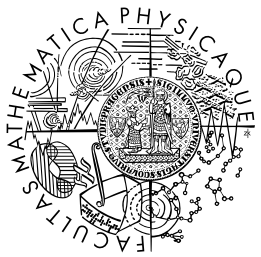


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Vojtěch Štěpančík

Formalization of Homotopy Pushouts in Homotopy Type Theory

Department of Algebra

Supervisor of the master thesis: doctor Egbert Rijke
Study programme: Mathematical Structures
Study branch: MSPN

Prague 2024

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

This thesis would not be what it is today if Egbert Rijke hadn't responded to my cold email, in which I asked if he was willing to supervise my project. I would like to express my thanks to him, not only for the positive response, but also for his guidance on my journey to understanding and formalization of synthetic homotopy theory, and for the work he has done for the Homotopy Type Theory community at large, through his written material and the agda-unimath library, which he initiated with his colleagues and actively maintains. I'm happy to be a part of it.

I would like to thank Fredrik Bakke, for our conversations both academic and personal. The office feels a lot emptier without him sharing his observations about simplicial type theory. I also appreciated his code review of my contributions to the library.

I am grateful to my family for being very excited about my work, even though I repeatedly failed to properly explain what I was doing, by nobody's fault but my own, because I often lacked the necessary understanding myself. It motivated me to try and find more intuitive descriptions of the code I was writing; hopefully some of it is reflected in the thesis.

I also want to thank my colleague and friend Max Hollmann, for often sitting down with me at a café after a long day's work, and listening to my tired rambling about commuting prisms. He and the rest of our friend group are a big reason why I remained motivated to keep working on my master's studies. I also appreciate the staff at Cafe Prostoru_, where much of the research and formalization for this thesis was done, for tolerating my everyday gloomy presence, with my face buried in my laptop, and for lifting my spirits with pretty latte art on my Chai Latté.

Finally, I would like to thank Matěj Dostál, who first taught me logic and type theory, and pointed me in the direction of Homotopy Type Theory in my formative years as an academic.

Vojtěch Štěpančík, Ljubljana

Title: Formalization of Homotopy Pushouts in Homotopy Type Theory

Author: Vojtěch Štěpančík

Department: Department of Algebra

Supervisor: doctor Egbert Rijke

Abstract: Homotopy pushouts can be constructed as higher inductive types in the logical framework of Homotopy Type Theory, where one may engage syntactic methods to explore their properties, and formalize them in a proof assistant. This thesis focuses on the descent property, due to Rijke [12], which characterizes type families over pushouts; the flattening lemma, due to Brunerie [5], which characterizes the total spaces of such families; and the universal property of identity types of pushouts, due to Kraus and von Raumer [7]. We also build elementary infrastructure for sequential colimits, following a paper of Sojakova, van Doorn, and Rijke [17]. We then use the built machinery to provide a partial formalized proof of Wärn's zigzag construction of identity types of pushouts as sequential colimits [19], leaving one coherence problem open. The thesis was simultaneously formalized in the proof assistant Agda [2] and results contributed to the agda-unimath library [15].

Keywords: synthetic homotopy theory, homotopy type theory, univalent foundations of mathematics, formalization, homotopy pushouts

Název práce: Formalizace homotopických pushoutů v homotopické teorii typů

Autor: Vojtěch Štěpančík

Katedra: Katedra algebry

Vedoucí diplomové práce: doctor Egbert Rijke

Abstrakt: Homotopické pushouty mohou být zkonstruovány jako vyšší induktivní typy v logickém rámci Homotopické Teorie Typů, ve kterém lze použít syntaktické metody pro zkoumání jejich vlastností, a formalizovat je v důkazovém asistentu. Tato diplomová práce se zaměřuje na vlastnost sestupu, popsanou Rijkem [12], která charakterizuje rodiny typů nad pushouty; na lemma o zplošťování, popsané Bruneriem [5], které charakterizuje totální prostory takových rodin; a univerzální vlastnost typů identifikací v pushoutech, formulovanou Krausem a von Raumerem [7]. Vybudujeme také základní infrastrukturu pro práci se sekvenčními kolimitami, podle článku Sojákové, van Doorna a Rijkeho [17]. Vybudované nástroje posléze použijeme na částečný formalizovaný důkaz Wärnovy klikaté konstrukce typů identifikací v pushoutech jako sekvenčních kolimit [19], s jedním neuzavřeným problémem koherence. Práce byla postupně formalizována v důkazovém asistentu Agda [2], a výsledky přispěny do knihovny agda-unimath [15].

Klíčová slova: syntetická homotopická teorie, homotopická teorie typů, univalentní základy matematiky, formalizace, homotopické pushouty

Contents

Introduction

Homotopy Type Theory [18] is a logical framework built on Martin-Löf's intensional type theory [9] and the univalence axiom, which characterizes identity types of universes. It is inspired by the homotopy interpretation of dependent type theory [4], in which types are interpreted as spaces, elements of types as points in the spaces, and identifications of elements as paths between the points. Using proof assistants such as Agda [2], one can translate proofs in Homotopy Type Theory into programs in a programming language, engaging in an activity called "formalization" [10]. The validity of the constructions is verified by type-checking the programs.

This work focuses on homotopy pushouts — specifically we provide an exposition and formalization of the descent property [12] and flattening lemma [5], and use the built infrastructure to formally construct a partial proof of correctness of Wärn's zigzag construction of identity types of pushouts [19], leaving one coherence problem open. Efforts to finish the formalization will continue, in response to Wärn's statement, taken from another paper of his about the zigzag construction: "At the time of writing, no such formalization has been carried out, but we believe it would be feasible and worthwhile." [20]

On top of pushouts we construct sequential colimits. The infrastructure we build is more extensive than strictly necessary for the zigzag construction, since we anticipate it will be useful when formalizing its applications. The formalized material comes from a paper on treatment of sequential colimits in Homotopy Type Theory by Sojakova, van Doorn and Rijke [17]. As a byproduct we started an effort to collect pages for formalization of results from the literature in the agda-unimath library. Even though it is not of strictly mathematical nature, this initiative is relevant to the social aspect of formalized mathematics, as it builds more documentation, makes the development accessible, and sets an example for beginning formalizers.

An important part of the thesis is the reusable and documented formalization of the presented material in the agda-unimath library [15]. The specific code contributions are listed in Appendix A. The relevant proofs were "unformalized" into English and are presented below.

Organization

The thesis is divided into four chapters. Chapter 1 provides an abridged introduction to Homotopy Type Theory, presenting the necessary foundational results on which we build. Those results are taken either from the "HoTT Book" [18], or from Rijke's textbook [13] and the formalization in the agda-unimath library. Chapter 2 introduces pushouts as structures satisfying a universal prop-

erty, and describes some of their elementary properties, namely formation of type families over pushouts, a universal property of total spaces of type families over pushouts, and a universal property of identity systems of pushouts. Chapter 3 defines coequalizers and sequential colimits as other kinds of colimits, and shows how their existence and properties can be derived from pushouts. The focus of the chapter is primarily on sequential colimits, as the constructions are used in the succeeding Chapter 4. The latter describes an explicit construction claimed to be an identity system of pushouts, built up of sequential colimits of pushouts, due to Wärn [19]. It then proceeds to describe a formalized partial proof that the construction is indeed an identity system, which has been carried out for the thesis.

Contributions

Chapter 1 and first section of Chapter 2 consist of exposition to material that had already been formalized in the library. Formalization of all other parts of the thesis are original contributions to the library. The application of descent data and their sections to identity systems, the presented proof of the flattening lemma, and the partial proof of correctness of the zigzag construction are original research.

Chapter 1

Homotopy Type Theory

This section briefly introduces important concepts and results from Homotopy Type Theory, the formal setting of the thesis. For a more extensive account, see the "HoTT Book" [18], or Rijke's textbook [13].

In Homotopy Type Theory, the basic objects of study are *types*, which may have *elements*. An element a of type A is written $a : A$. In contrast to set theory, where an element can belong to many different sets, an element cannot be of multiple different types. Types themselves are elements of special types called *universes*, written $A : \mathcal{U}$. To avoid inconsistency [6], we assume an infinite hierarchy of universes, such that a universe $\mathcal{U}$ is an element of a bigger universe, $\mathcal{U} : \mathcal{U}^+$. We generally do not comment on the handling of universe levels in the text of the thesis, but the attached formalization is developed in the maximally universe-polymorphic way possible.

For any two types A and B, there is the type of **functions** $A \to B$, whose elements are functions taking elements of A to elements of B. For any type A there is the identity function $\mathrm{id} : A \to A$, and functions can be composed using the composition operator

$$- \circ - : (B \to C) \to (A \to B) \to (A \to C),$$

which is associative and unital with respect to id.

Elements of a function type into a universe are called **type families**: a type family P *over* A is a function $P : A \to \mathcal{U}$. The type $P(a)$ for some specific $a : A$ is occasionally referred to as the **fiber at** a.

Given a type family $P : A \to \mathcal{U}$ we may construct the **Sigma type**, or its **total space**, $\Sigma(a : A).\, P(a)$ (or ΣAP). Elements of the type ΣAP are **dependent pairs** (a, p) where a is an element of A and p is an element of $P(a)$.

We may also construct the **Pi type** $\Pi(a : A).\, P(a)$. We prefer the syntax $(a : A) \to P(a)$ for Π types. Elements of the type $(a : A) \to P(a)$ are **dependent functions** f which take an element $a : A$ to an element $f(a) : P(a)$. They are also referred to as **sections of** P. Regular function types $A \to B$ are Π types where the codomain is constant over A.

For two elements of the same type $x, y : A$ there is the type of **identifications** $\mathrm{Id}(x, y)$, usually written as $x = y$. The identity types are in a precise sense generated by the elements $\mathrm{refl} : x = x$ for every $x : A$. This fact is encoded in their elimination principle, known as the **based** J **rule** [11]. The J rule is also called the **induction principle of identifications** or **path induction**.

Theorem 1.0.1. *Consider a type A and its element $a_0 : A$. Given a type family $P : (a : A) \to (a_0 = a) \to \mathcal{U}$, there is a function*

$$J : P(a_0, \mathrm{refl}) \to ((a : A)(r : a_0 = a) \to P(a, r))$$

such that for any point $p_0 : P(a_0, \mathrm{refl})$ it computes as $J(p_0, a_0, \mathrm{refl}) \doteq p_0$.

We distinguish between identifications and the metatheoretical concept of **judgmental equality**. The elements x and y are *identified* if there is an element of the type $x = y$. They are *judgmentally equal*, denoted $x \doteq y$, if they compute to syntactically identical expressions.

Elements of identity types can themselves be identified — for all $p, q : x = y$, there is the type $p = q$, which also has its own identity type, and so on. The identity types endow all types with a structure of an ∞-groupoid — there are inversion and composition operations

$$(-)^{-1} : (x = y) \to (y = x)$$
$$- \bullet - : (x = y) \to (y = z) \to (x = y)$$

such that taking an inverse is an involution, and concatenation is associative and unital with respect to refl. Most of the just mentioned laws hold up to a higher identification, e.g. the elements $p \bullet \mathrm{refl}$ and p are not judgmentally equal, but there is an identification $\mathrm{runit} : (p \bullet \mathrm{refl}) = p$. In our setting only the left unit law is judgmental, i.e. there is a judgmental equality $(\mathrm{refl} \bullet p) \doteq p$.

Identifications of elements $x, y : A$ induce a map between fibers for any type family $P : A \to \mathcal{U}$. This is called the **transport map**

$$\mathrm{tr}_P : (x = y) \to P(x) \to P(y).$$

The transport map distributes over concatenation of identifications.

Lemma 1.0.2. *Given a type A, a type family $P : A \to \mathcal{U}$, elements $x, y, z : A$, identifications $p : x = y$ and $q : y = z$, and an element $u : P(x)$, there is an identification*

$$\mathrm{tr}_P(p \bullet q, u) = \mathrm{tr}_P(q, \mathrm{tr}_P(p, u)).$$

Transport in type families of certain shapes can be characterized. For example, transport in the type family $\mathrm{Id}(a_0)$ behaves like concatenation.

Lemma 1.0.3. *Given identifications $p : x = y$ and $q : a_0 = x$, there is an identification*

$$\mathrm{tr}_{\mathrm{Id}(a_0)}(p, q) = q \bullet p.$$

Transport in a family of function types behaves like composition.

Lemma 1.0.4. *Given a type A, type families $P : A \to \mathcal{U}$ and $Q : A \to \mathcal{V}$, and an identification $x = y$ in A, then for all $h : P(x) \to Q(x)$ there is an identification*

$$\mathrm{tr}_{\lambda a \to (P(a) \to Q(a))}(p, h) = (\mathrm{tr}_Q(p) \circ h \circ \mathrm{tr}_P(p^{-1})).$$

Regular and dependent functions preserve identifications. Consider a function $f : A \to B$, a dependent function $h : (a : A) \to P(a)$, and elements $x, y : A$. Preservation of identifications is expressed by the maps

$$\mathsf{ap}_f : (x = y) \to (fx = fy)$$
$$\mathsf{apd}_h : (p : x = y) \to \mathsf{tr}_P(p, hx) = hy.$$

Lemma 1.0.5. *Given a map $f : A \to B$, a type family $P : B \to \mathcal{U}$, elements $x, y : A$, $u : P(fx)$, and an identification $p : x = y$, there is an identification*

$$\mathsf{tr}_P(\mathsf{ap}_f(p), u) = \mathsf{tr}_{(P \circ f)}(p, u).$$

Functions with a shared domain and a shared codomain can be compared by a relation called *homotopy*. Given two maps $f, g : (a : A) \to P(a)$, a **homotopy** between them is an element of the type $(a : A) \to (f(a) = g(a))$, written as $f \sim g$. Since the type of homotopies is the type of pointwise identifications, the operations on identifications induce operations on homotopies, namely we have

$$(-)^{-1} : (f \sim g) \to (g \sim f)$$
$$- \bullet_h - : (f \sim g) \to (g \sim h) \to (f \sim h)$$

and the element refl-htpy $: f \sim f$ for all functions f.

Additionally, we introduce left and right **whiskering** operations, given maps $f, g : A \to B$, a homotopy $H : f \sim g$, and maps $i : X \to A$ and $j : B \to C$

$$j \cdot_l H := (\lambda x \to \mathsf{ap}_f(Hx)) : (j \circ f) \sim (j \circ g)$$
$$H \cdot_r i := (\lambda x \to H(ix)) \quad : (f \circ i) \sim (g \circ i).$$

A homotopy $H : f \sim g$ can be diagrammatically represented as a cell between parallel arrows:

$$A \underset{f}{\overset{g}{\Downarrow H}} B.$$

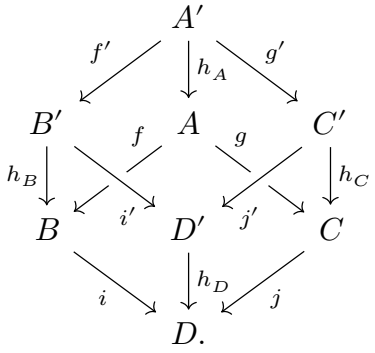
By convention, homotopies in diagrams go from bottom left to top right. We often mention more complex shapes: a **commuting triangle** of maps is a homotopy $h \sim (g \circ f)$, and a **commuting square** of maps is a homotopy $(i \circ j) \sim (k \circ l)$, pictured as

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ & {\scriptstyle h}\searrow \quad \swarrow{\scriptstyle g} & \\ & C & \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{l} & B \\ {\scriptstyle j}\downarrow & & \downarrow{\scriptstyle k} \\ C & \xrightarrow[i]{} & D. \end{array}$$

We will occasionally encounter higher shapes, e.g. a **commuting cube** is a homotopy of homotopies

$$\alpha : ((i \cdot_l BL) \bullet_h (FL \cdot_r f') \bullet_h (h_D \cdot_l T))$$
$$\sim ((B \cdot_r h_A) \bullet_h (j \cdot_l BR) \bullet_h (FR \cdot_r g')),$$

where BL, FL, T, B, BR and FR are homotopies which fit as the back left, front left, top, bottom, back right and front right faces, respectively, of the following cubical diagram:



Given a map $f : A \to B$, we call a converse map $s : B \to A$ its **section** if it comes equipped with a homotopy $(f \circ s) \sim \mathrm{id}$, and likewise we call a map $r : B \to A$ a **retraction** of f if there is a homotopy $(r \circ f) \sim \mathrm{id}$.

