makes sense since, by the uniqueness rule, these equalities would hold already.

We also want $\mathbf{0}$ to be initial. In pointed spaces, the unique map out of the initial object sends everything to the point; likewise, we have a derivable term-former `absurd` which takes a term of type $\mathbf{0}$, discards it, and returns the term 0 of the appropriate type. We get uniqueness of this map from the enriched structure.

To see this, let $f : \mathbf{0} \to Z$ be any map out of $\mathbf{0}$, then we can construct a path N^r from 0 to f by $N^r(p) \equiv \delta_0^r(f)$, which has $N^{\mathsf{i0}}(p) \equiv 0$, and $N^{\mathsf{i1}}(p) \equiv f(0) \equiv f(*) \equiv f(p)$, since every term of type $\mathbf{0}$ is judgmentally equal to $*$. Then the space of maps $\mathbf{0} \to Z$ is contractible.

5.3 Pullback Types

The data required to form a pullback type is that of a cospan.

$$\frac{f : A \to C \qquad g : B \to C}{A \underset{C}{\times_{f,g}} B\ \mathtt{obj}}\ \times\text{-FORM}$$

The data required to make a term of a pullback type is that of two elements and a path between their images under the maps in the cospan.

$$\frac{\begin{array}{c} f : A \to C \qquad g : B \to C \\ \Xi|\Phi|\Gamma \vdash a : A \qquad \Xi|\Phi|\Gamma \vdash b : B \qquad \Xi, x : \mathbb{I}|\Phi|\Gamma \vdash c : C \\ \Xi|\Phi|\Gamma \vdash c[\mathsf{i0}/x] \equiv f(a) : C \qquad \Xi|\Phi|\Gamma \vdash c[\mathsf{i1}/x] \equiv g(b) : C \end{array}}{\Xi|\Phi|\Gamma \vdash (a, b, x \mapsto c) : A \underset{C}{\times_{f,g}} B}\ \times\text{-INTRO}$$

Once we have an element, the elimination rule allows us to retrieve these same three pieces of data and two judgmental equalities.

$$\frac{\Xi|\Phi|\Gamma \vdash p : A \underset{C}{\times_{f,g}} B \qquad \Xi \vdash r : \mathbb{I}}{\begin{array}{c} \Xi|\Phi|\Gamma \vdash \pi_1(p) : A \qquad \Xi|\Phi|\Gamma \vdash \pi_2(p) : B \qquad \Xi|\Phi|\Gamma \vdash \pi_3^r(p) : C \\ \Xi|\Phi|\Gamma \vdash \pi_3^{\mathsf{i0}}(p) \equiv f(\pi_1(p)) : C \qquad \Xi|\Phi|\Gamma \vdash \pi_3^{\mathsf{i1}}(p) \equiv g(\pi_2(p)) : C \end{array}}\ \times\text{-ELIM}$$

These operations are mutually inverse.

$$\frac{\begin{array}{c}\times\text{-COMP}\\ \Xi|\Phi|\Gamma \vdash a : A \qquad \Xi|\Phi|\Gamma \vdash b : B \qquad \Xi, x : \mathbb{I}|\Phi|\Gamma \vdash c : C \\ \Xi|\Phi|\Gamma \vdash c[\mathsf{i0}/x] \equiv f(a) : C \qquad \Xi|\Phi|\Gamma \vdash c[\mathsf{i1}/x] \equiv g(b) : C \qquad \Xi \vdash r : \mathbb{I}\end{array}}{\begin{array}{c}\Xi|\Phi|\Gamma \vdash \pi_1((a, b, x \mapsto c)) \equiv a : A \qquad \Xi|\Phi|\Gamma \vdash \pi_2((a, b, x \mapsto c)) \equiv b : B \\ \Xi|\Phi|\Gamma \vdash \pi_3^r((a, b, x \mapsto c)) \equiv c[r/x] : C\end{array}}$$

$$\frac{\begin{array}{c}\times\text{-UNIQ}\\ \Xi|\Phi|\Gamma \vdash p : A \underset{C}{\times}_f{}_g B\end{array}}{\Xi|\Phi|\Gamma \vdash (\pi_1(p), \pi_2(p), x \mapsto \pi_3^x(p)) \equiv p : A \underset{C}{\times}_f{}_g B}$$

Kan composition for pullback types delegates to the constituent types. The rule resembles the rule that the type $\sum_{a:A} \sum_{b:B} \mathrm{Path}_C(f(a), g(b))$ would have in (Angiuli, Brunerie et al. 2021).

$$\frac{\begin{array}{c}\times\text{-KAN}\\ \Xi \vdash r : \mathbb{I} \qquad \Xi \vdash r' : \mathbb{I} \qquad \Xi, z : \mathbb{I}|\Phi, \phi|\Gamma \vdash t : A \underset{C}{\times}_f{}_g B \\ \Xi|\Phi|\Gamma \vdash b : A \underset{C}{\times}_f{}_g B \qquad \Xi|\Phi, \phi|\Gamma \vdash t[r/z] \equiv b : A \underset{C}{\times}_f{}_g B\end{array}}{\begin{array}{l}\Xi|\Phi|\Gamma \vdash \mathtt{com}^{z:r\to r'}_{A \underset{C}{\times}_f{}_g B}(\phi \mapsto t)(b) \\ \quad \equiv \big(\mathtt{com}^{z:r\to r'}_{A}(\phi \mapsto \pi_1(t))(\pi_1(b)), \\ \qquad \mathtt{com}^{z:r\to r'}_{B}(\phi \mapsto \pi_2(t))(\pi_2(b)), \\ \qquad x \mapsto \mathtt{com}^{z':r\to r'}_{C}(\\ \qquad\qquad \phi \mapsto \pi_3^x(t[z'/z]), \\ \qquad\qquad x = \mathsf{i0} \mapsto f(\mathtt{com}^{z:r\to z'}_{A}(\phi \mapsto \pi_1(t))(\pi_1(b))), \\ \qquad\qquad x = \mathsf{i1} \mapsto g(\mathtt{com}^{z:r\to z'}_{B}(\phi \mapsto \pi_2(t))(\pi_2(b))), \\ \qquad)(\pi_3^x(b))\big) : A \underset{C}{\times}_f{}_g B\end{array}}$$

Similarly, the enriched structure delegates to the components.

$$\frac{\begin{array}{c}\times\text{-0}\\ \Xi \vdash \Phi\ \mathtt{ftx} \qquad \Gamma\ \mathtt{ctx} \qquad f : A \to C \qquad g : B \to C\end{array}}{\begin{array}{l}\Xi|\Phi|\Gamma \vdash 0 \equiv \big(0, 0, x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{C}(\\ \qquad\qquad x = \mathsf{i0} \mapsto \delta_0^z(f), \\ \qquad\qquad x = \mathsf{i1} \mapsto \delta_0^z(g), \\ \qquad\qquad)(0)\big) : A \underset{C}{\times}_f{}_g B\end{array}}$$

$$\frac{\Xi|\Phi|\Gamma \vdash p : A \underset{C}{\times}_{f,g} B \qquad \Xi|\Phi|\Gamma \vdash q : A \underset{C}{\times}_{f,g} B}{\times\text{-}+}$$

$$\begin{aligned}\Xi|\Phi|\Gamma \vdash p + q \equiv \big(&\pi_1(p) + \pi_1(q),\\ &\pi_2(p) + \pi_2(q),\\ &x \mapsto \mathtt{com}_C^{z:\mathsf{i0}\to\mathsf{i1}}(\\ &\qquad x = \mathsf{i0} \mapsto \delta_+^z(f; \pi_1(p), \pi_1(q)),\\ &\qquad x = \mathsf{i1} \mapsto \delta_+^z(g; \pi_2(p), \pi_2(q)),\\ &\quad)(\pi_3^x(p) + \pi_3^x(q))\big) : A \underset{C}{\times}_{f,g} B\end{aligned}$$

We see that in both example rules we try to simply apply the operation componentwise, but we have to adjust the endpoints of the path, since $\pi_3^{\mathsf{i0}}(p) + \pi_3^{\mathsf{i0}}(q)$ is judgmentally equal to $f(\pi_1(p)) + f(\pi_1(q))$ rather than $f(\pi_1(p) + \pi_1(q))$.