A function $f : A \to B$ is an **equivalence** if it is equipped with both a section and a retraction. This is different from the classical notion of being an isomorphism, i.e. having a single inverse $g : B \to A$ and proofs that g is a section and a retraction of f. Every isomorphism induces an equivalence, and it can be shown that every equivalence induces an isomorphism, but this correspondence is not one-to-one. We denote an equivalence between A and B by $f : A \simeq B$.

We can already give examples of equivalences:

Lemma 1.0.6. *For every identification $p : x = y$ in A and an element $z : A$, the operations*

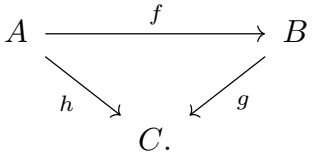
$$(-)^{-1} : (x = y) \to (y = x)$$
$$- \bullet p : (z = x) \to (z = y)$$
$$p \bullet - : (y = z) \to (x = z)$$

are equivalences.

Lemma 1.0.7. *For every type A, type family $P : A \to \mathcal{U}$, two elements $x, y : A$ and an identification $p : x = y$, the transport map $\mathrm{tr}_P p : P(x) \to P(y)$ is an equivalence.*

One of the elementary results about equivalences we use is their 3-for-2 property, an extension of the fact that they are closed under composition.

Lemma 1.0.8 (3-for-2 property of equivalences)**.** *Consider a commuting triangle of maps*



If any two of the maps are equivalences, then so is the third.

Every function $f : A \to B$ gives rise to the type family of **fibers** $\text{fib}_f : B \to \mathcal{U}$; the fiber of f over $b : B$ is the type of elements $a : A$ equipped with an identification of type $fa = b$.

We note that equivalences can be characterized by a special property of their fibers. To state it, we need a notion of contractibility — a type A is **contractible** if there is a point $a_0 : A$ and a family of identifications of type $(a : A) \to (a_0 = a)$.

Lemma 1.0.9. *A map $f : A \to B$ is an equivalence if and only if for all $b : B$ the fiber $\text{fib}_f(b)$ is contractible.*

For a thorough discussion of different characterization of equivalences, refer to Chapter 4 of the HoTT Book [18].

The hallmark of Homotopy Type Theory is the univalence axiom, due to Voevodsky, which characterizes the identity types of universes.

Axiom 1.0.10 (Univalence)**.** *For any two types $A, B : \mathcal{U}$, the map*

$$\text{equiv-eq} : (A = B) \to (A \simeq B),$$

defined by taking refl *to the identity equivalence* id*, is an equivalence.*

The map equiv-eq gives an alternative way to turn an identification $x = y$ into an equivalence $P(x) \simeq P(y)$. This turns out to be the same equivalence as transport.

Lemma 1.0.11. *Given an identification $p : x = y$ in A and a type family $P : A \to \mathcal{U}$, the equivalences $\text{tr}_P(p)$ and $\text{equiv-eq}(\text{ap}_P(p))$ are identical.*

Voevodsky showed that function extensionality follows from univalence. For a simplified proof, see [18, section 4.9].

Theorem 1.0.12 (Function extensionality)**.** *For any two types A and B, and maps $f, g : A \to B$, the map*

$$\text{htpy-eq} : (f = g) \to (f \simeq g),$$

defined by taking refl *to the reflexivity homotopy* refl-htpy*, is an equivalence. We call its inverse* eq-htpy*.*

Function extensionality gives us an induction principle for homotopies, analogous to path induction.

Lemma 1.0.13. *Consider a type A, a type family $P : A \to \mathcal{U}$, and a function $f : (a : A) \to P(a)$. Given a type family $C : (g : (a : A) \to P(a)) \to (f \sim g) \to \mathcal{V}$, there is a function*

$$\text{ind-htpy}_C : C(f, \text{refl-htpy}) \to ((g : (a : A) \to P(a))(r : f \sim g) \to C(g, r))$$

and for each $c_0 : C(f, \text{refl-htpy})$ an identification $\text{ind-htpy}_C(c_0, f, \text{refl-htpy}) = c_0$.

Definition 1.0.14. Consider a type A with a basepoint $a_0 : A$, and a pair (P, p_0) consisting of a type family $P : A \to \mathcal{U}$ and a point $p_0 : P(a_0)$. For any type family $Q : (a : A) \to P(a) \to \mathcal{V}$, there is an evaluation map

$$\text{ev-refl-id-system}_Q : ((a : A)(p : P(a)) \to Q(a, p)) \to Q(a_0, p_0)$$

defined as $h \mapsto h(a_0, p_0)$. The pair (P, p_0) is an **identity system** if for all Q the evaluation map $\text{ev-refl-id-system}_Q$ has a section, i.e. a converse map

$$\text{ind}_Q : Q(a_0, p_0) \to ((a : A)(p : P(a)) \to Q(a, p))$$

such that for all $q_0 : Q(a_0, p_0)$ there is an identification $\text{ind}_Q(q_0, a_0, p_0) = q_0$.

The based J rule essentially states that $(\text{Id}(a_0), \text{refl})$ is an identity system.

Lemma 1.0.15. *Given a type A with a basepoint a_0, the pair $(\text{Id}(a_0), \text{refl})$ is an identity system.*

The following is a variation of the encode-decode method due to Licata and Shulman [8].

Theorem 1.0.16 (Fundamental theorem of identity types)**.** *Given a type A, an element $a_0 : A$, a type family $P : A \to \mathcal{U}$, and a function $f : (x : A) \to (a_0 = x) \to P(x)$, the following are logically equivalent*

- *The function $f(x) : (a_0 = x) \to P(x)$ is an equivalence for all $x : A$*
- *The type ΣAP is contractible.*

Corollary 1.0.17. *For a type A with an element $a_0 : A$ and a type family $P : A \to \mathcal{U}$ with a point $p_0 : P(a_0)$, it holds that the type of families of equivalences*

$$e : (x : A) \to ((a_0 = x) \simeq P(x)),$$

such that $e(a_0, \text{refl}) = p_0$, is contractible.

Just as equivalences characterize the identity types in universes, and homotopies characterize the identity types in Π types, there is a characterization of identity types in Σ types.

Lemma 1.0.18. *Given a type A and a type family $P : A \to \mathcal{U}$, the map*

$$\text{pair-eq-}\Sigma : ((x, s) = (y, t)) \to \Sigma(p : x = y).\ \text{tr}_P(p, s) = t,$$

which sends refl *to* $(\text{refl}, \text{refl})$*, is an equivalence, for all $x, y : A$, $s : P(x)$ and $t : P(y)$.*

We can dispense without mentioning transport if we have characterizations of identity types of the individual components. The following theorem, known as the Structure Identity Principle, is due to Aczel [1].

Theorem 1.0.19 (Structure identity principle)**.** *Consider a type A with a point $a_0 : A$, type families $P : A \to \mathcal{U}$ and $Q : A \to \mathcal{V}$ pointed by $p_0 : P(a_0)$ and $q_0 : Q(a_0)$, respectively, and a type family $D : (x : A) \to (P(x) \to Q(x) \to \mathcal{U})$. If Q is an identity system at q_0 and there is an element $d_0 : D(a_0, p_0, q_0)$, then the following are equivalent:*

- *Any family of maps $f : (p : P(a_0)) \to ((p_0 = p) \to D(a_0, p, q_0))$ is a family of equivalences*
- *The total space $\Sigma(p : P(a_0)).\, D(a_0, p, q_0)$ is contractible.*

The last fact we will need is distributivity of Π over Σ, sometimes called the "type theoretic principle of choice". It states that a dependent function into a Σ type corresponds to a dependent pair of dependent functions.

Lemma 1.0.20. *Given a type A, a type family $P : A \to \mathcal{U}$ and another type family $Q : (a : A) \to P(a) \to \mathcal{V}$, there is an equivalence*

$$\begin{aligned}&((x : A) \to \Sigma(p : P(x)).\, Q(x, p)) \\ &\simeq (\Sigma(h : (x : X) \to P(x)).\, (x : X) \to Q(x, h(x))).\end{aligned}$$

Chapter 2

Pushouts

Pushouts are a well-known concept from category theory, encoding the operation of putting two or more objects side-by-side and abstractly "gluing" some parts together. Forming a homotopy pushout of a span of abstract spaces $A \leftarrow f - S - g \rightarrow B$ corresponds to forming the coproduct $A + B$, and then adding a path from $\mathrm{inl}(fs)$ to $\mathrm{inr}(gs)$ for every $s : S$. Thus pushout can be represented by the following higher inductive type:

```
data Pushout {S A B : Type} (f : S → A) (g : S → B) : Type where
  inl : A → Pushout f g
  inr : B → Pushout f g
  glue : (s : S) → inl(f s) = inr(g s)
```

The Agda proof assistant, in which the formalization is carried out, only supports definition of higher inductive types in its cubical mode, which the agda-unimath library does not use. Instead, we formalize pushouts as structured types satisfying a universal property, which encodes the elimination principle one would get from their definition as a higher inductive type. The downside is that maps induced by the universal property only compute up to an identification, i.e. a map $f : X \to Y$ defined by a cocone (i, j, H) applied to an element $\mathrm{inl}(a) : X$ does not reduce to $i(a)$, which complicates the development. On the other hand, it encourages a style of formalization which does not rely on the standard pushout type as much, making the formalization more modular, because theorems deriving the universal property for non-standard pushouts may be used directly, instead of going through an induced equivalence with the standard pushouts.

We open the chapter by introducing pushouts as structured types satisfying a universal property. We then proceed to discuss the descent property of pushouts, which describes how type families over pushouts correspond to type families over its components with a compatibility condition. We develop an infrastructure for relating other concepts between the realm of type families and the realm of descent data, such as sections of type families and fiberwise equivalences. The succeeding section describes the flattening lemma, which identifies Σ types over pushouts as pushouts themselves. Descent and flattening together allow us to concisely state an induction principle of identity types of pushouts, by identifying certain pointed descent data as identity systems, analogously to identity systems defined in Chapter 1.

2.1 Universal property

To define pushouts, we first define their indexing diagrams, the *span diagrams*.

Formally, we differentiate between the concept of a "span", which is an element on a structure with a fixed domain and codomain, and a "span diagram", which is a pair of types with a span between them. The distinction is important when looking at morphisms of these structures — a morphism of spans is a map between the spanning types, equipped with two homotopies for the appropriate triangles, while a morphism of span diagrams is a natural transformation, consisting of three maps and two squares. The presented material does not formally require the notion of spans, so we introduce span diagrams as the primitive notion. A similar distinction may be done between "cocone structure" on a specific vertex type, and a "cocone" as a type equipped with a cocone structure. It is not realized in the current work, but there are plans to make the change in the library.

Definition 2.1.1. A **span diagram** is a quintuple (A, B, S, f, g), where A, B and S are types, and $f : S \to A$ and $g : S \to B$ are ordinary maps.

We call A, B and S the **domain**, **codomain**, and the **spanning type** of the span diagram, respectively.

Remark 2.1.2. In the prose, we will often write $\mathcal{S}$ or $\mathcal{S} \doteq (f, g)$ for a span diagram, implicitly introducing the relevant types as the domains and codomains of the maps f and g, which will by convention be called A, B and S as in the definition. We hope to not cause confusion by this choice.

Definition 2.1.3. Given a span diagram $\mathcal{S} \doteq (f, g)$ and a type X, a **cocone** under $\mathcal{S}$ on X is a triple (i, j, H), where $i : A \to X$ and $j : B \to X$ are ordinary maps, and H is a homotopy witnessing that the square

$$\begin{array}{ccc} S & \xrightarrow{g} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X \end{array}$$

commutes, i.e. $H : i \circ f \sim j \circ g$.

We write $\mathrm{cocone}(\mathcal{S}, X)$ for the type of cocones under $\mathcal{S}$ on X.

To define what a "colimiting cocone" is in type theory, we derive inspiration from the categorical description as a coclassifier of cocones under the same span diagram: a cocone c under $\mathcal{S}$ on X is a pushout if maps $X \to Y$ are in bijection with cocones under $\mathcal{S}$ on Y. There is a natural construction for extending a cocone c on X by a map $X \to Y$ to a cocone on Y, and we say that c is a pushout of $\mathcal{S}$ exactly when this extension map is an equivalence $(X \to Y) \simeq \mathrm{cocone}(\mathcal{S}, Y)$.

Construction 2.1.4. Given a cocone $c \doteq (i, j, H) : \mathrm{cocone}(\mathcal{S}, X)$ and a type Y we construct a map

$$\text{cocone-map}_c^Y : (X \to Y) \to \mathrm{cocone}(\mathcal{S}, Y)$$

which sends h to $(h \circ i, h \circ j, h \cdot_l H)$.

We may omit the upper index Y or the lower index c, or both, if the appropriate value is clear from context.

Definition 2.1.5. A cocone c under $\mathcal{S}$ on X satisfies the **universal property of pushouts** if for all types Y the map cocone-map^Y_c is an equivalence.

A cocone satisfying the universal property of pushouts is called a **pushout**. We will sometimes abuse notation and call just the type X the pushout.

Having cocone-map be an equivalence means that we not only have the converse map, which maps cocones to functions, but additionally the converse map is a section and a retraction. In particular, it being a section means that the cocone induced by the obtained map is the same as the original cocone. However identifications of cocones are not very practical objects. Instead of using them directly, we characterize the identity types of cocones as homotopies of cocones.

Definition 2.1.6. Given a span diagram $\mathcal{S} \doteq (f, g)$ and two cocones $c \doteq (i, j, H)$ and $c' \doteq (i', j', H')$ on X, the type of **homotopies** between c and c', denoted $c \sim c'$, is the type of triples (K_A, K_B, α), where K_A and K_B are homotopies

$$K_A : i \sim i'$$
$$K_B : j \sim j'$$

and α is a coherence witnessing that the following square of homotopies commutes

$$\begin{array}{ccc} i \circ f & \overset{K_A \cdot_r f}{\sim} & i' \circ f \\ {\scriptstyle H}\big\| & & \big\|{\scriptstyle H'} \\ j \circ g & \underset{K_B \cdot_r g}{\sim} & j' \circ g. \end{array}$$

Construction 2.1.7. Given a cocone $c \doteq (i, j, H) : \text{cocone}(\mathcal{S}, X)$, construct the **reflexivity homotopy** cocone-refl-htpy $: c \sim c$ from the data

$$\begin{aligned} &\text{refl-htpy} &&: i \sim i \\ &\text{refl-htpy} &&: j \sim h \\ &\text{runit-htpy} &&: H \bullet_h \text{refl-htpy} \sim H. \end{aligned}$$

Lemma 2.1.8. *For a span diagram $\mathcal{S}$ and two cocones c and c' on X, the map*

$$\text{htpy-eq-cocone} : (c = c') \simeq (c \sim c'),$$

which sends refl *to* cocone-refl-htpy*, is an equivalence.*

The proof is a prototypical application of the fundamental theorem of identity types and the structure identity principle. We only write this one out for demonstration, as other straightforward proofs of characterizations of identity types are omitted from the thesis.