5.4 Pushout Types

Symmetrically, the data required to form a pushout type is that of a span.

$$\frac{f : C \to A \qquad g : C \to B}{A \underset{C}{\sqcup}_{f,g} B\ \mathtt{obj}}\ \sqcup\text{-FORM}$$

Pushout types are higher inductive types; so we have introduction rules for each of their constructors. We have point constructors ι_1, ι_2 that include A, B into the pushout, respectively, and a path constructor ι_3^r with judgmental equalities ensuring that the paths connect the chosen points in the images of A and B.

$$\frac{f : C \to A \qquad g : C \to B \qquad \Xi|\Phi|\Gamma \vdash a : A}{\Xi|\Phi|\Gamma \vdash \iota_1(a) : A \underset{C}{\sqcup}_{f,g} B}\ \sqcup\text{-INTRO}_1$$

$$\frac{f : C \to A \qquad g : C \to B \qquad \Xi|\Phi|\Gamma \vdash b : B}{\Xi|\Phi|\Gamma \vdash \iota_2(b) : A \underset{C}{\sqcup}_{f,g} B}\ \sqcup\text{-INTRO}_2$$

$$\frac{f : C \to A \qquad g : C \to B \qquad \Xi|\Phi|\Gamma \vdash c : C \qquad \Xi \vdash r : \mathbb{I}}{\Xi|\Phi|\Gamma \vdash \iota_3^r(c) : A \underset{C}{\sqcup}_{f,g} B}\ \sqcup\text{-INTRO}_3$$

$$\Xi|\Phi|\Gamma \vdash \iota_3^{\mathsf{i0}}(c) \equiv \iota_1(f(c)) : A \underset{C}{\sqcup}_{f,g} B \qquad \Xi|\Phi|\Gamma \vdash \iota_3^{\mathsf{i1}}(c) \equiv \iota_2(g(c)) : A \underset{C}{\sqcup}_{f,g} B$$

Higher inductive types have `match`-style eliminators; so our first attempt at the elimination rule would have one case for each of our constructors, subject to two judgmental equalities.

$$\frac{\begin{array}{c}\Xi|\Phi|\Gamma, a : A \vdash s : Z \qquad \Xi|\Phi|\Gamma, b : B \vdash t : Z \qquad \Xi, x : \mathbb{I}|\Phi|\Gamma, c : C \vdash u : Z\\ \Xi|\Phi|\Gamma, c : C \vdash u[\mathsf{i0}/x] \equiv s[f(c)/a] : Z \qquad \Xi|\Phi|\Gamma, c : C \vdash u[\mathsf{i1}/x] \equiv t[g(c)/b] : Z\\ \Xi|\Phi|\Gamma \vdash p : A \underset{C}{\sqcup}_{f,g} B\end{array}}{\Xi|\Phi|\Gamma \vdash \mathtt{match}\ p\ \{\iota_1(a) \mapsto s,\ \iota_2(b) \mapsto t,\ \iota_3^x(c) \mapsto u\} : Z}\ \sqcup\text{-ELIM}$$

As with the suspension type, the judgmental uniqueness rule would be too strong; so we need to be able to prove a propositional uniqueness principle from the elimination rule. The rule as we have written it is not strong enough for this, since we have no rule asserting an equality between a map of the form $p.\mathtt{match}\ p\ \{\cdots\}$ and a map not of that form.

When we perform a Kan composition, we can force it to agree with a partial term; we will add this for $\sqcup$-ELIM, and then we can prove uniqueness. That is, we have a partial term $v : Z$ that agrees judgmentally with s, t, u in the appropriate way, and we get a total term that agrees judgmentally with v where it is defined.

$\sqcup$-ELIM

$$\frac{\begin{array}{c}\Xi|\Phi|\Gamma, a : A \vdash s : Z \qquad \Xi|\Phi|\Gamma, b : B \vdash t : Z \qquad \Xi, x : \mathbb{I}|\Phi|\Gamma, c : C \vdash u : Z \\ \Xi|\Phi|\Gamma, c : C \vdash u[\mathsf{i0}/x] \equiv s[f(c)/a] : Z \qquad \Xi|\Phi|\Gamma, c : C \vdash u[\mathsf{i1}/x] \equiv t[g(c)/b] : Z \\ \Xi|\Phi, \phi|\Gamma, p' : A \underset{C}{\sqcup}_{f,g} B \vdash v : Z \\ \Xi|\Phi, \phi|\Gamma, a : A \vdash v[\iota_1(a)/p'] \equiv s : Z \qquad \Xi|\Phi, \phi|\Gamma, b : B \vdash v[\iota_2(b)/p'] \equiv t : Z \\ \Xi, x : \mathbb{I}|\Phi, \phi|\Gamma, c : C \vdash v[\iota_3^x(c)/p'] \equiv u : Z \qquad \Xi|\Phi|\Gamma \vdash p : A \underset{C}{\sqcup}_{f,g} B\end{array}}{\Xi|\Phi|\Gamma \vdash \mathtt{match}\ p\ \{\iota_1(a) \mapsto s,\ \iota_2(b) \mapsto t,\ \iota_3^x(c) \mapsto u,\ p'\ \mathtt{if}\ \phi \mapsto v\} : Z}$$

This has the expected computational behaviour when p is $\iota_1(e)$, $\iota_2(e)$ or $\iota_3^r(e)$, and whatever the value of p, if ϕ holds, the value is $v[p/p']$.

To see uniqueness, suppose we have two maps $h_0, h_1 : A \underset{C}{\sqcup}_{f,g} B \to Z$, and suppose $h_0(\iota_1(e)) \equiv h_1(\iota_1(e))$, $h_0(\iota_2(e)) \equiv h_1(\iota_2(e))$ and $h_0(\iota_3^r(e)) \equiv h_1(\iota_3^r(e))$. Then we can define a homotopy H^s between h_0 and h_1 by

$$\begin{aligned} H^s(p) \equiv \mathtt{match}\ p\ \{ & \\ & \iota_1(a) \mapsto h_0(\iota_1(a)), \\ & \iota_2(b) \mapsto h_0(\iota_2(b)), \\ & \iota_3^x(c) \mapsto h_0(\iota_3^x(c)), \\ & p'\ \mathtt{if}\ s = \mathsf{i0} \mapsto h_0(p'), \\ & p'\ \mathtt{if}\ s = \mathsf{i1} \mapsto h_1(p'), \\ \} & \end{aligned}$$

The pushout is a positive type; so we define Kan composition and the $\mathbb{E}_\infty$ operations to return corresponding constructors $\mathtt{vcom}$, $\mathtt{v0}$, $\mathtt{v+}$, $\mathtt{v}\delta_0^r(\iota_1)$ and so on, and we must explain how the eliminator acts on them. The eliminator maps $\mathtt{vcom}$'s to $\mathtt{com}$'s as for suspension types.

To write the computation rules for eliminating the free $\mathbb{E}_\infty$ operations, we need to forbid s, t, u, v from using the variables in Γ. Abbreviate

$$M(p) \equiv \mathtt{match}\ p\ \{\iota_1(a) \mapsto s,\ \iota_2(b) \mapsto t,\ \iota_3^x(c) \mapsto u,\ p'\ \mathtt{if}\ \phi \mapsto v\}$$

then $M(\mathtt{v0}) \equiv \mathtt{com}_Z^{z:\mathsf{i0}\to\mathsf{i1}}(\phi \mapsto \delta_0^z(p'.v))(0)$, and
$M(p\ \mathtt{v+}\ q) \equiv \mathtt{com}_Z^{z:\mathsf{i0}\to\mathsf{i1}}(\phi \mapsto \delta_+^z(p'.v; p, q))(M(p) + M(q))$ and so on.