Proof. We use Theorem 1.0.16 to prove that it is an equivalence, so it suffices to show that the type of cocones c' such that $c \sim c'$ is contractible. Since $c \sim c'$ is a Σ type, we invoke the structure identity principle (Theorem 1.0.19), which leaves us to show that the type

$$\Sigma(i' : A \to X).\, (i \sim i')$$

is contractible to some point (i', L), which by function extensionality (Theorem 1.0.12) it is at $(i, \text{refl-htpy})$, and then that the type

$$\begin{aligned}&\Sigma\,(j' : B \to X)(H' : i \circ f \sim j' \circ g).\\ &\quad \Sigma\,(K_B : j \sim j').\,(H \bullet_h K_B \cdot_r g) \sim (\text{refl-htpy} \bullet_h H')\end{aligned}$$

is contractible.

We use the structure identity principle again, so the new goal is to show that

$$\Sigma(j' : B \to X).\,(j \sim j')$$

is contractible, which it is at $(j, \text{refl-htpy})$, and that the type

$$\Sigma(H' : i \circ f \sim j \circ g).\,(H \bullet_h \text{refl-htpy}) \sim H'$$

is contractible. And it is contractible once again at $(H \bullet_h \text{refl-htpy}, \text{refl-htpy})$. □

Lemma 2.1.9. *Given a span diagram $\mathcal{S} \doteq (f, g)$, its pushout cocone $c \doteq (i, j, H)$ on X, and a cocone $c' \doteq (i', j', H') : \text{cocone}(\mathcal{S}, Y)$, there is a unique map $h : X \to Y$ equipped with the homotopies*

$$\begin{aligned}K_A &: h \circ i \sim i'\\ K_B &: h \circ j \sim j'\end{aligned}$$

and the coherence α witnessing that the following square of homotopies commutes

$$\begin{array}{ccc} h \circ i \circ f & \overset{K_A \cdot_r f}{\sim\!\sim} & i' \circ f \\ {\scriptstyle h \cdot_l H}\big\downarrow & & \big\downarrow{\scriptstyle H'} \\ h \circ j \circ g & \underset{K_B \cdot_r g}{\sim\!\sim} & j' \circ g. \end{array}$$

Proof. The data claimed to be unique is an element of the type

$$\Sigma(h : X \to Y).\,(\text{cocone-map}_c(h) \sim c'),$$

which is equivalent to the type of fibers of cocone-map$_c$ at c', by Lemma 2.1.8. Since cocone-map$_c$ is an equivalence by assumption, it has contractible fibers by Lemma 1.0.9. □

The universal property characterizes simple maps out of the colimit. In dependent type theory, we can also ask about characterizations of *dependent* maps out of the colimit. To that end we introduce dependent cocones and the dependent universal property.

Definition 2.1.10. Consider a cocone $c \doteq (i, j, H) : \text{cocone}(\mathcal{S}, X)$ and a type family $P : X \to \mathcal{U}$. A **dependent cocone** over c on P is a triple (i', j', H'), where $i' : (a : A) \to P(ia)$ and $j' : (b : B) \to P(jb)$ are dependent maps over i and j, respectively, and H is a family of identifications

$$H : (s : S) \to \text{tr}_P(Hs)(i'(fs)) = j'(gs).$$

We write dep-cocone(c, P) for the type of dependent cocones over c on P.

Construction 2.1.11. Given a cocone $c \doteq (i, j, H) : \text{cocone}(\mathcal{S}, X)$ and a type family $P : X \to \mathcal{U}$, define the map

$$\text{dep-cocone-map}_c^P : ((x : X) \to P(x)) \to \text{dep-cocone}(c, P)$$

which sends h to $(h \circ i, h \circ j, \lambda s \to \text{apd}_h(Hs))$.

We may omit the indices c or P if they are clear from context.

Definition 2.1.12. A cocone $c : \text{cocone}(\mathcal{S}, X)$ satisfies the **dependent universal property of pushouts** if for all $P : X \to \mathcal{U}$, the map $\text{dep-cocone-map}_c^P$ is an equivalence.

Note that the dependent universal property is not a property of dependent cocones, but rather a property of cocones and their extensions by dependent functions.

Definition 2.1.13. Given a span diagram $\mathcal{S} \doteq (f, g)$, a cocone $c : \text{cocone}(\mathcal{S}, X)$ and two dependent cocones $d \doteq (i, j, L)$ and $d' \doteq (i', j', L')$ on P, the type of **homotopies** between d and d', denoted $d \sim d'$, is the type of triples (K_A, K_B, α), where K_A and K_B are homotopies

$$K_A : i \sim i'$$
$$K_B : j \sim j'$$

and α is a coherence witnessing that the following square of identifications commutes for every $s : S$

$$\begin{array}{ccc} \text{tr}_P(Hs)(i(fs)) & \xrightarrow{\text{ap}_{\text{tr}_P(Hs)}(K_A(fs))} & \text{tr}_P(Hs)(i'(fs)) \\ {\scriptstyle L(s)}\Big\| & & \Big\|{\scriptstyle L'(s)} \\ j(gs) & \xrightarrow[K_B(gs)]{} & j'(gs), \end{array}$$

where H is the coherence of c.

Lemma 2.1.14. *For every pair of dependent cocones $d, d' : \text{dep-cocone}(c, P)$, there is an equivalence*

$$\text{htpy-eq-dep-cocone} : (d = d') \simeq (d \sim d').$$

Proof. Using the structure identity principle. □

Lemma 2.1.15. *Given a span diagram $\mathcal{S} \doteq (f, g)$, its pushout cocone $c \doteq (i, j, H)$ on X, and a dependent cocone $d \doteq (i', j', H') : \text{dep-cocone}(c, P)$, there is a unique dependent map $h : (x : X) \to P(x)$ equipped homotopies*

$$K_A : h \circ i \sim i'$$
$$K_B : h \circ j \sim j'$$

and a coherence α witnessing that the following square of identifications commutes for all $s : S$

$$\begin{array}{ccc} \mathrm{tr}_P(Hs)(h(i(fs))) & \overset{\mathrm{ap}_{\mathrm{tr}_P(Hs)}(K_A(fs))}{=\!=\!=} & \mathrm{tr}_P(Hs)(i'(fs)) \\ {\scriptstyle \mathrm{apd}_h(Hs)}\Big\| & & \Big\downarrow{\scriptstyle H'(s)} \\ h(j(gs)) & \underset{K_B(gs)}{=\!=\!=} & j'(gs). \end{array}$$

Proof. The type is equivalent to the contractible fibers of dep-cocone-map^P_c. □

We do not introduce a new name for cocones satisfying the dependent universal property, because the two properties turn out to be equivalent. The proof relies on the pullback property and the dependent pullback property of pushouts, which relate pushouts and pullbacks of function types. As this thesis does not discuss pullbacks, we defer the proof to Rijke [14, Theorem 25.1.4].

Theorem 2.1.16. *A cocone* $c : \mathrm{cocone}(\mathcal{S}, X)$ *satisfies the universal property of pushouts if and only if it satisfies the dependent universal property of pushouts.*

Proof. There is a chain of logical equivalences

c satisfies the universal property of pushouts
$\leftrightarrow$ c satisfies the pullback property of pushouts
$\leftrightarrow$ c satisfies the dependent pullback property of pushouts
$\leftrightarrow$ c satisfies the dependent universal property of pushouts.

□

Remark 2.1.17. This equivalence of a non-dependent and dependent universal property is a more general phenomenon. In Homotopy Type Theory, there are often multiple ways of describing universal properties. These expressions usually involve a base sort of objects and dependent objects, which can be equipped with some structure functorial in an appropriate notion of maps and dependent maps.

The functorial action

$$\mathrm{fmap} : (X \to Y) \to \mathrm{structure}(X) \to \mathrm{structure}(Y)$$

can have its arguments rearranged so that for every structured object (X, s) and a plain object Y, we get an "evaluation" map

$$\text{ev-map}^Y_{(X,s)} : (X \to Y) \to \mathrm{structure}(Y).$$

In the dependent case, we get a map

$$\text{dep-ev-map}^P_{(X,s)} : ((x : X) \to P(x)) \to \text{dep-structure}((X, s), P).$$

When talking about pushouts, we take the base objects to be types, dependent objects to be type families, and maps and dependent maps to be ordinary functions and dependent functions. The functorial structure on a type X is the structure of a cocone on X under a fixed span diagram $\mathcal{S}$, and the dependent structure on a type family $P : X \to \mathcal{U}$ over a cocone c on X is the structure of a dependent cocone on P over c. The evaluation maps are cocone-map and dep-cocone-map, respectively.

With these general definitions in place, consider a structured object (X, s); we may ask for the following properties to be satisfied:

- Universal property: For every object Y, ev-map$^{Y}_{(X,s)}$ is an equivalence.
- Dependent universal property: For every dependent object P, dep-ev-map$^{P}_{(X,s)}$ is an equivalence.
- Recursion principle and uniqueness: For every object Y, ev-map$^{Y}_{(X,s)}$ has a unique section.
- Induction principle: For every dependent object P, dep-ev-map$^{P}_{(X,s)}$ has a section.

The universal properties correspond to a notion of initiality: the evaluation map is an equivalence if and only if it has contractible fibers, i.e. for all structured objects (X, s) and (Y, t), there is a unique map $h : X \to Y$ such that ev-map$^{Y}_{(X,s)}\, h = t$. The condition asks for h to preserve the structure. In other words, the universal property says that (X, s) is the initial object in a hypothetical "category" of structured objects and homomorphisms.

It was first shown by Awodey, Gambino and Sojakova [3] that the four properties are equivalent for a class of examples, where we have objects, type families, ordinary functions, dependent functions, and the (dependent) structures are (fibered) algebras for a polynomial functor. The result was later extended by Sojakova [16] to include (fibered) algebras for W-suspensions, a higher inductive analogue of W-types. The structure of a (dependent) cocone can be expressed as a (fibered) algebra of a specific W-suspension, so this result is applicable to Theorem 2.1.16, but it has not been formalized in the library.

We will rely on informal understanding of this principle when discussing options for formalization of a universal property of the identity types of pushouts in section 2.4.

2.2 Descent property

The study of type theoretic descent describes how type families over a colimit and related concepts, such as fiberwise maps or sections, arise as local data with gluing conditions. It has been studied to some extent by Rijke in [12] and [14].

The universal property of pushouts characterizes maps out of a pushout to any type in any universe, so in particular maps where the codomain itself is a universe: a type family $P : X \to \mathcal{U}$ corresponds to a cocone (P_A, P_B, H) where $P_A : A \to \mathcal{U}$, $P_B : B \to \mathcal{U}$ are type families, and H is a homotopy in the universe $H : (s : S) \to P_A(fs) = P_B(gs)$. Since identifications in universes are characterized by equivalences via the univalence axiom, we arrive at the definition of descent data:

Definition 2.2.1. Given a span diagram $\mathcal{S} := (f, g)$, we call **descent data** over $\mathcal{S}$ a triple (P_A, P_B, P_S) consisting of type families

$$P_A : A \to \mathcal{U}$$
$$P_B : B \to \mathcal{U}$$

and a fiberwise equivalence

$$P_S : (s : S) \to P_A(fs) \simeq P_B(gs).$$

We use the notation $\mathrm{DD}(\mathcal{S})$ for the type of descent data over a span diagram $\mathcal{S}$.

It may not be immediately clear why "descent data" is an appropriate name for this concept, because there is no apparent downward motion. Traditionally, descent is studied in the context of a collection of objects X_i covering a single object X, and local structure on the individual X_i's descending onto X, collecting into a global structure, given that the pieces are appropriately compatible on any "overlaps". A pushout X of $\mathcal{S}$ is covered by A and B, and the overlaps are encoded in f and g. Structure on A and B, expressed as type families P_A and P_B, "descends" to a structure on X (a type family over X). Two elements "overlap" in X if there is an identification between them coming from S, and the gluing/compatibility condition exactly requires the local structure of P_A and P_B to agree on such elements, i.e. asks for an equivalence $P_A(fs) \simeq P_B(gs)$.

The first task is to establish an equivalence between type families over a pushout and descent data over its defining span. A map from type families to descent data is easy enough to construct:

Construction 2.2.2. Given a cocone $c \doteq (i, j, H)$ on X, construct a map

$$\text{dd-fam}_c : (X \to \mathcal{U}) \to \mathrm{DD}(\mathcal{S})$$

which sends a type family $P : X \to \mathcal{U}$ to the descent data (P_A, P_B, P_S) obtained by precomposing

$$\begin{aligned} P_A &:= (\lambda a \to P(ia)) && : A \to \mathcal{U} \\ P_B &:= (\lambda b \to P(jb)) && : B \to \mathcal{U} \end{aligned}$$

and transporting in P

$$P_S := (\lambda s \to \mathrm{tr}_P(Hs)) \quad : (s : S) \to P(i(fs)) \simeq P(j(gs)).$$

Note that $\mathrm{tr}_P(Hs)$ is an equivalence by Lemma 1.0.7.

To show that dd-fam_c is an equivalence, we employ a common technique for proving equivalences: construct a commuting diagram involving dd-fam_c in which all other maps are equivalences. By repeated applications of Lemma 1.0.8, it follows that dd-fam_c is an equivalence.

Theorem 2.2.3 (Descent property)**.** *Consider a span diagram $\mathcal{S} := (f, g)$ and its pushout cocone c on X. Then the map* dd-fam_c *is an equivalence $(X \to \mathcal{U}) \simeq \mathrm{DD}(\mathcal{S})$.*

Proof. There is a triangle of maps

$$\begin{array}{ccc} (X \to \mathcal{U}) & \xrightarrow[\simeq]{\text{cocone-map}_c} & \mathrm{cocone}(\mathcal{S}, \mathcal{U}) \\ & {\scriptstyle \text{dd-fam}_c} \searrow \quad \swarrow {\scriptstyle \simeq \ \text{tot}(\text{tot}(\lambda s \to \text{equiv-eq}))} & \\ & \mathrm{DD}(\mathcal{S}). & \end{array}$$

The top map is an equivalence by assumption, since c is a pushout. The right map is an equivalence, because the map $\mathrm{tot}(h)$ is an equivalence if and only if h is a fiberwise equivalence, and equiv-eq is an equivalence by the univalence axiom (Axiom 1.0.10). By the 3-for-2 property of equivalences (Lemma 1.0.8), it suffices to show that the triangle commutes to prove that dd-fam$_c$ is an equivalence.

By chasing a type family P along the diagram, we see that we need to provide an identification

$$(P \circ i, P \circ j, \lambda s \to \mathrm{tr}_P(Hs)) = (P \circ i, P \circ j, \lambda s \to \text{equiv-eq}(\mathrm{ap}_P(Hs))).$$

The first two components are identical. To identify the third component, we invoke function extensionality; then it suffices to prove that for all $s : S$, there is an identification of equivalences

$$\mathrm{tr}_P(Hs) = \text{equiv-eq}(\mathrm{ap}_P(Hs)),$$

which is always the case by Lemma 1.0.11 applied to the identification $Hs : i(fs) = j(gs)$. □

A corollary of dd-fam$_c$ being an equivalence is that it has contractible fibers by Lemma 1.0.9, i.e. for any descent data (P_A, P_B, P_S) there is a unique type family P such that dd-fam$_c(P) = (P_A, P_B, P_S)$. We proceed to work on characterization of identifications of descent data to get a more pleasant statement of this theorem.