Then our final attempt is

$$\frac{\begin{array}{c}\Xi|\Phi|a : A \vdash s : Z \qquad \Xi|\Phi|b : B \vdash t : Z \qquad \Xi, x : \mathbb{I}|\Phi|c : C \vdash u : Z \\ \Xi|\Phi|c : C \vdash u[\mathsf{i0}/x] \equiv s[f(c)/a] : Z \qquad \Xi|\Phi|c : C \vdash u[\mathsf{i1}/x] \equiv t[g(c)/b] : Z \\ \Xi|\Phi, \phi|p' : A \underset{C}{\sqcup}_{f,g} B \vdash v : Z \\ \Xi|\Phi, \phi|a : A \vdash v[\iota_1(a)/p'] \equiv s : Z \qquad \Xi|\Phi, \phi|b : B \vdash v[\iota_2(b)/p'] \equiv t : Z \\ \Xi, x : \mathbb{I}|\Phi, \phi|c : C \vdash v[\iota_3^x(c)/p'] \equiv u : Z \qquad \Xi|\Phi|\Gamma \vdash p : A \underset{C}{\sqcup}_{f,g} B\end{array}}{\Xi|\Phi|\Gamma \vdash \mathtt{match}\ p\ \{\iota_1(a) \mapsto s,\ \iota_2(b) \mapsto t,\ \iota_3^x(c) \mapsto u,\ p'\ \mathtt{if}\ \phi \mapsto v\} : Z}\ \sqcup\text{-ELIM}$$

5.5 Special Cases

We will make use of four special cases of pushout and pullback types. When the feet of the span or cospan are zero, we get the reduced suspension and loop space, respectively. When the body of the span or cospan is zero, we get sum and product.

$$\Sigma A \equiv \mathbf{0} \underset{A}{\sqcup}_0 \mathbf{0} \qquad \Omega A \equiv \mathbf{0} \underset{A}{\times}_0 \mathbf{0}$$

$$A + B \equiv A \underset{\mathbf{0}}{\sqcup}_0 B \qquad A \times B \equiv A \underset{\mathbf{0}}{\times}_0 B$$

Then, when constructing terms of type ΩA or $A \times B$, we do not need to write out the data that we know are zero, with the following definitions.

$$(x \mapsto a) \equiv (0, 0, x \mapsto a) : \Omega A \qquad (a, b) \equiv (a, b, x \mapsto 0) : A \times B$$

Likewise, when eliminating terms of type ΣA or $A + B$, we can omit the cases that we know must be zero, with the following definitions.

$$\frac{\begin{array}{c}\Xi, x : \mathbb{I}|\Phi|a : A \vdash u : Z \\ \Xi|\Phi|a : A \vdash u[\mathsf{i0}/x] \equiv 0 : Z \qquad \Xi|\Phi|a : A \vdash u[\mathsf{i1}/x] \equiv 0 : Z \\ \Xi|\Phi, \phi|p' : \Sigma A \vdash v : Z \qquad \Xi, x : \mathbb{I}|\Phi, \phi|a : A \vdash v[\iota_3^x(a)/p'] \equiv u : Z \\ \Xi|\Phi|\Gamma \vdash p : A + B\end{array}}{\begin{array}{l}\Xi|\Phi|\Gamma \vdash \mathtt{match}\ p\ \{ \\ \qquad \iota_3^x(a) \mapsto u, \\ \qquad p'\ \mathtt{if}\ \phi \mapsto v, \\ \} \equiv \mathtt{match}\ p\ \{ \\ \qquad \iota_1(e) \mapsto 0, \\ \qquad \iota_2(e) \mapsto 0, \\ \qquad \iota_3^x(a) \mapsto u, \\ \qquad p'\ \mathtt{if}\ \phi \mapsto v, \\ \} : Z\end{array}}\ \Sigma\text{-ELIM-DEF}$$

$+$-ELIM-DEF

$$\frac{\begin{array}{c}\Xi|\Phi|a : A \vdash s : Z \qquad \Xi|\Phi|b : B \vdash t : Z \qquad \Xi|\Phi, \phi|p' : A + B \vdash v : Z \\ \Xi|\Phi, \phi|a : A \vdash v[\iota_1(a)/p'] \equiv s : Z \qquad \Xi|\Phi, \phi|b : B \vdash v[\iota_2(b)/p'] \equiv t : Z \\ \Xi|\Phi|\Gamma \vdash p : A + B\end{array}}{\begin{array}{l}\Xi|\Phi|\Gamma \vdash \texttt{match}\ p\ \{ \\ \quad \iota_1(a) \mapsto s, \\ \quad \iota_2(b) \mapsto t, \\ \quad p'\ \texttt{if}\ \phi \mapsto v, \\ \} \equiv \texttt{match}\ p\ \{ \\ \quad \iota_1(a) \mapsto s, \\ \quad \iota_2(b) \mapsto t, \\ \quad \iota_3^x(c) \mapsto \mathtt{com}_Z^{z:\mathsf{i0}\to\mathsf{i1}}(\\ \qquad \phi \mapsto \delta_0^z((p'.v) \circ \iota_3^x), \\ \qquad x = \mathsf{i0} \mapsto \delta_0^z(a.s), \\ \qquad x = \mathsf{i1} \mapsto \delta_0^z(b.t), \\ \quad)(0), \\ \quad p'\ \texttt{if}\ \phi \mapsto v, \\ \} : Z\end{array}}$$

5.6 Biproducts

As with semi-additive categories, for types A, B, we have two natural maps $\phi, \psi : A + B \to A \times B$, given by

$$\begin{array}{ll}\phi(p) \equiv (\texttt{match}\ p\ \{ & \psi(p) \equiv \texttt{match}\ p\ \{ \\ \quad \iota_1(a) \mapsto a, & \quad \iota_1(a) \mapsto (a, 0), \\ \quad \iota_2(b) \mapsto 0, & \quad \iota_2(b) \mapsto (0, b), \\ \}, & \} \\ \texttt{match}\ p\ \{ & \\ \quad \iota_1(a) \mapsto 0, & \\ \quad \iota_2(b) \mapsto b, & \\ \}) & \end{array}$$

The idea behind both of these maps is the block matrix

$$\begin{bmatrix} \mathrm{id}_A & 0 \\ 0 & \mathrm{id}_B \end{bmatrix}$$

For semi-additive categories, the two maps agree on the nose, but for us they are equal only up to a path. We construct this path with the eliminator for $+$ as follows.

$$\begin{aligned}
&\chi^r : A + B \to A \times B \\
&\chi^r(p) \equiv \mathtt{match}\ p\ \{ \\
&\qquad\qquad \iota_1(a) \mapsto (a, 0), \\
&\qquad\qquad \iota_2(b) \mapsto (0, b), \\
&\qquad\qquad p'\ \mathtt{if}\ r = \mathsf{i0} \mapsto \phi(p'), \\
&\qquad\qquad p'\ \mathtt{if}\ r = \mathsf{i1} \mapsto \psi(p'), \\
&\qquad\quad \}
\end{aligned}$$

To show that we have biproducts we must prove that ψ, and hence equally ϕ, is an equivalence. For this, let $g, h : A \times B \to A + B$ be given by

$$g(p) \equiv h(p) \equiv \iota_1(\pi_1(p)) + \iota_2(\pi_2(p))$$

First, consider the composite $\psi \circ g$. We see

$$\begin{aligned}
\psi(g(p)) &\equiv \mathtt{match}\ \iota_1(\pi_1(p)) + \iota_2(\pi_2(p))\ \{ \\
&\qquad \iota_1(a) \mapsto (a, 0), \\
&\qquad \iota_2(b) \mapsto (0, b), \\
&\quad \} \\
&\equiv \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A\times B}()(\mathtt{match}\ \iota_1(\pi_1(p))\ \{ \\
&\qquad\qquad\qquad \iota_1(a) \mapsto (a, 0), \\
&\qquad\qquad\qquad \iota_2(b) \mapsto (0, b), \\
&\qquad\qquad \} + \mathtt{match}\ \iota_2(\pi_2(p))\ \{ \\
&\qquad\qquad\qquad\quad \iota_1(a) \mapsto (a, 0), \\
&\qquad\qquad\qquad\quad \iota_2(b) \mapsto (0, b), \\
&\qquad\qquad\qquad \}) \\
&\equiv \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A\times B}()((\pi_1(p), 0) + (0, \pi_2(p))) \\
&\equiv \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A\times B}()((\pi_1(p) + 0, 0 + \pi_2(p))) \\
&\rightsquigarrow (\pi_1(p), \pi_2(p)) \\
&\equiv p
\end{aligned}$$

Then we have a path $M^r : A \times B \to A \times B$ between $\mathrm{id}_{A\times B}$ and $\psi \circ g$.

$$M^r(p) \equiv \mathtt{com}^{z:\mathsf{i0}\to r}_{A\times B}()((\rho^r(\pi_1(p)), \lambda^r(\pi_2(p))))$$

When we consider the other composite, $h \circ \psi$, we see that there are no computation or uniqueness rules that we can usefully apply. The composite is, however, a map

out of $A + B$; so we seek to construct a path $N^r : A + B \to A + B$ connecting it to the identity using the eliminator for $+$.

To that end, observe the value of $h \circ \psi$ on $\iota_1(a)$.

$$\begin{aligned}
h(\psi(\iota_1(a))) &\equiv h(\mathtt{match}\ \iota_1(a)\ \{ \\
&\qquad\qquad \iota_1(a) \mapsto (a, 0), \\
&\qquad\qquad \iota_2(b) \mapsto (0, b), \\
&\qquad\quad \}) \\
&\equiv h((a, 0)) \\
&\equiv \iota_1(\pi_1((a, 0))) + \iota_2(\pi_2((a, 0))) \\
&\equiv \iota_1(a) + \iota_2(0) \\
&\rightsquigarrow \iota_1(a) + 0 \\
&\rightsquigarrow \iota_1(a)
\end{aligned}$$

Then we have a path connecting $\iota_1(a)$ to $h(\psi(\iota_1(a)))$ and, together with the corresponding path for $\iota_2(b)$, it makes a path from id_{A+B} to $h \circ \psi$.

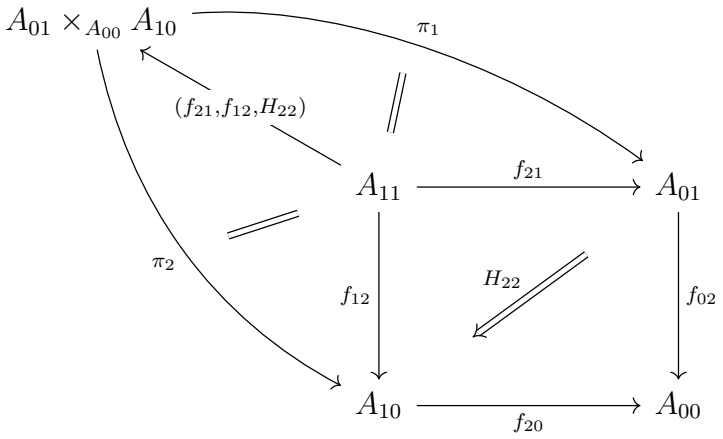
$$\begin{aligned}
N^r(p) \equiv \mathtt{match}\ p\ \{ & \\
&\iota_1(a) \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A+B}(\\
&\qquad\qquad r = \mathsf{i0} \mapsto \iota_1(a), \\
&\qquad\qquad r = \mathsf{i1} \mapsto \iota_1(a) + \delta^z_0(\iota_2), \\
&\qquad\quad)(\rho^r(\iota_1(a))), \\
&\iota_2(b) \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A+B}(\\
&\qquad\qquad r = \mathsf{i0} \mapsto \iota_2(b), \\
&\qquad\qquad r = \mathsf{i1} \mapsto \delta^z_0(\iota_1) + \iota_2(b), \\
&\qquad\quad)(\lambda^r(\iota_2(b))), \\
&p' \ \mathtt{if}\ r = \mathsf{i0} \mapsto p', \\
&p' \ \mathtt{if}\ r = \mathsf{i1} \mapsto h(\psi(p')), \\
\}&
\end{aligned}$$

Thus ψ is an equivalence, and the category of types has biproducts.

6 Theorems

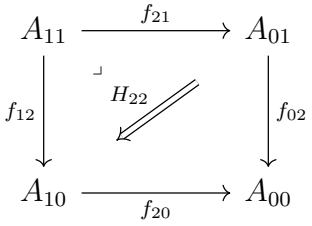
6.1 Pullback

Definition 6.1 (Gap Map)**.** For every commuting square, we get a canonical map (f_{21}, f_{12}, H_{22}) from the body of the span to the pullback of the cospan.



With $\pi_1 \circ (f_{21}, f_{12}, H_{22}) \equiv f_{21}$, $\pi_2 \circ (f_{21}, f_{12}, H_{22}) \equiv f_{12}$, and $\pi_3^r \circ (f_{21}, f_{12}, H_{22}) \equiv H_{22}^r$.

Definition 6.2 (Pullback Square)**.** A pullback square is a commuting square whose gap map is an equivalence.



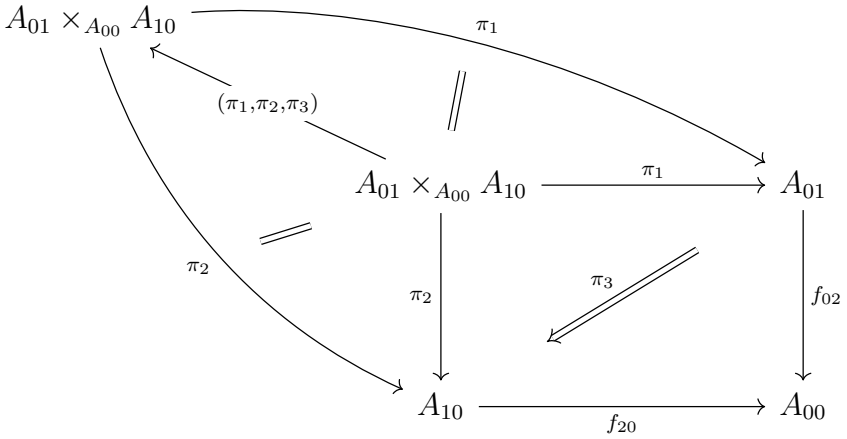
The data of a pullback square consists of:

- the data of a commuting square;
- maps $S, R : A_{01} \times_{A_{00}} A_{10} \to A_{11}$;
- homotopies $P^r : A_{01} \times_{A_{00}} A_{10} \to A_{01} \times_{A_{00}} A_{10}$, $Q^r : A_{11} \to A_{11}$, such that

$$P^{\mathsf{i0}} \equiv \mathrm{id}_{A_{01} \times_{A_{00}} A_{10}} \qquad P^{\mathsf{i1}} \equiv (f_{21}, f_{12}, H_{22}) \circ S$$
$$Q^{\mathsf{i0}} \equiv \mathrm{id}_{A_{11}} \qquad Q^{\mathsf{i1}} \equiv R \circ (f_{21}, f_{12}, H_{22})$$

This definition agrees with the usual definition in terms of the universal property of pullback (Rijke 2019, Proposition 1.4.10), but will be more convenient for our use. The idea is that the pullback type has a strict universal property, and anything that is equivalent to it will have the same universal property weakly.

Example 6.3 (Canonical Pullback Square)**.** For any cospan, we get a canonical commuting square by taking the pullback.

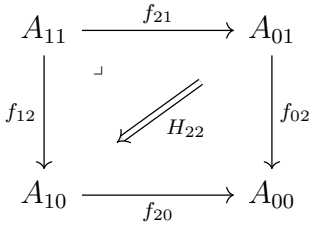


In this case, the gap map is $(\pi_1, \pi_2, \pi_3) \equiv \mathrm{id}_{A_{01} \times_{A_{00}} A_{10}}$; so, we can let

$$S \equiv R \equiv P^r \equiv Q^r \equiv \mathrm{id}_{A_{01} \times_{A_{00}} A_{10}}$$

to give the canonical pullback the structure of a pullback square.

Theorem 6.4 (Pullback of Equivalence is Equivalence)**.** *Given a pullback square*



If f_{20} is an equivalence, then f_{21} is an equivalence.