Definition 2.2.4. Consider a span diagram $\mathcal{S} \doteq (f, g)$, and two descent data (P_A, P_B, P_S) and (Q_A, Q_B, Q_S) over it. A **morphism** of descent data between them is a pair of fiberwise maps

$$h_A : (a : A) \to P_A(a) \to Q_A(a)$$
$$h_B : (b : B) \to P_B(b) \to Q_B(b)$$

equipped with a family of homotopies h_S indexed by $s : S$ making the square

$$\begin{array}{ccc} P_A(fs) & \xrightarrow{h_A(fs)} & Q_A(fs) \\ {\scriptstyle P_S s}\downarrow & & \downarrow{\scriptstyle Q_S s} \\ P_B(gs) & \xrightarrow[h_B(gs)]{} & Q_B(gs) \end{array}$$

commute.

We write $(h_A, h_B, h_S) : (P_A, P_B, P_S) \to (Q_A, Q_B, Q_S)$.

Analogously, we define equivalences of descent data.

Definition 2.2.5. Consider a span diagram $\mathcal{S} \doteq (f, g)$, and two descent data (P_A, P_B, P_S) and (Q_A, Q_B, Q_S) over it. An **equivalence** of descent data between them is a pair of fiberwise equivalences

$$e_A : (a : A) \to P_A(a) \simeq Q_A(a)$$
$$e_B : (b : B) \to P_B(b) \simeq Q_B(b)$$

equipped with a family of homotopies e_S indexed by $s : S$ making the square

$$\begin{array}{ccc} P_A(fs) & \xrightarrow{e_A(fs)} & Q_A(fs) \\ {\scriptstyle P_S s}\downarrow & & \downarrow{\scriptstyle Q_S s} \\ P_B(gs) & \xrightarrow[e_B(gs)]{} & Q_B(gs) \end{array}$$

commute.

We write $(e_A, e_B, e_S) : (P_A, P_B, P_S) \simeq (Q_A, Q_B, Q_S)$.

Remark 2.2.6. Alternatively, one could define equivalences of descent data as morphisms of descent data equipped with witnesses that the relevant maps are equivalences. The two definitions of equivalences of descent data would be equivalent, but the presented one can be used directly with the structure identity principle.

Lemma 2.2.7. *For any two descent data $P' \doteq (P_A, P_B, P_S)$ and $Q' \doteq (Q_A, Q_B, Q_S)$, the map*

$$\text{equivDD-eq} : (P' = Q') \to (P' \simeq Q'),$$

defined by sending refl *to the identity equivalence, is an equivalence.*

Proof. Using the structure identity principle. □

Theorem 2.2.8. *Consider a span diagram $\mathcal{S} \doteq (f, g)$ and a pushout cocone $c \doteq (i, j, H)$ on X. Then for any descent data (P_A, P_B, P_S) over $\mathcal{S}$, the type of type families $P : X \to \mathcal{U}$ equipped with an equivalence of descent data* $\text{dd-fam}_c(P) \simeq (P_A, P_B, P_S)$ *is contractible. In other words, there is a unique quadruple (P, e_A, e_B, e_S) consisting of a type family $P : X \to \mathcal{U}$, equivalences*

$$\begin{aligned} e_A &: (a : A) \to P(ia) \simeq P_A(a) \\ e_B &: (b : B) \to P(jb) \simeq P_B(b), \end{aligned}$$

and a family of commuting squares e_s

$$\begin{array}{ccc} P(i(fs)) & \xrightarrow{e_A(fs)} & P_A(fs) \\ {\scriptstyle \mathrm{tr}_P(Hs)}\downarrow & & \downarrow{\scriptstyle P_S s} \\ P(j(gs)) & \xrightarrow[e_B(gs)]{} & P_B(gs) \end{array}$$

indexed by $s : S$.

Proof. Equivalences of descent data characterize identifications of descent data, so the type of type families $P : X \to \mathcal{U}$ equipped with an equivalence $\text{dd-fam}_c(P) \simeq (P_A, P_B, P_S)$ is equivalent to the type of type families $P : X \to \mathcal{U}$ with an identification $\text{dd-fam}_c(P) = (P_A, P_B, P_S)$. Since contractibility is preserved by equivalences, it suffices to show that the latter type is contractible. But that is exactly the type of fibers of dd-fam_c over (P_A, P_B, P_S), which are contractible on account of dd-fam_c being an equivalence. □

When relating concepts from the world of type families with concepts from the world of descent data, it can be beneficial to be parametric over the data of a type family P and its "corresponding descent data", meaning some descent

data (P_A, P_B, P_S) which is equivalent to the descent data induced by P. Of course, by the descent theorem this data is completely determined by either P or (P_A, P_B, P_S) up to identification, but this level of generality allows users to provide their own equivalences for potentially better computational properties. We introduce a shorthand notation.

Definition 2.2.9. Given a span diagram $\mathcal{S}$ and a cocone c on X, we define the type of **families with descent data** to be the type of triples (P, P', e) consisting of a type family $P : X \to \mathcal{U}$, descent data $P' : \mathrm{DD}(\mathcal{S})$, and an equivalence of descent data $e : \text{dd-fam}_c(P) \simeq P'$.

We write $e : P \approx P'$ for a family P with descent data P' related by an equivalence e. We also say that P is characterized by P'.

Remark 2.2.10. Note that we do not require c to be a pushout. In subsequent development, and in the formalization, we often parameterize constructions by a family with descent data, which incentivizes general constructions applicable to non-pushout cocones.

Remark 2.2.11. The concept of a family with descent data has a direction: the equivalence relates $\text{dd-fam}_c(P)$ on the left with P' on the right. It lends itself well to characterizations of concrete type families, where P has a specific shape, and we want to recover the shape of corresponding descent data by computing $P(ia)$'s and $P(jb)$'s. However there are applications where the converse direction is more suitable. In those cases we write $e : P' \approx P$ for descent data P', a type family P, and an equivalence of descent data $e : P' \simeq \text{dd-fam}(P)$.

As a first example of a family with descent data, we characterize the type family of based identity types.

Construction 2.2.12. Given a span diagram $\mathcal{S} \doteq (f, g)$, a cocone (i, j, H) on X and a point $x_0 : X$, construct the descent data (I_A, I_B, I_S) as

$$\begin{aligned} I_A &:= (\lambda a \to (x = ia)) && : A \to \mathcal{U} \\ I_B &:= (\lambda b \to (x = jb)) && : B \to \mathcal{U} \\ I_S &:= (\lambda s, p \to p \bullet (Hs)) : (s : S) \to I_A(fs) \simeq I_B(gs). \end{aligned}$$

The concatenation operation is an equivalence by Lemma 1.0.6.

Remark 2.2.13. Note that the basepoint x_0 is not mentioned in the notation (I_A, I_B, I_S). Whenever we use this notation, the basepoint should be clear from context.

Lemma 2.2.14. *Given a cocone and a basepoint $x_0 : X$ as above, the type family $\mathrm{Id}(x_0) : X \to \mathcal{U}$ is characterized by the descent data (I_A, I_B, I_S). Explicitly, there are equivalences*

$$\begin{aligned} e_A &: (a : A) \to (x_0 = ia) \simeq I_A(a) \\ e_B &: (b : B) \to (x_0 = jb) \simeq I_B(b) \end{aligned}$$

and a family of commuting squares e_S

$$\begin{array}{ccc} (x_0 = i(fs)) & \xrightarrow{e_A(fs)} & I_A(fs) \\ {\scriptstyle \mathrm{tr}_{\mathrm{Id}(x_0)}(Hs)}\downarrow & & \downarrow{\scriptstyle I_S s} \\ (x_0 = j(gs)) & \xrightarrow[e_B(gs)]{} & I_B(gs). \end{array}$$

Proof. By definition, $I_A(a) \doteq (x_0 = ia)$ and $I_B(b) \doteq (x_0 = jb)$, so we may choose the identity equivalence for both e_A and e_B. Then the coherence datum amounts to showing that $\mathrm{tr}_{\mathrm{Id}(x_0)}(Hs, p) = p \bullet (Hs)$, which is Lemma 1.0.3. □

For any given type family $P : X \to \mathcal{U}$, we can talk about its *sections*, elements of the type $(x : X) \to P(x)$. We define an analogous concept of *sections of descent data*, and show that indeed they correspond to sections of type families over pushouts.

Definition 2.2.15. Given a span diagram $\mathcal{S}$ and descent data (P_A, P_B, P_S) over it, a **section** of (P_A, P_B, P_S) is a triple (t_A, t_B, t_S) consisting of sections

$$t_A : (a : A) \to P_A(a)$$
$$t_B : (b : B) \to P_B(b)$$

and a coherence

$$t_S : (s : S) \to P_S(s, t_A(fs)) = t_B(gs).$$

We write $\mathsf{sect}(P_A, P_B, P_S)$ for the type of sections of (P_A, P_B, P_S).

Construction 2.2.16. Given a span diagram $\mathcal{S} \doteq (f, g)$, a cocone $c \doteq (i, j, H)$ on X, and a family with descent data $e : P \approx (P_A, P_B, P_S)$, construct a map

$$\text{sect-sect}_c : ((x : X) \to P(x)) \to \mathsf{sect}(P_A, P_B, P_S)$$

by assigning to a dependent function h the section

$$\begin{aligned}
&(\lambda a \to e_A(h(ia))) && : (a : A) \to P_A(a)\\
&(\lambda b \to e_B(h(jb))) && : (b : B) \to P_B(b)\\
&(\lambda s \to (e_S(h(i(fs))))^{-1} \bullet \mathsf{ap}_{e_B}(\mathsf{apd}_h(Hs))) && : (s : S) \to\\
& && \quad P_S(s, e_A(h(i(fs)))) = e_B(h(j(gs))).
\end{aligned}$$

Lemma 2.2.17. *Consider a span diagram $\mathcal{S}$, a pushout cocone c on X and a family with descent data $P \approx (P_A, P_B, P_S)$. Then the map* sect-sect_c *is an equivalence.*

Proof. The map factors through the dependent cocone map as

$$\begin{array}{ccc}
((x : X) \to P(x)) & \xrightarrow[\simeq]{\text{dep-cocone-map}_c} & \mathsf{dep\text{-}cocone}(c, P)\\
& {\scriptstyle \text{sect-sect}_c} \searrow \quad \swarrow {\scriptstyle \simeq} & \\
& \mathsf{sect}(P_A, P_B, P_S), &
\end{array}$$

where the right map takes (i', j', H') to

$$\begin{aligned}
&(\lambda a \to e_A(i'a)) && : (a : A) \to P_A(a)\\
&(\lambda b \to e_B(j'b)) && : (b : B) \to P_B(b)\\
&(\lambda s \to (e_S(i'(fs)))^{-1} \bullet \mathsf{ap}_{e_B}(H's)) && : (s : S) \to\\
& && \quad P_S(s, e_A(i'(fs))) = e_B(j'(gs)).
\end{aligned}$$

The right map is an equivalence, because its action on the first two components is postcomposition by a fiberwise equivalence, which is an equivalence, and its action on the third component is a fiberwise application of ap_{e_B}, which is an equivalence, and concatenation with an identification, which is an equivalence.

The triangle commutes by refl-htpy. By the 3-for-2 property of equivalences, it follows that sect-sect$_c$ is an equivalence. □

Equipped with the tools for computing data over pushouts by gluing together data over its components, we continue by computing fiberwise maps and equivalences over pushouts. We first characterize type families of fiberwise maps, i.e. families with fibers of the shape $P(x) \to Q(x)$.

Remark 2.2.18. It is important to differentiate between families of *function types*, i.e. a type family that to every $x : X$ assigns the *type* $P(x) \to Q(x)$, and families of *functions*, i.e. a family that to every $x : X$ assigns a *function* from $P(x)$ to $Q(x)$. Descent data plays the role of a family of types, so it makes sense to talk about "descent data corresponding to a family of function types", but it does not make sense to talk about "descent data corresponding to a family of functions". The kind of objects that corresponds to families of functions are the sections of the descent data of a family of function types.

Lemma 2.2.19. *Given a span diagram $\mathcal{S} \doteq (f, g)$, a cocone c on X and two families with descent data $e^P : P \approx (P_A, P_B, P_S)$ and $e^Q : Q \approx (Q_A, Q_B, Q_S)$, the type family*

$$(\lambda x \to (P(x) \to Q(x))) : X \to \mathcal{U}$$

is characterized by the descent data

$$\begin{aligned}
&(\lambda a \to (P_A(a) \to Q_A(a))) && : A \to \mathcal{U} \\
&(\lambda b \to (P_B(b) \to Q_B(b))) && : B \to \mathcal{U} \\
&(\lambda s, h \to Q_S(s) \circ h \circ (P_S(s))^{-1}) && : (s : S) \to \\
& && \quad (P_A(fs) \to Q_A(fs)) \simeq (P_B(gs) \to Q_B(gs)).
\end{aligned}$$

Note that postcomposition and precomposition by an equivalence is an equivalence of function types.

Proof. We need to provide equivalences

$$\begin{aligned}
e_A &: (a : A) \to (P(ia) \to Q(ia)) \simeq (P_A(a) \to Q_A(a)) \\
e_B &: (b : B) \to (P(jb) \to Q(jb)) \simeq (P_B(b) \to Q_B(b))
\end{aligned}$$

and a coherence e_S

$$\begin{array}{ccc}
(P(i(fs)) \to Q(i(fs))) & \xrightarrow{e_A(fs)} & (P_A(fs) \to Q_A(fs)) \\
{\scriptstyle \mathrm{tr}_{(\lambda x \to (P(x) \to Q(x)))}(Hs)}\downarrow & & \downarrow{\scriptstyle Q_S(s) \circ - \circ (P_S(s))^{-1}} \\
(P(j(gs)) \to Q(j(gs))) & \xrightarrow[e_B(gs)]{} & (P_B(gs) \to Q_B(gs)).
\end{array}$$

Define the equivalences by

$$e_A(a,h) :\equiv e_A^Q(a) \circ h \circ (e_A^P(a))^{-1}$$
$$e_B(b,h) :\equiv e_B^Q(b) \circ h \circ (e_B^P(b))^{-1}.$$

Transport in a type family of function types can be computed as composition of transports in the involved families by Lemma 1.0.4, so the left map can be replaced by $\mathrm{tr}_Q(Hs) \circ - \circ \mathrm{tr}_P(Hs)^{-1}$. Since we want to identify two functions, we invoke function extensionality, and are left with the goal

$$\begin{array}{ccccccc}
P_B(gs) & \xrightarrow{(P_S(s))^{-1}} & P_A(fs) & & & & \\
\downarrow{\scriptstyle (e_B^P(gs))^{-1}} & & \downarrow{\scriptstyle (e_A^P(fs))^{-1}} & & & & \\
P(j(gs)) & \xrightarrow[\mathrm{tr}_P(Hs)^{-1}]{} & P(i(fs)) & \xrightarrow{h} & Q(i(fs)) & \xrightarrow{e_A^Q(fs)} & Q_A(fs) \\
 & & & & \downarrow{\scriptstyle \mathrm{tr}_Q(Hs)} & & \downarrow{\scriptstyle Q_S(s)} \\
 & & & & Q(j(gs)) & \xrightarrow[e_B^Q(gs)]{} & Q_B(gs)
\end{array}$$

for all $h : P(i(fs)) \to Q(i(fs))$. The right square is exactly $e_S^Q(s)$, and the left square is $e_S^P(s)$ mirrored vertically and horizontally. □

Lemma 2.2.20. *The type of sections of the descent data defined in Lemma 2.2.19 is equivalent to morphisms $(P_A, P_B, P_S) \to (Q_A, Q_B, Q_S)$.*

Proof. The first two components of a section and a morphism are the same, namely

$$h_A : (a : A) \to P_A(a) \to Q_A(a)$$
$$h_B : (b : B) \to P_B(b) \to Q_B(b).$$

It then suffices to give, for every $s : S$, an equivalence

$$((Q_S(s) \circ h_A(fs) \circ (P_S(s))^{-1}) = h_B(gs))$$
$$\simeq ((h_B(gs) \circ P_S(s)) \sim (Q_S(s) \circ h_A(fs))).$$

We obtain it by composing the following chain of equivalences:

$$\begin{array}{ll}
((Q_S(s) \circ h_A(fs) \circ (P_S(s))^{-1}) = h_B(gs)) & \\
\simeq ((Q_S(s) \circ h_A(fs) \circ (P_S(s))^{-1}) \sim h_B(gs)) & \text{by function extensionality} \\
\simeq ((Q_S(s) \circ h_A(fs)) \sim (h_B(gs) \circ P_S(s))) & \text{by transposition} \\
\simeq ((h_B(gs) \circ P_S(s)) \sim (Q_S(s) \circ h_A(fs))) & \text{by inversion.}
\end{array}$$

□

Theorem 2.2.21. *Consider a span diagram $\mathcal{S}$, a pushout cocone $c \doteq (i, j, H)$ on X, and two families with descent data $e^P : P \approx (P_A, P_B, P_S)$ and $e^Q : Q \approx (Q_A, Q_B, Q_S)$. Then there is an equivalence*

$$\text{hom-map} : ((x : X) \to P(x) \to Q(x)) \simeq ((P_A, P_B, P_S) \to (Q_A, Q_B, Q_S)).$$

Additionally, the following diagrams commute for all $h : (x : X) \to P(x) \to Q(x)$

$$\begin{array}{ccc} P(ia) & \xrightarrow{h(ia)} & Q(ia) \\ {\scriptstyle e^P_A(a)}\downarrow & & \downarrow{\scriptstyle e^Q_A(a)} \\ P_A(a) & \xrightarrow[\text{hom-map}(h)_A(a)]{} & Q_A(a) \end{array} \qquad \begin{array}{ccc} P(jb) & \xrightarrow{h(jb)} & Q(jb) \\ {\scriptstyle e^P_B(b)}\downarrow & & \downarrow{\scriptstyle e^Q_B(b)} \\ P_B(b) & \xrightarrow[\text{hom-map}(h)_B(b)]{} & Q_B(b). \end{array}$$

Proof. The type of fiberwise maps is by definition the type of sections of the family $\lambda x \to (P(x) \to Q(x))$, which is equivalent to the type of sections of the descent data from Lemma 2.2.19 by Lemma 2.2.17. That type of sections is equivalent to the type of morphisms of descent data by Lemma 2.2.20.

Computing the action of this composed equivalence on a fiberwise map $h : (x : X) \to P(x) \to Q(x)$, we get the judgmental equalities

$$\begin{aligned} \text{hom-map}(h)_A &\doteq \lambda a \to e^Q_A(a) \circ h(ia) \circ (e^P_A(a))^{-1} \\ \text{hom-map}(h)_B &\doteq \lambda b \to e^Q_B(b) \circ h(jb) \circ (e^P_B(b))^{-1}, \end{aligned}$$

so by transposing $e^P_A(a)$ and $e^P_B(b)$, we get the desired computation rules. □

Completely analogously, we may characterize the type family of equivalence types, and show that fiberwise equivalences correspond to equivalences of descent data. We present the statements here without proof, however their formalization is available in the attached source code.

Lemma 2.2.22. *Given a cocone c on X and two families with descent data $e^P : P \approx (P_A, P_B, P_S)$ and $e^Q : Q \approx (Q_A, Q_B, Q_S)$, the type family*

$$(\lambda x \to (P(x) \simeq Q(x))) : X \to \mathcal{U}$$

is characterized by the descent data

$$\begin{aligned} &(\lambda a \to (P_A(a) \simeq Q_A(a))) && : A \to \mathcal{U} \\ &(\lambda b \to (P_B(b) \simeq Q_B(b))) && : B \to \mathcal{U} \\ &(\lambda s, h \to Q_S(s) \circ h \circ (P_S(s))^{-1}) && : (s : S) \to \\ & && \quad (P_A(fs) \simeq Q_A(fs)) \simeq (P_B(gs) \simeq Q_B(gs)). \end{aligned}$$

Theorem 2.2.23. *Consider a span diagram $\mathcal{S}$, a pushout cocone $c \doteq (i, j, H)$ on X, and two families with descent data $e^P : P \approx (P_A, P_B, P_S)$ and $e^Q : Q \approx (Q_A, Q_B, Q_S)$. Then there is an equivalence*

$$\text{equivDD-equiv} : ((x : X) \to P(x) \simeq Q(x)) \simeq ((P_A, P_B, P_S) \simeq (Q_A, Q_B, Q_S)).$$

Additionally, the following diagrams commute for all $e : (x : X) \to P(x) \simeq Q(x)$

$$\begin{array}{ccc} P(ia) & \xrightarrow{e(ia)} & Q(ia) \\ {\scriptstyle e^P_A(a)}\downarrow & & \downarrow{\scriptstyle e^Q_A(a)} \\ P_A(a) & \xrightarrow[\text{equivDD-equiv}(e)_A(a)]{} & Q_A(a) \end{array} \qquad \begin{array}{ccc} P(jb) & \xrightarrow{e(jb)} & Q(jb) \\ {\scriptstyle e^P_B(b)}\downarrow & & \downarrow{\scriptstyle e^Q_B(b)} \\ P_B(b) & \xrightarrow[\text{equivDD-equiv}(e)_B(b)]{} & Q_B(b). \end{array}$$

	Families	Descent data
Objects	$P : X \to \mathcal{U}$	(P_A, P_B, P_S)
Sections	$(x : X) \to P(x)$	sect(P_A, P_B, P_S)
Morphisms	$(x : X) \to P(x) \to Q(x)$	$(P_A, P_B, P_S) \to (Q_A, Q_B, Q_S)$
Equivalences	$(x : X) \to P(x) \simeq Q(x)$	$(P_A, P_B, P_S) \simeq (Q_A, Q_B, Q_S)$
Identity objects	$\lambda x \to (x_0 = x)$	(I_A, I_B, I_S)
Identity induction	Identity systems	???

Figure 2.1: Translation table between type families and descent data

The correspondence of concepts between the world of type families over pushouts and the world of descent data is summarized in Figure 2.1. Since we want to arrive at an alternative characterization of the identity descent data (I_A, I_B, I_S) via the zigzag construction, we chose to identify its universal property. We can take inspiration from the various universal properties satisfied by the family of identity types $\mathrm{Id}(x_0) :\equiv (\lambda x \to (x_0 = x))$. Some of those properties arise from it being the initial pointed type family, in the sense of Remark 2.1.17. As we will see, in this case the induction principle can be reduced to defining a converse map; it will automatically be a section. It also corresponds to the induction principle stated by Kraus and von Raumer [7].

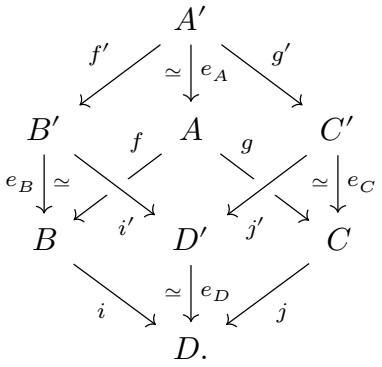
However, the induction principle speaks about dependent type families of the sort $(x : X) \to (p : P(x)) \to \mathcal{U}$. Instead of building new infrastructure for "dependent descent data", we notice that by uncurrying, those dependent type families are exactly the type families $\Sigma XP \to \mathcal{U}$! This observation makes us ask another question — to use descent, we need type families over a pushout; by assumption, X is a pushout, but here we require ΣXP to be a pushouts as well. The next section is dedicated to proving that indeed, the total space of a family over a pushout is a pushout.

2.3 Flattening lemma

The flattening lemma for pushouts effectively states that pushouts commute with dependent pair types — the total space of a type family over a pushout is a pushout of total spaces of the corresponding descent data.

The presented proof is split into two parts. First we prove the statement specifically for a type family and the descent data it induces, which reduces the amount of data we need to make coherent. Then we relate the cocone for descent data induced by the family to the cocone for arbitrary corresponding descent data, via a commuting cube whose vertical maps are equivalences. We do not write down the proof of the lemma that such cubes preserve pushouts, since it factors through the dual statement for pullbacks, which is out of scope of the thesis. Its formalization can be found in the attached code.

Lemma 2.3.1. *Consider a commuting cube*



where the vertical maps are equivalences. Then the bottom square is a pushout square if and only if the top square is a pushout square.

Construction 2.3.2. Given a span diagram $\mathcal{S} \doteq (f, g)$ and descent data (P_A, P_B, P_S), construct the **total span diagram** $\Sigma\mathcal{S}$

$$\Sigma A P_A \xleftarrow{\mathrm{tot}_f(\mathrm{id})} \Sigma S(P_A \circ f) \xrightarrow{\mathrm{tot}_g(P_S)} \Sigma B P_B.$$

Construction 2.3.3. Given a span diagram $\mathcal{S} \doteq (f, g)$, a cocone $c \doteq (i, j, H)$ on X, and a family with descent data $(e_A, e_B, e_S) : (P_A, P_B, P_S) \approx P$, construct the **total cocone** Σc under the total span diagram

$$\begin{array}{ccc} \Sigma S(P_A \circ f) & \xrightarrow{\mathrm{tot}_g(P_S)} & \Sigma B P_B \\ {\scriptstyle \mathrm{tot}_f(\mathrm{id})}\downarrow & H' & \downarrow{\scriptstyle \mathrm{tot}_j(e_B)} \\ \Sigma A P_A & \xrightarrow[\mathrm{tot}_i(e_A)]{} & \Sigma X P, \end{array}$$

where the coherence H' at $s : S$, $p : P_A(fs)$ is given by

$$\begin{aligned} H'_1 &:\equiv H(s) &&: i(fs) = j(gs) \\ H'_2 &:\equiv e_S(s,p)^{-1} &&: \mathrm{tr}_P(H(s), e_A(s,p)) = e_B(P_S(s,p)), \end{aligned}$$

implicitly using the fact that an identifications in Σ types consist of pairs of identifications (Lemma 1.0.18).

Lemma 2.3.4. *Given a pushout square (i, j, H) on X and a type family $P : X \to \mathcal{U}$, the total cocone of $(P \circ i, P \circ j, \mathrm{tr}_P(H)) \approx P$ is a pushout.*

Proof. The goal is to prove that for any type Y, the map

$$\text{cocone-map}_{\Sigma c} : (\Sigma X P \to Y) \to \text{cocone}(\Sigma c, Y)$$

is an equivalence. We achieve that by forming a commuting pentagon, in which all other maps are equivalences:

$$
\begin{array}{ccc}
(\Sigma XP \to Y) & \xrightarrow{\text{cocone-map}_{\Sigma c}} & \Sigma(h_A : \Sigma A(P \circ i) \to Y) \\
& & \quad (h_B : \Sigma B(P \circ j) \to Y). \\
& & \quad ((s,p) : \Sigma S(P \circ i \circ f)) \to \\
& & \quad h_A(fs,p) = h_B(gs, \mathrm{tr}_P(Hs,p)) \\
\Big\uparrow \text{ind-}\Sigma \simeq & & \\
(x : X) \to P(x) \to Y & & \Big\downarrow \simeq \text{ev-pair}^3 \\
\Big\downarrow \text{dep-cocone-map}_c \simeq & & \\
\Sigma(h_A : (a : A) \to P(ia) \to Y) & & \Sigma(h_A : (a : A) \to P(ia) \to Y) \\
\quad (h_B : (b : B) \to P(jb) \to Y). & & \quad (h_B : (b : B) \to P(jb) \to Y). \\
\quad (s : S) \to & \xleftarrow[\text{tot}(\text{tot}(\varphi))]{\simeq} & \quad (s : S)(p : P(i(fs))) \to \\
\quad \mathrm{tr}_{(\lambda x \to (P(x) \to Y))}(Hs, h_A(fs)) = h_B(gs) & & \quad h_A(fs,p) = h_B(gs, \mathrm{tr}_P(Hs,p)).
\end{array}
$$

The types $\text{cocone}(\Sigma\mathcal{S}, Y)$ and $\text{dep-cocone}(c, (\lambda x \to (P(x) \to Y)))$ were expanded in the diagram. The pentagon commutes by reflexivity on the first two components. To finish the proof, we need to define an equivalence

$$\varphi : (h_A(fs) \sim h_B(gs) \circ \mathrm{tr}_P(Hs)) \simeq (\mathrm{tr}_{(\lambda x \to (P(x) \to Y))}(Hs, h_A(fs)) = h_B(gs))$$

such that $\mathrm{apd}_h(Hs) = \varphi(\lambda p \to \mathrm{ap}_{\text{ind-}\Sigma(h)}((Hs, \mathrm{refl})))$. This map and its computation rule is defined in the next lemma in more generality, which finishes the proof. □

Lemma 2.3.5. *Given maps $i, j : S \to X$ with a homotopy $H : i \sim j$, a type family $P : X \to \mathcal{U}$, a type Y, and two dependent maps*

$$\begin{aligned} k &: (s : S) \to P(is) \to Y \\ l &: (s : S) \to P(js) \to Y, \end{aligned}$$

there is for every $s : S$ an equivalence

$$\varphi : (k(s) \sim l(s) \circ \mathrm{tr}_P(Hs)) \simeq (\mathrm{tr}_{(\lambda x \to (P(x) \to Y))}(Hs, k(s)) = l(s))$$

Additionally, for $k \doteq (h \circ i)$ and $l \doteq (h \circ j)$ where $h : (x : X) \to P(x) \to Y$ is any dependent map, it computes as

$$\varphi(\lambda p \to \mathrm{ap}_{\text{ind-}\Sigma(h)}((Hs, \mathrm{refl}))) = \mathrm{apd}_h(Hs).$$

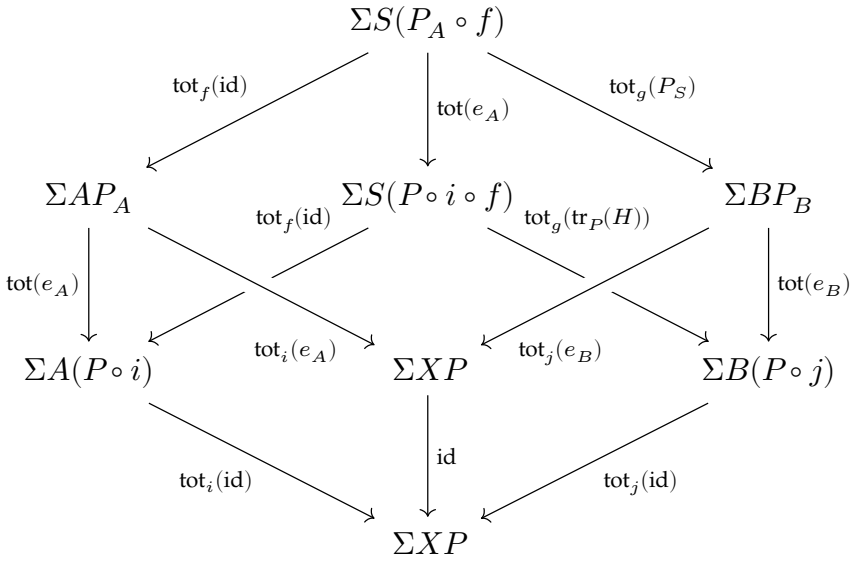
Proof. By homotopy induction (Lemma 1.0.13), it suffices to consider the case where $j \doteq i$ and H is the reflexivity homotopy. The goal is

$$(k(s) \sim l(s)) \simeq (k(s) = l(s)),$$

which holds by function extensionality.