Proof. By, Lemma 3.6, we may assume that f_{20} is a half-adjoint equivalence. Then, the data we are given is that of a pullback square, as in Definition 6.2, together with the following, exhibiting that f_{20} is a half-adjoint equivalence.

$$g_{20} : A_{00} \to A_{10} \qquad M^r_{20} : A_{00} \to A_{00} \qquad N^r_{20} : A_{10} \to A_{10}$$

$$M^{\mathsf{i0}}_{20} \equiv \mathrm{id}_{A_{00}} \qquad M^{\mathsf{i1}}_{20} \equiv f_{20} \circ g_{20} \qquad N^{\mathsf{i0}}_{20} \equiv \mathrm{id}_{A_{10}} \qquad N^{\mathsf{i1}}_{20} \equiv g_{20} \circ f_{20}$$

$$\tau^{r,s}_{20} : A_{10} \to A_{00} \qquad \tau^{\mathsf{i0},s}_{20} \equiv f_{20} \qquad \tau^{\mathsf{i1},s}_{20} \equiv f_{20} \circ g_{20} \circ f_{20}$$

$$\tau^{r,\mathsf{i0}}_{20} \equiv f_{20} \circ N^r_{20} \qquad \tau^{r,\mathsf{i1}}_{20} \equiv M^r_{20} \circ f_{20}$$

We must implement the following data to make f_{21} an equivalence.

$$g_{21}, h_{21} : A_{11} \to A_{01} \qquad M^r_{21} : A_{01} \to A_{01} \qquad N^r_{21} : A_{11} \to A_{11}$$

$$M^{\mathsf{i0}}_{21} \equiv \mathrm{id}_{A_{01}} \qquad M^{\mathsf{i1}}_{21} \equiv f_{21} \circ g_{21} \qquad N^{\mathsf{i0}}_{21} \equiv \mathrm{id}_{A_{11}} \qquad N^{\mathsf{i1}}_{21} \equiv h_{21} \circ f_{21}$$

We let

$$\begin{aligned}
g_{21}(a) &\equiv S(a, g_{20}(f_{02}(a)), x \mapsto M^x_{20}(f_{02}(a))) \\
h_{21}(a) &\equiv R(a, g_{20}(f_{02}(a)), \\
&\qquad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{00}}(\\
&\qquad\qquad x = \mathsf{i0} \mapsto f_{02}(a), \\
&\qquad\qquad x = \mathsf{i1} \mapsto M^z_{20}(f_{02}(a)), \\
&\qquad)(f_{02}(a)))
\end{aligned}$$

Then

$$\begin{aligned}
f_{21}(g_{21}(a)) &\equiv f_{21}(S(a, g_{20}(f_{02}(a)), x \mapsto M^x_{20}(f_{02}(a)))) \\
&\equiv (\pi_1 \circ (f_{21}, f_{12}, H_{22}) \circ S)((a, g_{20}(f_{02}(a)), x \mapsto M^x_{20}(f_{02}(a)))) \\
&\rightsquigarrow \pi_1((a, g_{20}(f_{02}(a)), x \mapsto M^x_{20}(f_{02}(a)))) \\
&\equiv a
\end{aligned}$$

So we can have $M^r_{21}(a) \equiv \pi_1(P^r(g_{21}(a)))$.

Also

$$\begin{aligned}
h_{21}(f_{21}(a)) &\equiv R(f_{21}(a), g_{20}(f_{02}(f_{21}(a))), \\
&\qquad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{00}}(\\
&\qquad\qquad x = \mathsf{i0} \mapsto f_{02}(f_{21}(a)), \\
&\qquad\qquad x = \mathsf{i1} \mapsto M^z_{20}(f_{02}(f_{21}(a))), \\
&\qquad)(f_{02}(f_{21}(a)))) \\
&\rightsquigarrow R(f_{21}(a), g_{20}(f_{20}(f_{12}(a))), \\
&\qquad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{00}}(\\
&\qquad\qquad x = \mathsf{i0} \mapsto f_{02}(f_{21}(a)), \\
&\qquad\qquad x = \mathsf{i1} \mapsto M^z_{20}(f_{20}(f_{12}(a))), \\
&\qquad)(H^x_{22}(a))) \\
&\rightsquigarrow R(f_{21}(a), g_{20}(f_{20}(f_{12}(a))), \\
&\qquad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{00}}(\\
&\qquad\qquad x = \mathsf{i0} \mapsto f_{02}(f_{21}(a)), \\
&\qquad\qquad x = \mathsf{i1} \mapsto f_{20}(N^z_{20}(f_{12}(a))), \\
&\qquad)(H^x_{22}(a))) \\
&\rightsquigarrow R(f_{21}(a), f_{12}(a), x \mapsto H^x_{22}(a)) \\
&\equiv (R \circ (f_{21}, f_{12}, H_{22}))(a) \\
&\rightsquigarrow a
\end{aligned}$$

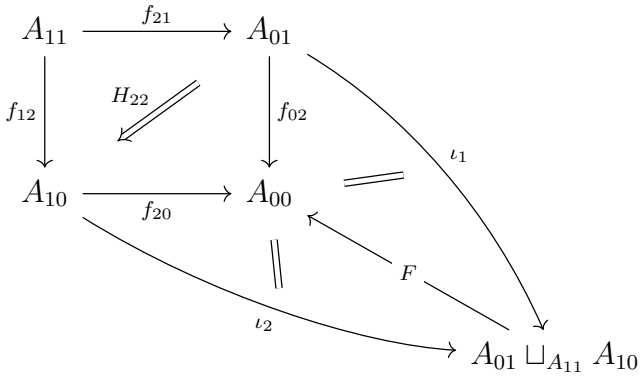
So

$$
\begin{aligned}
N^r_{21}(a) \equiv\ & \mathtt{com}^{y':\mathsf{i0}\to\mathsf{i1}}_{A_{11}}(\\
& \quad r = \mathsf{i0} \mapsto a, \\
& \quad r = \mathsf{i1} \mapsto R(f_{21}(a), g_{20}(H^{\neg y'}_{22}(a)), \\
& \qquad\qquad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{00}}(\\
& \qquad\qquad\quad x = \mathsf{i0} \mapsto f_{02}(f_{21}(a)), \\
& \qquad\qquad\quad x = \mathsf{i1} \mapsto M^z_{20}(H^{\neg y'}_{22}(a)), \\
& \qquad\qquad)(H^{x\wedge\neg y'}_{22}(a))), \\
&)(\mathtt{com}^{y:\mathsf{i0}\to\mathsf{i1}}_{A_{11}}(\\
& \quad r = \mathsf{i0} \mapsto Q^{\neg y}(a), \\
& \quad r = \mathsf{i1} \mapsto R(f_{21}(a), g_{20}(f_{20}(f_{12}(a)))), \\
& \qquad\qquad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{00}}(\\
& \qquad\qquad\quad x = \mathsf{i0} \mapsto f_{02}(f_{21}(a)), \\
& \qquad\qquad\quad x = \mathsf{i1} \mapsto \tau^{z,y}_{20}(f_{12}(a)), \\
& \qquad\qquad)(H^x_{22}(a))), \\
&)(R(f_{21}(a), N^r_{20}(f_{12}(a)), \\
& \quad x \mapsto \mathtt{com}^{z:\mathsf{i0}\to r}_{A_{00}}(\\
& \qquad\quad x = \mathsf{i0} \mapsto f_{02}(f_{21}(a)), \\
& \qquad\quad x = \mathsf{i1} \mapsto f_{20}(N^z_{20}(f_{12}(a))), \\
& \qquad)(H^x_{22}(a))))) \\
\end{aligned}
$$

□

6.2 Pushout

Definition 6.5 (Co-Gap Map)**.** For every commuting square, we get a canonical map F from the the pushout of the span to the body of the cospan.

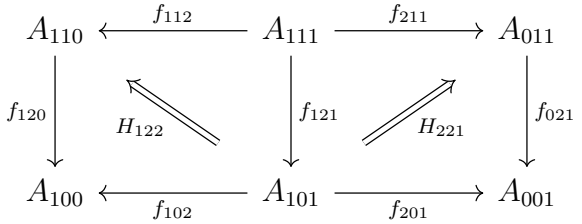


With F given by

$$\begin{aligned} F(p) \equiv \texttt{match}\ p\ \{ & \\ & \iota_1(a) \mapsto f_{02}(a), \\ & \iota_1(b) \mapsto f_{20}(b), \\ & \iota_3^x(c) \mapsto H_{22}^x(c), \\ \} & \end{aligned}$$

This has $F \circ \iota_1 \equiv f_{02}$, $F \circ \iota_2 \equiv f_{20}$, and $F \circ \iota_3^r \equiv H_{22}^r$.