The computation rule follows again by induction on H. Then it suffices to show that $\varphi(\text{refl-htpy}) = \text{refl}$. By computation of homotopy induction $\varphi(\text{refl-htpy})$ computes to $\text{eq-htpy}(\text{refl-htpy})$, which computes to refl. □

Lemma 2.3.6. *Given a type family $P : X \to \mathcal{U}$ with corresponding descent data (P_A, P_B, P_S), there is a commuting cube*



where the top square is the coherence of the total cocone of $(P_A, P_B, P_S) \approx P$, and the bottom square is the coherence of the total cocone of $(P \circ i, P \circ j, \mathrm{tr}_P(H)) \approx P$.

Proof. The back left, front left, and front right squares commute by refl-htpy. The back right square commutes by $(\mathrm{refl}, (e_s)^{-1})$. The commuting cube is therefore an element of the type

$$(\mathrm{tot}_i(\mathrm{id}) \cdot_l \text{refl-htpy}) \bullet_h (\text{refl-htpy} \cdot_r \mathrm{tot}_f(\mathrm{id})) \bullet_h (\mathrm{id} \cdot_l (H, e_S^{-1})) \sim$$
$$((H, \text{refl-htpy}) \cdot_r \mathrm{tot}(e_A)) \bullet_h (\mathrm{tot}_j(\mathrm{id}) \cdot_l (\text{refl-htpy}, e_S^{-1})) \bullet_h (\text{refl-htpy} \cdot_r \mathrm{tot}_g(P_S)).$$

The left homotopy computes to $\mathrm{id} \cdot_l (H, e_S^{-1})$, which is homotopic to (H, e_S^{-1}). The last concatenant of the right homotopy is refl-htpy, so we can compute it away.

The new goal is

$$(H, e_S^{-1}) \sim (H, \text{refl-htpy}) \bullet_h (\mathrm{tot}_j(\mathrm{id}) \cdot_l (\text{refl-htpy}, e_S^{-1})).$$

The total map $\mathrm{tot}_j(\mathrm{id})$ acts on $(\text{refl-htpy}, e_S^{-1})$ component-wise, so it can be further computed to $(\text{refl-htpy}, \mathrm{id} \cdot_l (e_S^{-1}))$, which is homotopic to $(\text{refl-htpy}, e_S^{-1})$. To finish the proof, we note that any identification $(p, q) : (s, t) = (s', t')$ in a Σ type can be decomposed as $(p, \mathrm{refl}) \bullet (\mathrm{refl}, q)$. □

Theorem 2.3.7 (Flattening lemma)**.** *Given a pushout c and a family with descent data $(P_A, P_B, P_S) \approx P$, the total cocone is a pushout.*

Proof. By Lemma 2.3.4, the bottom square in Lemma 2.3.6 is a pushout, and all of $e_A(a)$, $e_A(fs)$, $e_B(b)$ and id are equivalences, so it follows by Lemma 2.3.1 that the top square is a pushout. □

2.4 Identity systems

We define a universal property of descent data for the identity types of pushouts, which allows their alternative characterizations. The property is analogous to a pointed type family being an identity system, which manifests it as the homotopy-initial pointed type family (Lemma 1.0.15); in fact, we show that a type family over a pushout is an identity system if and only if the corresponding descent data satisfies this universal property.

Given descent data (P_A, P_B, P_S) for a span diagram $\mathcal{S} \doteq (f, g)$ and a point $p_0 : P_A(a_0)$ over a basepoint $a_0 : A$, we would like to mirror the definition of identity systems. A naïve translation would lead us to define dependent descent data and its sections. We choose to sidestep building that technical infrastructure.

By the descent property, there is a unique type family $P : X \to \mathcal{U}$ corresponding to (P_A, P_B, P_S). Observe that the type of dependent type families $(x : X) \to P(x) \to \mathcal{U}$ is equivalent to the uncurried form $(\Sigma X P) \to \mathcal{U}$. By the flattening lemma, the total space $\Sigma X P$ is the pushout of the span diagram of total spaces

$$\Sigma A P_A \xleftarrow{\mathrm{tot}_f \mathrm{id}} \Sigma S(P_A \circ f) \xrightarrow{\mathrm{tot}_g P_S} \Sigma B P_B$$

so, again by the descent property, descent data over it correspond to type families over $\Sigma X P$. Hence we can talk about descent data $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$ over the total span diagram instead of dependent descent data. We write a Σ in the indices of Q to remind ourselves that it is descent data over the total span diagram.

Construction 2.4.1. Assume a span diagram $\mathcal{S}$, descent data (P_A, P_B, P_S) over it, a basepoint $a_0 : A$ and a point $p_0 : P_A(a_0)$. Then for any descent data $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$ over the total span, define the map

$$\text{ev-refl-id-system-DD} : \mathrm{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S}) \to Q_{\Sigma A}(a_0, p_0)$$

by sending (t_A, t_B, t_S) to $t_A(a_0, p_0)$.

Definition 2.4.2. Descent data (P_A, P_B, P_S) equipped with a point $p_0 : P_A(a_0)$ satisfies the **induction principle of identity systems** if for all $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$, the map ev-refl-id-system-DD has a section, in the sense that there is a converse map

$$\text{ind-Q} : Q_{\Sigma A}(a_0, p_0) \to \mathrm{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$$

and an identification

$$(\text{ind-Q}(q_0))_A(a_0, p_0) = q_0$$

for all $q_0 : Q_{\Sigma A}(a_0, p_0)$.

Such descent data is called an **identity system** at p_0.

Mind the unfortunate terminology clash between "sections of descent data" and "sections of a map". A section of descent data is an analogue of a dependent map into Q, while a section of a map h is a converse map s such that $(h \circ s) \sim \mathrm{id}$.

Remark 2.4.3. Note that this development is biased towards the left — we pick a basepoint in the domain $a_0 : A$, a point in the left type family $p_0 : P_A(a_0)$, and the evaluation map evaluates the left map of the section. By symmetry of pushouts we could just as well work with the points $b_0 : B$, $p_0 : P_B(b_0)$, and the evaluation map evaluating the right map of the section.

Remark 2.4.4. By showing that the canonical descent data for identity types is an identity system, we recover the "induction principle for pushout equality" stated and proved by Kraus and von Raumer [7].

First observe that the type of sections of ev-refl-id-system-DD is

$$\begin{aligned}\Sigma\ &(\text{ind-Q} : (Q_{\Sigma A}(a_0, p_0)) \to \text{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S}))\\ &((q_0 : Q_{\Sigma A}(a_0, p_0)) \to (\text{ind-Q}\, q_0)_A(a_0, p_0) = q_0),\end{aligned}$$

which is equivalent to the type

$$\begin{aligned}&(q_0 : Q_{\Sigma A}(a_0, p_0)) \to \\ &\Sigma\ (\text{ind-Q} : \text{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})) && (2.1)\\ &\quad (\text{ind-Q}_A(a_0, p_0) = q_0) && (2.2)\end{aligned}$$

by Lemma 1.0.20.

Then the induction terms from [7] (with names changed to fit our naming scheme)

$$\begin{aligned}\text{ind}_\text{A} &: (a : A)(r : i(a_0) = i(a)) \to Q_{\Sigma A}(a, r)\\ \text{ind}_\text{B} &: (b : B)(r : i(a_0) = j(b)) \to Q_{\Sigma B}(b, r)\end{aligned}$$

are the first and second components of the section 2.1 induced by q_0, and their computation rules

$$\begin{aligned}&\text{ind}_\text{A}(a_0, \text{refl}) = q_0\\ &Q_{\Sigma S}(s, r, \text{ind}_\text{A}(fs, r)) = \text{ind}_\text{B}(gs, r \bullet Hs)\end{aligned}$$

arise as the second component 2.2, and the coherence condition of 2.1, respectively.

We first show a result relating identity systems stated as pointed type families and identity systems stated as pointed descent data.

Lemma 2.4.5. *Consider a pushout cocone c on X, a type family with corresponding descent data $e^P : P \approx (P_A, P_B, P_S)$ and a point $p_0 : P_A(a_0)$. Then for any type family with corresponding descent data $e^Q : Q_\Sigma \approx (Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$ there is a commuting diagram*

$$\begin{array}{ccccc}((x : X)(p : P(x)) \to Q_\Sigma(x, p)) & \xrightarrow{\simeq} & ((u : \Sigma X P) \to Q_\Sigma u) & \xrightarrow{\simeq} & \text{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})\\ \Big\downarrow{\scriptstyle \text{ev-refl-id-system}} & & & & \Big\downarrow{\scriptstyle \text{ev-refl-id-system-DD}}\\ Q_\Sigma(ia_0, (e^P_A(a_0))^{-1}(p_0)) & & \xrightarrow[e^Q_A(a_0,p_0)]{\simeq} & & Q_{\Sigma A}(a_0, p_0).\end{array} \tag{2.3}$$

Proof. The top equivalences are, from left to right, ind-Σ and sect-sect$_{\Sigma c}$. To see that the square commutes, note that the first component of sect-sect$_{\Sigma c}$(ind-$\Sigma(h)$) sends $(a, p) : \Sigma A P_A$ to $e^Q_A(h(ia, (e^P_A(a))^{-1}(p)))$ by definition of the total cocone. The square commutes by refl-htpy. □

Corollary 2.4.6. *Assume a pushout cocone c on X and a family with descent data $e : P \approx (P_A, P_B, P_S)$ where P is an identity system at $(e_A(a_0))^{-1}(p_0) : P(ia_0)$. Then (P_A, P_B, P_S) is an identity system at p_0.*

Proof. For every $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$ there is a corresponding type family Q_Σ. Then we may apply Equation 2.4.5. The top and bottom maps are equivalences, and the left maps has a section by assumption, hence the right map has a section. □

Corollary 2.4.7. *Analogously, if (P_A, P_B, P_S) is an identity system at $p_0 : P_A(a_0)$, then P is an identity system at $(e_A(a_0))^{-1}(p_0)$.*

Theorem 2.4.8. *Given a span diagram $\mathcal{S}$, a point $a_0 : A$, and a pushout cocone c on X, the descent data (I_A, I_B, I_S) is an identity system at* refl_{ia_0}.

Proof. By Lemma 2.2.14 and Corollary 2.4.6, the descent data (I_A, I_B, I_S) is an identity system at refl $: (ia_0) = (ia_0)$ if and only if the corresponding type family $\text{Id}(ia_0) : X \to \mathcal{U}$ is an identity system at refl, which is established in Lemma 1.0.15. □

The induction principle of identity systems is stated in terms of an evaluation map having a section, which makes it consistent with statements of other induction principles in Homotopy Type Theory. However, the following lemma shows that the condition on the converse map of being a section is redundant.

Lemma 2.4.9. *Consider a span diagram $\mathcal{S}$, its pushout cocone c on X, and descent data (P_A, P_B, P_S). To show that (P_A, P_B, P_S) is an identity system at $p_0 : P_A(a_0)$, it suffices to provide a map*

$$M : Q_{\Sigma A}(a_0, p_0) \to \text{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$$

for every descent data $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$ over the total span diagram.

Proof. Construct the unique type family $P : X \to \mathcal{U}$ for (P_A, P_B, P_S). It suffices to show that P is an identity system. Equivalently, it suffices to show that the total space $\Sigma X P$ is contractible. We can prove that using the property that a type is contractible if we provide a point, here $(ia_0, (e^P_A a_0)^{-1}(p_0))$, and a map

$$M' : (Q_\Sigma : \Sigma X P \to \mathcal{U}) \to (q_0 : Q_\Sigma(ia_0, (e^P_A a)^{-1} p_0)) \to (u : \Sigma X P) \to Q_\Sigma(u).$$

Assume such Q_Σ and q_0. Q_Σ induces descent data $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$, and a section $(u : \Sigma X P) \to Q_\Sigma(u)$ is given by a section of $(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$. We can get such a section by applying M to $e^Q_A((a_0, p_0), q_0) : Q_{\Sigma A}(a_0, p_0)$. □

Remark 2.4.10. Note that the pushout c is not used in the statement of the lemma. We include it as a parameter to avoid assuming existence of all pushouts.

Theorem 2.4.11. *Consider a span diagram $\mathcal{S}$, a point $a_0 : A$, and a pushout cocone c. For any identity system (P_A, P_B, P_S) at $p_0 : P_A(a_0)$, there is a unique equivalence of descent data*

$$e : (I_A, I_B, I_S) \simeq (P_A, P_B, P_S)$$

such that $e_A(\text{refl}) = p_0$.

Proof. Construct the unique type family $P : X \to \mathcal{U}$ corresponding to (P_A, P_B, P_S). By Theorem 2.2.23 the type of point preserving equivalences between (I_A, I_B, I_S) and (P_A, P_B, P_S) is equivalent to the type of fiberwise equivalences $(x : X) \to ((ia_0) = x) \simeq P(x)$ that send refl to $(e^P_A a_0)^{-1}(p_0)$. To show that this type is contractible, it suffices to show that the total space $\Sigma X P$ is contractible, by Corollary 1.0.17. It is contractible if P is an identity system, which it is by Corollary 2.4.7 and the assumption that (P_A, P_B, P_S) is an identity system. □

Unfolding the equivalence, we get the data

$$\begin{aligned}
&e_A : (a : A) \to (ia_0 = ia) \simeq P_A(a) \\
&e_B : (b : B) \to (ia_0 = jb) \simeq P_B(b) \\
&e_S : (s : S)(p : ia_0 = i(fs)) \to e_B(gs, p \bullet (Hs)) = P_S(s, e_A(fs, p)),
\end{aligned}$$

which justifies that claim that identity systems allow alternative characterizations of identity types of pushouts.

Chapter 3

Other colimits

Pushouts and the empty type suffice to construct many other kinds of colimits. We are particularly interested in sequential colimits, which figure prominently in the zigzag construction in section 4.2. Sequential colimits and some of their properties may be derived from pushouts. Their construction is more natural if we first formalize a basic theory of coequalizers on top of pushouts, and then we formalize sequential colimits on top of coequalizers.

3.1 Coequalizers

Definition 3.1.1. A **double arrow** is a pair of types A, B, equipped with a pair of maps $f, g : A \to U$.

Definition 3.1.2. Given a double arrow $\mathcal{D} \doteq (f, g)$ and a type $X : \mathcal{U}$, a **cofork** under $\mathcal{D}$ on X is a pair (i, H), where $i : B \to X$ is a map, and H is a homotopy of type $i \circ f \sim i \circ g$.

We write $\mathrm{cofork}(\mathcal{D}, X)$ for the type of coforks under $\mathcal{D}$ on X.

Construction 3.1.3. Given a cofork $c \doteq (i, H) : \mathrm{cofork}(\mathcal{D}, X)$ on X and a type Y, we construct a map

$$\text{cofork-map}_c^Y : (X \to Y) \to \mathrm{cofork}(\mathcal{D}, Y)$$

which sends h to $(h \circ i, h \cdot_l H)$.

Definition 3.1.4. A cofork c under $\mathcal{D}$ on X satisfies the **universal property of coequalizers** if for all types Y the map cofork-map_c^Y is an equivalence.

A cofork satisfying the universal property of coequalizers is called a **coequalizer**.

Construction 3.1.5. Construct the map span-double-arrow from double arrows to span diagrams by

$$A \underset{f}{\overset{g}{\rightrightarrows}} B \qquad \mapsto \qquad A \xleftarrow{\nabla} A + A \xrightarrow{[f,g]} B,$$

where ∇ is the codiagonal map, sending $\mathrm{inl}(a)$ and $\mathrm{inr}(a)$ to a, and the right map is defined by the universal property of coproducts to send $\mathrm{inl}(a)$ to $f(a)$ and $\mathrm{inr}(a)$ to $g(a)$.

The standard coequalizer of a double arrow $\mathcal{D}$ may be obtained as the pushout of span-double-arrow$(\mathcal{D})$.