Definition 6.6 (Functoriality of Pushout)**.** For any morphism of spans,

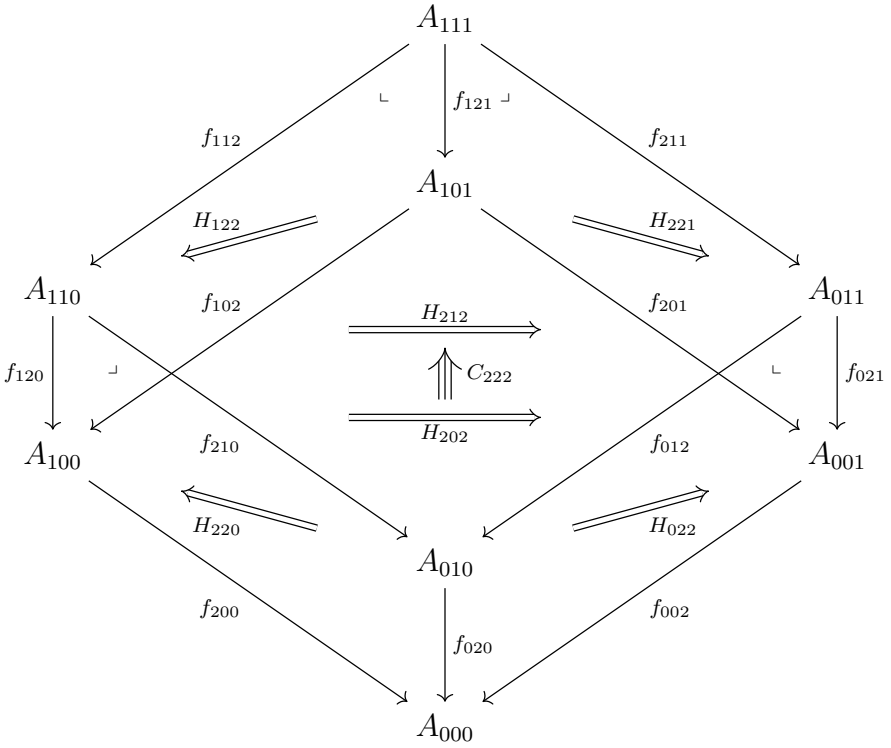


we get a canonical morphism $F : A_{110} \sqcup_{A_{111}} A_{011} \to A_{100} \sqcup_{A_{101}} A_{001}$ given by

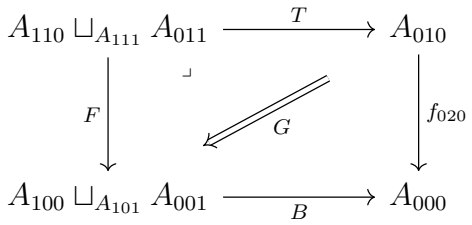
$$\begin{aligned} F(p) \equiv \texttt{match}\ p\ \{ & \\ & \iota_1(a) \mapsto \iota_1(f_{120}(a)), \\ & \iota_2(b) \mapsto \iota_2(f_{021}(b)), \\ & \iota_3^x(c) \mapsto \texttt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{100}\sqcup_{A_{101}}A_{001}}(\\ & \qquad x = \mathsf{i0} \mapsto \iota_1(H_{122}^z(c)), \\ & \qquad x = \mathsf{i1} \mapsto \iota_2(H_{221}^z(c)), \\ & \qquad)(\iota_3^x(f_{121}(c))), \\ \} & \end{aligned}$$

6.3 Cube

Theorem 6.7 (Cube Gives Pullback of Pushouts (Riley, Finster and Licata 2021, Theorem 4.5) (Rijke 2019, Theorem 2.2.12))**.** *Suppose we have a commuting cube with vertical faces pullback squares.*



Then we get a pullback square.



This is a theorem in the setting of (Rijke 2019), and we would like to use it to derive stability of our category of types, but we cannot prove it in StabTT as it is currently.

We are given the data of a commuting cube, as in Definition 3.10, together with data exhibiting the vertical squares as pullback squares, as in Definition 6.2.

The maps T, B on the top and bottom of the pullback square we are trying to construct are the co-gap maps of the top and bottom faces of the cube, as in Definition 6.5.

These are:

$$T(p) \equiv \texttt{match}\ p\ \{ \quad \iota_1(a) \mapsto f_{210}(a), \quad \iota_2(b) \mapsto f_{012}(b), \quad \iota_3^x(c) \mapsto H_{212}^x(c), \quad \}$$

$$B(p) \equiv \texttt{match}\ p\ \{ \quad \iota_1(a) \mapsto f_{200}(a), \quad \iota_2(b) \mapsto f_{002}(b), \quad \iota_3^x(c) \mapsto H_{202}^x(c), \quad \}$$

The map F on the left is the canonical map from Definition 6.6:

$$\begin{aligned} F(p) \equiv \texttt{match}\ p\ \{ & \\ & \iota_1(a) \mapsto \iota_1(f_{120}(a)), \\ & \iota_2(b) \mapsto \iota_2(f_{021}(b)), \\ & \iota_3^x(c) \mapsto \texttt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A_{100}\sqcup_{A_{101}}A_{001}}(\\ & \qquad x = \mathsf{i0} \mapsto \iota_1(H_{122}^z(c)), \\ & \qquad x = \mathsf{i1} \mapsto \iota_2(H_{221}^z(c)), \\ & \qquad)(\iota_3^x(f_{121}(c))), \\ \} & \end{aligned}$$

Now, we can define $G^s : A_{110} \sqcup_{A_{111}} A_{011} \to A_{000}$ by

$$\begin{aligned} G^s(p) \equiv \texttt{match}\ p\ \{ & \\ & \iota_1(a) \mapsto H_{220}^s(a), \\ & \iota_2(b) \mapsto H_{022}^s(b), \\ & \iota_3^r(c) \mapsto \texttt{com}^{y:\mathsf{i1}\to s}_{A_{000}}(\\ & \qquad s = \mathsf{i0} \mapsto \texttt{com}^{z:\mathsf{i0}\to y}_{A_{000}}(\\ & \qquad\qquad r = \mathsf{i0} \mapsto H_{220}^z(f_{112}(c)), \\ & \qquad\qquad r = \mathsf{i1} \mapsto H_{022}^z(f_{211}(c)), \\ & \qquad\qquad)(f_{020}(H_{212}^r(c))), \\ & \qquad r = \mathsf{i0} \mapsto H_{220}^y(f_{112}(c)), \\ & \qquad r = \mathsf{i1} \mapsto H_{022}^y(f_{211}(c)), \\ & \qquad)(C^{r,s}(c)), \\ & p'\ \texttt{if}\ s = \mathsf{i0} \mapsto f_{020}(T(p')), \\ & p'\ \texttt{if}\ s = \mathsf{i1} \mapsto B(F(p')), \\ \} & \end{aligned}$$

Then we've constructed the commuting square. To promote it to a pullback square, we must define a section and a retraction

$$S, R : A_{010} \times_{A_{000}} (A_{100} \sqcup_{A_{101}} A_{001}) \to A_{110} \sqcup_{A_{111}} A_{011}$$

of the gap map (T, F, G). We'll try to construct S and see where we need a stronger eliminator for pushout than we have given ourselves.

Take $p : A_{010} \times_{A_{000}} (A_{100} \sqcup_{A_{101}} A_{001})$. Then we have

$$\pi_1(p) : A_{010} \qquad \pi_2(p) : A_{100} \sqcup_{A_{101}} A_{001} \qquad \pi_3^r(p) : A_{000}$$

$$\pi_3^{\mathsf{i0}}(p) \equiv f_{020}(\pi_1(p)) \qquad \pi_3^{\mathsf{i1}}(p) \equiv B(\pi_2(p))$$

We need to make a term of type $A_{110} \sqcup_{A_{111}} A_{011}$, but we don't know what constructor to use; so we need to proceed by cases. The only candidate to match on is $\pi_2(p) : A_{100} \sqcup_{A_{101}} A_{001}$.

Consider the case that $\pi_2(p)$ is $\iota_1(a)$. Now we have $\pi_1(p) : A_{010}$ and $a : A_{100}$. Then we'd like to make a term of type $A_{010} \times_{A_{000}} A_{100}$ and apply S_{220} to get a term of type A_{110}, whereupon we can apply ι_1 to return $A_{110} \sqcup_{A_{111}} A_{011}$.

We have the first and second components of our term of type $A_{010} \times_{A_{000}} A_{100}$, and we just need a path between their images. This path is in A_{000}, so we turn to our path $\pi_3^r(p) : A_{000}$.

We have $\pi_3^{\mathsf{i0}}(p) \equiv f_{020}(\pi_1(p))$, as we need. For the other endpoint, it seems as though we have $\pi_3^{\mathsf{i1}}(p) \equiv B(\pi_2(p)) \equiv B(\iota_1(a)) \equiv f_{200}(a)$, as required. But if we try to define S by

$$\begin{aligned} S(p) \equiv \mathtt{match}\ \pi_2(p)\ \{ & \\ & \iota_1(a) \mapsto S_{220}((\pi_1(p), a, x \mapsto \pi_3^x(p))), \\ & \vdots \\ \} & \end{aligned}$$

we see that this is not allowed by our $\sqcup$-ELIM rule, since we cannot mention p while writing the cases.

Even if we permit ourselves to use variables from Γ, we do not have $\pi_2(p) \equiv \iota_1(a)$ when we are writing the ι_1 case, since the hypothesis $\Xi|\Phi|\Gamma, a : A \vdash s : Z$ does not mention the scrutinee.

This problem could be circumvented in a dependent type theory by passing the path as an argument to a function that we construct by induction on $A_{100} \sqcup_{A_{101}} A_{001}$. The motive of our induction will be the type family

$$\begin{aligned} p : A_{010} \times_{A_{000}} (A_{100} \sqcup_{A_{101}} A_{001}), q : A_{100} \sqcup_{A_{101}} A_{001} & \\ \vdash \mathrm{Path}_{A_{000}}(f_{020}(\pi_1(p)), B(q)) \to A_{110} \sqcup_{A_{111}} A_{011}\ \mathtt{type} & \end{aligned}$$

Then we could write

$$\begin{aligned} S(p) \equiv \mathtt{match}\ \pi_2(p)\ \{ & \\ & \iota_1(a) \mapsto \lambda m. S_{220}((\pi_1(p), a, m)), \\ & \vdots \\ \}(\pi_3(p)) & \end{aligned}$$

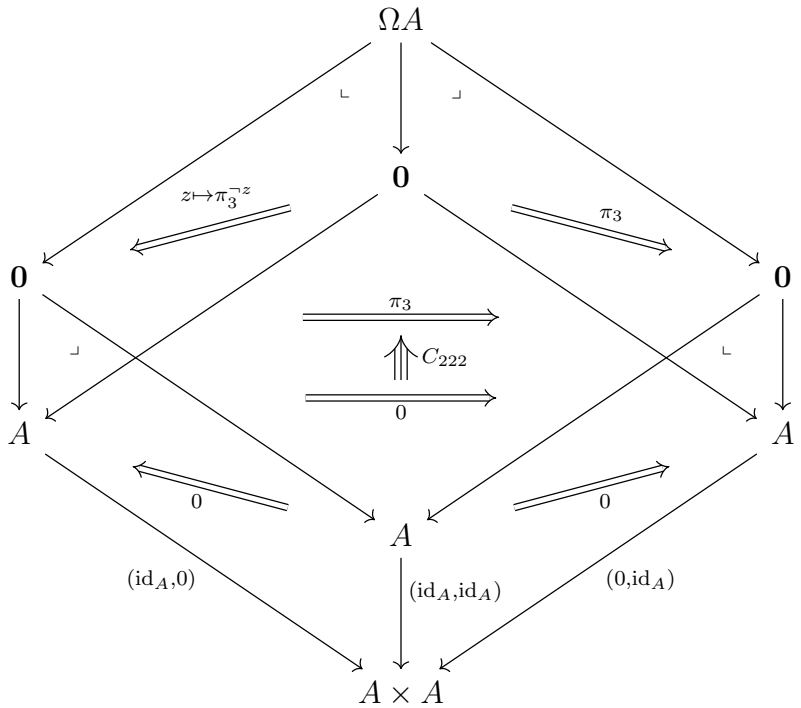
This solution is not available to us since we have neither function types nor dependent types.

A simple approach to fixing this problem without dependent types is to allow ourselves to hypothesise judgmental equalities. Then the ⊔-ELIM rule would have a hypothesis resembling $\Xi|\Phi|\Gamma, a : A|\Theta, (p \equiv \iota_1(a)) \vdash s : Z$. This does not seem to be a promising approach. First, hypothesising judgmental equalities is liable to make type-checking undecidable. Second, we may need to hypothesise judgmental equalities that depend on additional interval variables, and that only hold when an equation involving these variables holds; this would be very complicated. Third, the semantics for adding a judgmental equality to the context would be something like taking the strict equaliser. The ⊔-ELIM rule would then assert that pushout is well-behaved with respect to strict limits, whereas it seems more appropriate to refer to homotopy limits.

We would like a ⊔-ELIM rule that expresses that pushouts in a stable ∞-category are stable under pullback, while still computing on the free $\mathbb{E}_\infty$ operations. With this, we should be able to prove Theorem 6.7 in StabTT.

6.4 Putting It All Together

Now, assuming a version of StabTT where Theorem 6.7 holds, we can proceed as in (Riley, Finster and Licata 2021, Section 4). Consider the following cube.



Where $C^{r,s}_{222} : \Omega A \to A \times A$ is given by

$$\begin{aligned} C^{r,s}_{222}(p) \equiv \mathtt{com}^{z:\mathsf{i0}\to\mathsf{i1}}_{A\times A}(& \\ & r = \mathsf{i0} \mapsto (\pi_3^{s\wedge\neg z}(p), 0), \\ & r = \mathsf{i1} \mapsto (0, \pi_3^{z\vee\neg s}(p)), \\ &)((\pi_3^{r\vee s}(p), \pi_3^{r\wedge\neg s}(p))) \end{aligned}$$

The idea behind $C^{r,s}_{222}$ is that traversing the same loop in each component of $A \times A$ consecutively is equivalent to traversing them concurrently.

Applying Theorem 6.7, we get a pullback square

$$\begin{array}{ccc} \Sigma\Omega A & \xrightarrow{\varepsilon} & A \\ {\scriptstyle F}\downarrow & \lrcorner \;\swarrow G & \downarrow{\scriptstyle (\mathrm{id}_A,\mathrm{id}_A)} \\ A + A & \xrightarrow[\psi]{} & A \times A \end{array}$$

We know that ψ is an equivalence; so by Theorem 6.4, ε is also an equivalence.

7 Future Work

7.1 Elimination Rule for Pushout

To complete the proof of Theorem 6.7, we need an elimination rule for the pushout type that ensures that pushouts are stable under pullback, but this must not come at the cost of being able to compute `match`'s on terms returned by the free $\mathbb{E}_\infty$ operations.

7.2 Unit Map

Once we've proven Theorem 6.7, we will have shown that the co-unit ε of the adjunction between Σ and Ω is a natural equivalence. Then, for stability, it will remain to prove that the unit η is also a natural equivalence.

One approach to this would be to dualise our argument for the co-unit. This may fail, however, because there is not complete symmetry between inputs and outputs in StabTT, since our contexts have multiple variables; yet we only produce a single term.

Another approach would be to continue to follow (Riley, Finster and Licata 2021). They make use of the Little Blakers-Massey Theorem (Anel et al. 2020, Corollary 4.1.4); so this approach would require proving it in StabTT. The Little Blakers-Massey Theorem states that, given a pushout square, if a certain map is an equivalence, then the square is also a pullback square. In (Riley, Finster and Licata 2021),

they use the existence of biproducts to show that the map is always an equivalence and, therefore, that every pushout square is a pullback square. Then, for any type A, we have

$$\begin{array}{ccc} A & \longrightarrow & \mathbf{0} \\ \downarrow & \lrcorner & \downarrow \\ \mathbf{0} & \longrightarrow & \Sigma A \end{array}$$

which is a pullback square since it is a pushout square. But also, we have a pullback square

$$\begin{array}{ccc} \Omega\Sigma A & \longrightarrow & \mathbf{0} \\ \downarrow & \lrcorner & \downarrow \\ \mathbf{0} & \longrightarrow & \Sigma A \end{array}$$

Then, by uniqueness of pullbacks, $\Omega\Sigma A \cong A$; so we have stability.

Conjecture 7.1 (Stability)**.** *At least the second of these approaches will work. Thus, the category of types is stable.*

7.3 Internal Language

The intended semantics of StabTT is stable ∞-categories. If Conjecture 7.1 holds, it remains to verify that our definition of stability genuinely corresponds to stability of the ∞-categories that model the type theory. We would also like the reverse implication: that every stable ∞-category is a model of StabTT in an essentially unique way.

Conjecture 7.2 (Internal Language for Stable ∞-Categories)**.** *StabTT acts as the internal language for stable ∞-categories.*

7.4 Smash Product

The smash product of spectra is important in higher algebra, since we use it to define $\mathbb{E}_\infty$-rings. In (Riley 2022), Riley extends the type theory of (Riley, Finster and Licata 2021) with linear type-formers, including a dependent form of the smash product.