Lemma 3.1.6. *For any double arrow $\mathcal{D}$ and a type X, there is an equivalence*

$$\text{cocone-cofork} : \text{cofork}(\mathcal{D}, X) \simeq \text{cocone}(\text{span-double-arrow}(\mathcal{D}), X)$$

which fits into the following commuting triangle for every cofork $c : \text{cofork}(\mathcal{D}, X)$

$$\begin{array}{ccc} (X \to Y) & \xrightarrow{\text{cofork-map}_c} & \text{cofork}(\mathcal{D}, Y) \\ & {\scriptstyle \text{cocone-map}_{\text{cocone-cofork}(c)}} \searrow \quad \swarrow {\scriptstyle \simeq \ \text{cocone-cofork}} & \\ & \text{cocone}(\text{span-double-arrow}(\mathcal{D}), Y). & \end{array}$$

Proof. To define the forward map, assume a cofork (j, H), where $j : B \to X$ and $H : j \circ f \sim j \circ g$. To construct the cocone under span-double-arrow$(\mathcal{D})$, take $j \circ f : A \to X$ to be the first component and j to be the second component. It remains to construct a homotopy

$$\begin{array}{ccc} A + A & \xrightarrow{[f,g]} & B \\ {\scriptstyle \nabla}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[j \circ f]{} & X. \end{array}$$

On $\text{inl}(a) : A+A$ the square commutes by refl, and on $\text{inr}(a) : A+A$ it commutes by $H : j \circ f \sim j \circ g$, which we write as $[\text{refl-htpy}, H]$.

To define the inverse map, assume a cocone (i, j, H) where $i : A \to X$, $j : B \to X$, and H witnesses commutativity of the square

$$\begin{array}{ccc} A + A & \xrightarrow{[f,g]} & B \\ {\scriptstyle \nabla}\downarrow & & \downarrow{\scriptstyle j} \\ A & \xrightarrow[i]{} & X. \end{array}$$

Since the codomain of the homotopy is a coproduct, it corresponds to a pair of homotopies $H_1 : i \sim j \circ f$ and $H_2 : i \sim j \circ g$. To construct the cofork under $\mathcal{D}$, take j to be the first component, and the concatenation $H_1^{-1} \bullet_h H_2 : j \circ f \sim j \circ g$ for the second component.

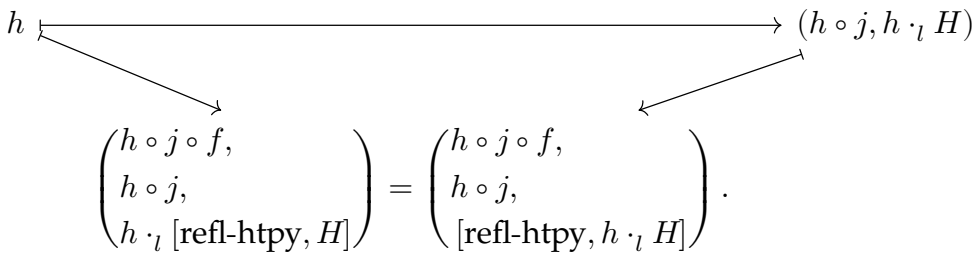
We need to show that the maps are mutual inverses. A cofork (j, H) is first mapped to $(j \circ f, j, [\text{refl-htpy}, H])$ and then to $(j, \text{refl-htpy}^{-1} \bullet_h H)$, so the composition is homotopic to id via refl-htpy. Conversely, assume a cocone $c \doteq (i, j, H)$. It gets mapped to $(j, H_1^{-1} \bullet_h H_2)$ and then to $(j \circ f, j, [\text{refl-htpy}, H_1^{-1} \bullet_h H_2])$, call it c'. We construct a homotopy of cocones $c' \sim c$ — the homotopies on maps are given by

$$\begin{aligned} H_1^{-1} &: (j \circ f) \sim i \\ \text{refl-htpy} &: j \sim j, \end{aligned}$$

and the coherence is $[\alpha_1, \alpha_2]$, where

$$\begin{aligned}\alpha_1 &:= \text{linv-htpy}^{-1} : \text{refl-htpy} \sim (H_1^{-1} \bullet_h H_1)\\ \alpha_2 &:= \text{runit-htpy} \;\; : (H_1^{-1} \bullet_h H_2 \bullet_h \text{refl-htpy}) \sim (H_1^{-1} \bullet_h H_2).\end{aligned}$$

To show commutativity of the triangle, chase a map $h : X \to Y$:



The first two components are identical, so it suffices to show that for all $a : A + A$, the identifications $\text{ap}_h([\text{refl-htpy}, H]\,(a))$ and $\left[\text{refl-htpy}, \text{ap}_h(H(a))\right]$ agree. On elements of the form $\text{inl}(a)$ they both compute to refl, and on elements of the form $\text{inr}(a)$ they compute to $\text{ap}_h(H(a))$, so they are identified by refl. □

Lemma 3.1.7. *The equivalence* cocone-cofork *restricts to an equivalence between coequalizers of $\mathcal{D}$ and pushouts of* span-double-arrow$(\mathcal{D})$. *In other words, a cofork c satisfies the universal property of coequalizers if and only if the cocone* cocone-cofork(c) *satisfies the universal property of pushouts.*

Proof. By the commuting triangle in Lemma 3.1.6 and the 3-for-2 property of equivalences, cofork-map_c is an equivalence if and only if the corresponding cocone-map is an equivalence. □

We define dependent coforks and the dependent universal property of coequalizers analogously to the case of dependent cocones.

Definition 3.1.8. Consider a cofork $c \doteq (i, H) : \text{cofork}(\mathcal{D}, X)$ and a type family $P : X \to \mathcal{U}$. A **dependent cofork** over c on P is a pair (i', H'), where

$$\begin{aligned}i' &\;\; : (b : B) \to P(ib)\\ H' &: (a : A) \to \text{tr}_P(Ha)(i(fa)) = i(ga).\end{aligned}$$

We write $\text{dep-cofork}(c, P)$ for the type of dependent coforks over c on P.

Construction 3.1.9. Given a cofork $c \doteq (i, H) : \text{cofork}(\mathcal{D}, X)$ and a type family $P : X \to \mathcal{U}$, define a map

$$\text{dep-cofork-map}_c^P : ((x : X) \to P(x)) \to \text{dep-cofork}(c, P)$$

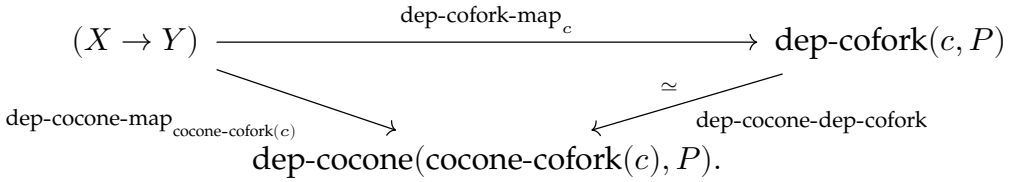
which sends h to $(h \circ i, \lambda a \to \text{apd}_h(Ha))$.

Definition 3.1.10. A cofork c satisfies the **dependent universal property of coequalizers** if for all $P : X \to \mathcal{U}$, the map $\text{dep-cofork-map}_c^P$ is an equivalence.

Lemma 3.1.11. *For any cofork $c : \text{cofork}(\mathcal{D}, X)$, there is an equivalence*

$$\text{dep-cocone-dep-cofork} : \text{dep-cofork}(c, P) \simeq \text{dep-cocone}(\text{cocone-cofork}(c), P)$$

such that the following diagram commutes



Proof. Completely analogous to the proof of Lemma 3.1.6. □

Lemma 3.1.12. *A cofork $c : \text{cofork}(\mathcal{D}, X)$ satisfies the dependent universal property of coequalizers if and only if the cocone* $\text{cocone-cofork}(c)$ *satisfies the dependent universal property of pushouts.*

Proof. By the commuting triangle in Lemma 3.1.11, the map dep-cofork-map_c is an equivalence if and only if $\text{dep-cocone-map}_{\text{cocone-cofork}(c)}$ is an equivalence. □

Theorem 3.1.13. *A cofork satisfies the universal property of coequalizers if and only if it satisfies the dependent universal property of coequalizers.*

Proof. Given a cofork c, there is a sequence of logical equivalences

c satisfies the universal property of coequalizers
$\leftrightarrow$ $\text{cocone-cofork}(c)$ satisfies the universal property of pushouts
$\leftrightarrow$ $\text{cocone-cofork}(c)$ satisfies the dependent universal property of pushouts
$\leftrightarrow$ c satisfies the dependent universal property of coequalizers.

The equivalences are, in order: Lemma 3.1.7, Theorem 2.1.16, and Lemma 3.1.12. □

Coequalizers also satisfy descent, but we do not explore it in the thesis. It is simple enough to derive for different colimits from the univalence axiom. We do, however, prove the flattening lemma for coequalizers, which we will use to prove the flattening lemma for sequential colimits.

Construction 3.1.14. Given a double arrow $\mathcal{D} \doteq (f, g)$, a cofork $c \doteq (i, H)$ on X, and a type family $P : X \to \mathcal{U}$, define the **total cofork** Σc to be

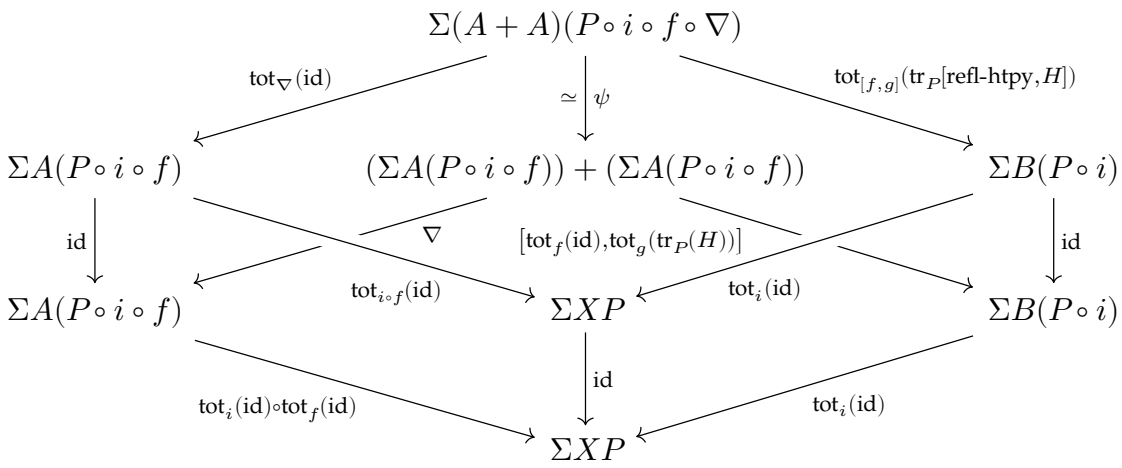
$$\Sigma A(P \circ i \circ f) \underset{\text{tot}_f(\text{id})}{\overset{\text{tot}_g(\text{tr}_P(H))}{\rightrightarrows}} \Sigma B(P \circ i) \xrightarrow{\text{tot}_i(\text{id})} \Sigma X P,$$

where the map $\text{tot}_g(\text{tr}_P(H))$ takes (a, p) to $(ga, \text{tr}_P(Ha, p))$, and the homotopy is

$$(H, \text{refl-htpy}) : (\text{tot}_i(\text{id}) \circ \text{tot}_f(\text{id})) \sim (\text{tot}_i(\text{id}) \circ \text{tot}_g(\text{tr}_P(H))).$$

Theorem 3.1.15 (Flattening lemma for coequalizers)**.** *Given a coequalizer c on X and a type family $P : X \to \mathcal{U}$, the total cofork is also a coequalizer.*

Proof. To show that the total cocone is a coequalizer, it suffices to show that the corresponding cocone is a pushout. Construct the cube



where the homotopy

$$[\text{refl-htpy}, H] : (i \circ f \circ \nabla) \sim (i \circ [f, g])$$

is defined by sending elements $\mathrm{inl}(a)$ to $\mathrm{refl} : i(fa) = i(fa)$ and elements $\mathrm{inr}(a)$ to $H(a) : i(fa) = i(ga)$. The equivalence

$$\psi : (\Sigma(A + A)(P \circ i \circ f \circ \nabla)) \simeq ((\Sigma A(P \circ i \circ f)) + (\Sigma A(P \circ i \circ f)))$$

sends $(\mathrm{inl}(a), p)$ to $\mathrm{inl}(a, p)$ and $(\mathrm{inr}(a), p)$ to $\mathrm{inr}(a, p)$.

The bottom square is the cocone corresponding to the total cofork, cocone-cofork(Σc), and the top square is the total cocone of the corresponding cocone, $\Sigma(\text{cocone-cofork}(c))$. The two front squares commute by refl-htpy, and the back two squares commute by $([\text{refl-htpy}, \text{refl-htpy}], \text{refl-htpy})$.

Since c is a coequalizer by assumption, it holds that the corresponding cocone is a pushout, so by Theorem 2.3.7 the top square of the cube is a pushout. Note that all the vertical maps are equivalences, and the goal is to show that the bottom square is a pushout. Hence it suffices to show that the cube commutes.

To show that the cube commutes, we proceed by cases: for elements of the form $(\mathrm{inl}(a), p)$, all the identifications collapse to refl, so in that fiber the cube commutes by refl. For elements of the form $(\mathrm{inr}(a), p)$, we need an identification

$$\mathrm{refl} \bullet \mathrm{refl} \bullet (\mathrm{ap}_{\mathrm{id}}(H(a), \mathrm{refl})) = (H(a), \mathrm{refl}) \bullet \mathrm{refl} \bullet \mathrm{refl},$$

which we can get by unit laws for concatenating refl's and whiskering by id. □

3.2 Sequential colimits

Sequential colimits are colimits of sequential diagrams, which are sequences of types indexed by natural numbers, equipped with connecting maps between each type A_n and its successor A_{n+1}. In the literature, sequential diagrams are also known as "cotowers". We decided to use "sequential diagrams" in the agda-unimath library, because we believe it to be a more approachable term.

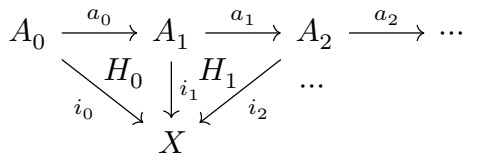
This development formalizes sections 3 and 4 from a paper on treatment of sequential colimits in Homotopy Type Theory, due to Sojakova, van Doorn and Rijke [17], and their main theorem about commutativity of Σ types and sequential colimits, for the special case of equifibered sequential diagrams.

Not all of the built infrastructure is necessary for the zigzag construction, which we are building towards, but the flattening lemma and especially the full "generalized flattening lemma" from [17] are used in the same paper to prove truncatedness and connectivity results, which are used in applications of the zigzag construction.

Definition 3.2.1. A **sequential diagram** is a pair (A, a) of a sequence of types $A : \mathbb{N} \to \mathcal{U}$ and a family of maps $a : (n : \mathbb{N}) \to A(n) \to A(n+1)$.

We will often write A_n and a_n for $A(n)$ and $a(n)$, respectively.

Definition 3.2.2. Given a sequential diagram $\mathcal{A} \doteq (A, a)$ and a type X, a **cocone** under $\mathcal{A}$ on X is a pair (i, H) consisting of a family of maps $i : (n : \mathbb{N}) \to A_n \to X$ and a family of homotopies $H : (n : \mathbb{N}) \to i_n \sim i_{n+1} \circ a_n$.



We write $\mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X)$ for the type of cocones under the sequential diagram $\mathcal{A}$ on the type X.

Remark 3.2.3. In the prose, we use "cocone" to mean both "cocone under a span diagram" and "cocone under a sequential diagram". The kind of diagram should be clear from the context.