The same could be done with StabTT. This would likely require a more complex context structure, since we would have two product operations: cartesian product, and smash product. This could look like bunched type theory (Pym 2002), or it could follow (Riley 2022) and have an ordinary context together with a 'palette' keeping track of the linearity requirements.

7.5 Indexing over a Base

Our original motivation to study spectra was algebraic topology, but in the type theory we ended up with there are only spectra and no spaces to compute algebraic invariants of.

We could define an *indexed type theory* (Isaev 2021) whose base type theory is ordinary cubical type theory, and whose indexed type theory is StabTT, with the types in the two levels interpreted as spaces and spectra, respectively. Then we could define type-formers $\Sigma^\infty, \Omega^\infty$ that form the suspension spectrum of a pointed space, and the underlying pointed space of a spectrum.

Then, following (Riley 2022, Definition 4.2.1), we could define the homology and cohomology of a pointed space X with coefficients in a spectrum E by

$$E_n(X) \equiv \pi_n(\Omega^\infty(\Sigma^\infty(X) \otimes E))$$
$$E^n(X) \equiv \pi_n(\Omega^\infty(\Sigma^\infty(X) \multimap E))$$

7.6 Higher Coherators

Part of our motivation for working synthetically was to avoid the combinatorics associated with explicit descriptions of ∞-categorical entities; so it is unsatisfying that our syntax has infinitely-many term-formers for the various coherators of an $\mathbb{E}_\infty$-space.

The proof of stability is finite; so it only uses finitely-many of these coherators. We may question, therefore, why the higher coherators are necessary? The higher coherators appear in the computation rules for the lower coherators; so if we removed them outright, we would end up with the stuck terms we sought to eliminate.

That said, the higher coherators would still be derivable, since we can relate the addition in Ω types to concatenation of paths, which is completely coherent because of Kan composition, and, given stability, every type A is equivalent to $\Omega\Sigma A$. Possibly we could remove most of the higher coherators, and use these derivable coherators in the computation rules that would have broken, but this might lead to non-terminating computations.

Kan composition provides complete coherence of concatenation of paths with a single operation. We can even use Kan composition to prove distributivity of any operation over Kan composition. Given $f : A \to B$, and the data of a Kan composition in A, we can define a path $\delta^s_{\mathtt{com}} : B$ by

$$\begin{aligned}
\delta^s_{\mathtt{com}} \equiv \mathtt{com}_B^{z':r\to r'}(& \\
& \phi \mapsto f(t[z'/z]), \\
& s = \mathsf{i0} \mapsto \mathtt{com}_B^{z:r\to z'}(\phi \mapsto f(t))(f(b)), \\
& s = \mathsf{i1} \mapsto f(\mathtt{com}_A^{z:r\to z'}(\phi \mapsto t)(b)), \\
)(f(b)) &
\end{aligned}$$

which has

$$\delta^{\mathsf{i0}}_{\mathtt{com}} \equiv \mathtt{com}^{z:r\to r'}_{B}(\phi \mapsto f(t))(f(b))$$
$$\delta^{\mathsf{i1}}_{\mathtt{com}} \equiv f(\mathtt{com}^{z:r\to r'}_{A}(\phi \mapsto t)(b))$$

and, throughout, if ϕ holds, $\delta^{s}_{\mathtt{com}} \equiv f(t[r'/z])$, and if $r = r'$, $\delta^{s}_{\mathtt{com}} \equiv f(b)$. Perhaps there is some operation akin to Kan composition that unifies all the higher coherators including the distributors; then we would only need rules for this single coherator.

7.7 Operations on Type Families

Another approach to enrichment is to define the operations for type families rather than individual types. This fits with the semantics of type families as morphisms and terms as sections, since post-composition by these morphisms would preserve the operations. This would also strengthen the analogy with Kan composition, which is defined for type families.

Bibliography

Anel, Mathieu et al. (2020). 'A generalized Blakers–Massey theorem'. In: *Journal of Topology* 13.4, pp. 1521–1553. DOI: https://doi.org/10.1112/topo.12163. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/topo.12163. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/topo.12163.

Angiuli, Carlo, Guillaume Brunerie et al. (2021). 'Syntax and models of Cartesian cubical type theory'. In: *Mathematical Structures in Computer Science* 31.4, pp. 424–468. DOI: 10.1017/S0960129521000347.

Angiuli, Carlo and Daniel Gratzer (2024). 'Principles of Dependent Type Theory'. URL: https://www.danielgratzer.com/papers/type-theory-book.pdf (visited on 26th Nov. 2024).

Berg, Benno van den and Richard Garner (2011). 'Types are weak ω-groupoids'. In: *Proceedings of the London Mathematical Society* 102.2, pp. 370–394. DOI: https://doi.org/10.1112/plms/pdq026. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/pdq026. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/plms/pdq026.

Bezem, Marc, Thierry Coquand and Simon Huber (2014). 'A Model of Type Theory in Cubical Sets'. In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*. Ed. by Ralph Matthes and Aleksy Schubert. Vol. 26. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 107–128. ISBN: 978-3-939897-72-9. DOI: 10.4230/LIPIcs.TYPES.2013.107. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2013.107.

Cohen, Cyril et al. (2018). 'Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom'. In: *21st International Conference on Types for Proofs and Programs (TYPES 2015)*. Ed. by Tarmo Uustalu. Vol. 69. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 5:1–5:34. ISBN: 978-3-95977-030-9. DOI: 10.4230/LIPIcs.TYPES.2015.5. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2015.5.

Deflorin, Gian (2019). 'The Homotopy Hypothesis'. MA thesis. University of Zurich.

Gepner, David, Moritz Groth and Thomas Nikolaus (Dec. 2015). 'Universality of multiplicative infinite loop space machines'. In: *Algebraic & Geometric Topology* 15.6, pp. 3107–3153. ISSN: 1472-2747. DOI: 10.2140/agt.2015.15.3107. URL: http://dx.doi.org/10.2140/agt.2015.15.3107.

Huber, Simon (Aug. 2019). 'Canonicity for Cubical Type Theory'. In: *J. Autom. Reason.* 63.2, pp. 173–210. ISSN: 0168-7433. DOI: 10.1007/s10817-018-9469-1. URL: https://doi.org/10.1007/s10817-018-9469-1.

Isaev, Valery (2021). 'Indexed type theories'. In: *Mathematical Structures in Computer Science* 31.1, pp. 3–63. DOI: 10.1017/S0960129520000092.

Kelly, G. (Jan. 2005). 'The Basic Concepts of Enriched Category Theory'. In: *Reprints in Theory and Applications of Categories [electronic only]* 2005.

Ker, Andrew (2009). 'Lambda Calculus'. In.

Klabnik, Steve and Carol Nichols (2022). *The Rust Programming Language*.

Lurie, Jacob (2009). *Higher Topos Theory*. Princeton: Princeton University Press. ISBN: 9781400830558. DOI: doi:10.1515/9781400830558. URL: https://doi.org/10.1515/9781400830558.

– (2017). *Higher Algebra*. Harvard University.

Martin-Löf, Per (1975). 'An Intuitionistic Theory of Types: Predicative Part'. In: *Logic Colloquium '73*. Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. Elsevier, pp. 73–118. DOI: https://doi.org/10.1016/S0049-237X(08)71945-1. URL: https://www.sciencedirect.com/science/article/pii/S0049237X08719451.

Pym, D J (2002). *The Semantics and Proof Theory of the Logic of Bunched Implications*. English. Applied Logic Series. Netherlands: Kluwer Academic Publishers. ISBN: 1402007450.

Riehl, Emily and Michael Shulman (2023). *A type theory for synthetic ∞-categories*. arXiv: 1705.07442 [math.CT]. URL: https://arxiv.org/abs/1705.07442.

Rijke, Egbert (2019). *Classifying Types*. arXiv: 1906.09435 [math.LO]. URL: https://arxiv.org/abs/1906.09435.

Riley, Mitchell (2022). 'A Bunched Homotopy Type Theory for Synthetic Stable Homotopy Theory'. PhD thesis. Wesleyan University.

Riley, Mitchell, Eric Finster and Daniel R. Licata (2021). *Synthetic Spectra via a Monadic and Comonadic Modality*. arXiv: 2102.04099 [math.CT]. URL: https://arxiv.org/abs/2102.04099.

Univalent Foundations Program, The (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book.

Warren, M (2008). 'Homotopy Theoretic Aspects of Constructive Type Theory'. In: URL: https://api.semanticscholar.org/CorpusID:123575477.