Construction 3.2.4. Given a cocone $c \doteq (i, H) : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X)$ and a type Y, we construct the map

$$\mathrm{cocone}_{\mathbb{N}}\text{-}\mathrm{map}_c^Y : (X \to Y) \to \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, Y)$$

which sends h to $(\lambda n \to h \circ i_n, \lambda n \to h \cdot_l H_n)$.

Definition 3.2.5. A cocone $c : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X)$ satisfies the **universal property of sequential colimits** if for all types Y, the map $\mathrm{cocone}_{\mathbb{N}}\text{-}\mathrm{map}_c^Y$ is an equivalence.

A cocone satisfying the universal property of sequential colimits is called a **sequential colimit**.

To provide usable computation rules for maps out of sequential colimits, we introduce homotopies of cocones under sequential diagrams.

Definition 3.2.6. Given a sequential diagram $\mathcal{A} \doteq (A, a)$ and two cocones $c \doteq (i, H)$ and $c' \doteq (i', H')$ on X, a **homotopy** between c and c' is a pair (K, α) consisting of a family of homotopies

$$K : (n : \mathbb{N}) \to i_n \sim i'_n$$

and a family of commuting squares of homotopies, one for each $n : \mathbb{N}$,

$$\begin{array}{ccc} i_n & \overset{K_n}{\rightsquigarrow} & i'_n \\ {\scriptstyle H_n}\big\downarrow & & \big\downarrow{\scriptstyle H'_n} \\ i_{n+1} \circ a_n & \underset{K_{n+1} \cdot_r a_n}{\rightsquigarrow} & i'_{n+1} \circ a_n. \end{array}$$

We write $c \sim c'$ for the type of homotopies between c and c'.

Lemma 3.2.7. *For a sequential diagram $\mathcal{A}$ and two cocones $c, c' : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$, there is an equivalence*

$$\text{htpy-eq-cocone}\mathbb{N} : (c = c') \simeq (c \sim c').$$

Lemma 3.2.8. *Given a sequential diagram $\mathcal{A} \doteq (A, a)$, its sequential colimit $c \doteq (i, H) : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$ and a cocone $c' \doteq (i', H') : \mathrm{cocone}\mathbb{N}(\mathcal{A}, Y)$, there is a unique map $h : X \to Y$ equipped with a family of homotopies*

$$K : (n : \mathbb{N}) \to h \circ i_n \sim i'_n$$

and a family of commuting squares of homotopies, indexed by $n : \mathbb{N}$

$$\begin{array}{ccc} h \circ i_n & \overset{K_n}{\rightsquigarrow} & i'_n \\ {\scriptstyle h \cdot_l H_n}\big\downarrow & & \big\downarrow{\scriptstyle H'} \\ h \circ i_{n+1} \circ a_n & \underset{K_{n+1} \cdot_r a_n}{\rightsquigarrow} & i'_{n+1} \circ a_n. \end{array}$$

Lemma 3.2.9. *Given a sequential diagram $\mathcal{A}$ and its sequential colimit $c : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$, the unique map induced by the universal property of c by the cocone c is the identity map* $\mathrm{id} : X \to X$.

Proof. Write $c \doteq (i, H)$. The map induced by the universal property is the unique map $h : X \to X$ such that there is a homotopy of cocones between cocone$\mathbb{N}$-map$_c(h)$ and c. It then suffices to show that there is a homotopy of cocones between cocone$\mathbb{N}$-map$_c(\mathrm{id})$ and c. The homotopy on maps is satisfied by

$$(\lambda n \to \text{refl-htpy}) : (n : \mathbb{N}) \to (\mathrm{id} \circ i_n) \sim i_n.$$

The coherence

$$\alpha_n : ((\mathrm{id} \cdot_l H_n) \bullet_h \text{refl-htpy}) \sim (\text{refl-htpy} \bullet_h H_n)$$

is provided by a combination of the unit law of left whiskering and right unit law of concatenating homotopies. □

We proceed to build sequential colimits out of coequalizers.

Construction 3.2.10. Construct the map double-arrow-seq from sequential diagrams to double arrows by

$$A_0 \xrightarrow{a_0} A_1 \xrightarrow{a_1} \cdots \qquad \mapsto \qquad \Sigma\mathbb{N}A \underset{\mathrm{id}}{\overset{\mathrm{tot}_{+1}(a_-)}{\rightrightarrows}} \Sigma\mathbb{N}A,$$

where the map $\text{tot}_{+1}(a_-)$ takes (n, x) to $(n+1, a_n(x))$.

The sequential colimit of $\mathcal{A}$ may be obtained as the coequalizer of double-arrow-seq($\mathcal{A}$). Proofs of some of the following lemmas mirror exactly their counterparts in section 3.1, and are therefore omitted.

Lemma 3.2.11. *For any sequential diagram $\mathcal{A}$ and a type X, there is an equivalence*

$$\text{cofork-cocone}\mathbb{N} : \text{cocone}\mathbb{N}(\mathcal{A}, X) \simeq \text{cofork}(\text{double-arrow-seq}(\mathcal{A}), X)$$

which fits into the following commuting triangle for every cocone $c : \text{cocone}\mathbb{N}(\mathcal{A}, X)$

$$\begin{array}{ccc} (X \to Y) & \xrightarrow{\text{cocone}\mathbb{N}\text{-map}_c} & \text{cocone}\mathbb{N}(\mathcal{A}, Y) \\ & {\scriptstyle \text{cofork-map}_{\text{cofork-cocone}\mathbb{N}(c)}} \searrow \quad \swarrow {\scriptstyle \simeq \ \text{cofork-cocone}\mathbb{N}} & \\ & \text{cofork}(\text{double-arrow-seq}(\mathcal{A}), X). & \end{array}$$

Proof. To define the forward map, assume a cocone (i, H) where

$$\begin{aligned} i &: (n : \mathbb{N}) \to A_n \to X \\ H &: (n : \mathbb{N})(x : A_n) \to i_n(x) = i_{n+1}(a_n x). \end{aligned}$$

Uncurrying both components, we get

$$\begin{aligned} \text{ind-}\Sigma(i) &: \Sigma\mathbb{N}A \to X \\ \text{ind-}\Sigma(H) &: \text{ind-}\Sigma(i) \sim \text{ind-}\Sigma(i) \circ \text{tot}_{+1}(a_-), \end{aligned}$$

which is a cofork under double-arrow-seq($\mathcal{A}$).

In reverse, assume a cofork (j, K) where

$$\begin{aligned} j &: \Sigma\mathbb{N}A \to X \\ K &: j \sim j \circ \text{tot}_{+1}(a_-), \end{aligned}$$

and curry both components to get

$$\begin{aligned} \text{ev-pair}(j) &: (n : \mathbb{N}) \to A_n \to X \\ \text{ev-pair}(K) &: (n : \mathbb{N}) \to \text{ev-pair}(j)(n) \sim \text{ev-pair}(j)(n+1) \circ a_n, \end{aligned}$$

which is a cocone under $\mathcal{A}$.

Since currying and uncurrying are judgmental inverses, we conclude that the forward and backward maps are inverses of each other by refl-htpy and refl-htpy.

To prove commutativity of the triangle, assume a cocone (i, H) and compute the action on a map $h : X \to Y$. The resulting coforks we get are

$$\begin{array}{ll} h \circ \text{ind-}\Sigma(i) & \text{ind-}\Sigma(\lambda n \to h \circ i_n) \\ h \cdot_l \text{ind-}\Sigma(H) & \text{ind-}\Sigma(\lambda n \to h \cdot_l H_n) \end{array} \qquad \text{and}$$

which both compute to

$$\begin{aligned} &\lambda(n, x) \to h(i_n x) \\ &\lambda(n, x) \to \text{ap}_h(H_n x), \end{aligned}$$

so the triangle also commutes by refl-htpy. □

Lemma 3.2.12. *A cocone* $c : \text{cocone}\mathbb{N}(\mathcal{A}, X)$ *is a sequential colimit if and only if the cofork* $\text{cofork-cocone}\mathbb{N}(c)$ *is a coequalizer.*

Proof. Omitted. □

Definition 3.2.13. Consider a cocone $c \doteq (i, H) : \text{cocone}\mathbb{N}(\mathcal{A}, X)$ and a type family $P : X \to \mathcal{U}$. A **dependent cocone** over c on P is a pair (i', H') where

$$\begin{aligned} i' &: (n : \mathbb{N})(x : A_n) \to P(i_n x) \\ H' &: (n : \mathbb{N})(x : A_n) \to \text{tr}_P(H_n x, i'_n x) = i'_{n+1}(a_n a). \end{aligned}$$

We write $\text{dep-cocone}\mathbb{N}(c, P)$ for the type of dependent cocones over c on P.

Construction 3.2.14. Given a cofork $c \doteq (i, H) : \text{cocone}\mathbb{N}(\mathcal{A}, X)$ and a type family $P : X \to \mathcal{U}$, construct the map

$$\text{dep-cocone}\mathbb{N}\text{-map}_c^P : ((x : X) \to P(x)) \to \text{dep-cocone}\mathbb{N}(c, P)$$

which sends h to $(\lambda n \to h \circ i_n, \lambda n, x \to \text{apd}_h(H_n x))$.

Definition 3.2.15. A cocone c satisfies the **dependent universal property of sequential colimits** if for all $P : X \to \mathcal{U}$, the map $\text{dep-cocone}\mathbb{N}\text{-map}_c^P$ is an equivalence.

We define homotopies of dependent cocones, because later on in section 4.3 we work with the computation rules of dependent maps from sequential colimits induced by dependent cocones.

Definition 3.2.16. Given a sequential diagram $\mathcal{A} \doteq (A, a)$, a cocone $c \doteq (i, H)$ on X, and two dependent cocones $d \doteq (j, L)$ and $d' \doteq (j', L')$ on P, a **homotopy** between d and d' is a pair (K, α) of a family of homotopies

$$K : (n : \mathbb{N}) \to j_n \sim j'_n$$

and a family of commuting squares of identifications, indexed by $n : \mathbb{N}$ and $x : A_n$

$$\begin{array}{ccc} \text{tr}_P(H_n x)(j_n x) & \xrightarrow{\text{ap}_{\text{tr}_P(H_n x)}(K_n x)} & \text{tr}_P(H_n x)(j'_n x) \\ {\scriptstyle L_n(x)}\Big\| & & \Big\|{\scriptstyle L'_n(x)} \\ j_{n+1}(a_n x) & \xrightarrow[K_{n+1}(a_n x)]{} & j'_{n+1}(a_n x). \end{array}$$

We write $d \sim d'$ for the type of homotopies between d and d'.

Lemma 3.2.17. *For every pair of dependent cocones* $d, d' : \text{dep-cocone}\mathbb{N}(c, P)$*, there is an equivalence*

$$\text{htpy-eq-dep-cocone}\mathbb{N} : (d = d') \simeq (d \sim d').$$

Lemma 3.2.18. *Given a sequential diagram* $\mathcal{A} \doteq (A, a)$*, its sequential colimit* $c \doteq (i, H)$ *on* X*, and a dependent cocone* $d \doteq (i', H') : \text{dep-cocone}\mathbb{N}(c, P)$*, there is a unique dependent map* $h : (x : X) \to P(x)$ *equipped with a family of homotopies*

$$K : (n : \mathbb{N}) \to h \circ i_n \sim i'_n$$

and a family of commuting squares of identifications indexed by $n : \mathbb{N}$ *and* $x : A_n$

$$\begin{array}{ccc} \mathrm{tr}_P(H_n x)(h(i_n x)) & \overset{\mathrm{ap}_{\mathrm{tr}_P(H_n x)}(K_n x)}{=\!=\!=} & \mathrm{tr}_P(H_n x)(i'_n x) \\ {\scriptstyle \mathrm{apd}_h(H_n x)}\Big\| & & \Big\| {\scriptstyle H'(x)} \\ h(i_{n+1}(a_n x)) & \underset{K_{n+1}(a_n x)}{=\!=\!=} & i'_{n+1}(a_n x) \end{array}$$

Lemma 3.2.19. *For any cocone $c : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$, there is an equivalence*

$$\begin{aligned} \text{dep-cofork-dep-cocone}\mathbb{N} : \text{dep-cocone}\mathbb{N}(c, P) \\ \simeq \text{dep-cofork}(\text{cofork-cocone}\mathbb{N}(c), P) \end{aligned}$$

such that the following diagram commutes

$$\begin{array}{ccc} ((x : X) \to P(x)) & \xrightarrow{\text{dep-cocone}\mathbb{N}\text{-map}_c} & \text{dep-cocone}\mathbb{N}(c, P) \\ & {\scriptstyle \text{dep-cofork-map}_{\text{cofork-cocone}\mathbb{N}(c)}} \searrow \quad \swarrow {\scriptstyle \text{dep-cofork-dep-cocone}\mathbb{N}} & \\ & \text{dep-cofork}(\text{cofork-cocone}\mathbb{N}(c), P). & \end{array}$$

Proof. Omitted. □

Lemma 3.2.20. *A cocone $c : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$ satisfies the dependent universal property of sequential colimits if and only if the cofork* $\text{cofork-cocone}\mathbb{N}(c)$ *satisfies the dependent universal property of coequalizers.*

Proof. Omitted. □

Theorem 3.2.21. *A cocone $c : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$ satisfies the universal property of sequential colimits if and only if it satisfies the dependent universal property of sequential colimits.*

Proof. Omitted. □

3.2.1 Functoriality

Uniformly constructing a sequential colimit of every sequential diagram amounts to having a map from the type of sequential diagrams to the type of types equipped with the structure of a sequential colimit on it. We show that this action on objects, taking a sequential diagram to a type, extends to an action on morphisms, which takes a morphism of sequential diagrams to a map between the corresponding types. Additionally, this action on morphisms is functorial, i.e. it takes the identity morphism to the identity map and composition of morphisms to composition of maps.

To formally state this property, we first need to define morphisms of sequential diagrams and their composition. We also show that homotopies of morphisms of sequential diagrams induce homotopies of the appropriate maps between colimits.

The theory does not assume a uniform construction of standard sequential colimits. Instead the constructions and proofs are parametric over a user-provided sequential colimit. This generality is important for later applications in section 4.1, where the colimit is not judgmentally equal to the standard one.

The presented results cover Lemma 3.5 from [17], except preservation of equivalences. That result is part of the attached formalization, but not included in the thesis text. Likewise, elementary theory of commuting prisms of maps was introduced for the formalization, but not described in this text.

Definition 3.2.22. Given sequential diagrams (A, a) and (B, b), define the type of **morphisms** from (A, a) to (B, b), denoted $(A, a) \to (B, b)$, as the type of pairs (f, H) consisting of a family of maps

$$f : (n : \mathbb{N}) \to A_n \to B_n$$

and a family of homotopies witnessing that the following squares of maps, indexed by $n : \mathbb{N}$, commute

$$\begin{array}{ccc} A_n & \xrightarrow{a_n} & A_{n+1} \\ {\scriptstyle f_n}\downarrow & H_n & \downarrow{\scriptstyle f_{n+1}} \\ B_n & \xrightarrow[b_n]{} & B_{n+1}. \end{array}$$

All sequential diagrams come equipped with an identity morphism.

Construction 3.2.23. Given a sequential diagram (A, a), construct the **identity morphism** $(A, a) \to (A, a)$ consisting of the data

$$\begin{aligned} (\lambda n \to \text{id}) &: (n : \mathbb{N}) \to A_n \to A_n \\ (\lambda n \to \text{refl-htpy}) &: (n : \mathbb{N}) \to a_n \sim a_n. \end{aligned}$$

Morphisms can be composed.

Construction 3.2.24. Given sequential diagrams (A, a), (B, b) and (C, c), and morphisms

$$\begin{aligned} F \doteq (f, H) &: (A, a) \to (B, b) \\ G \doteq (g, K) &: (B, b) \to (C, c), \end{aligned}$$

construct the **composed morphism** $G \circ F : (A, a) \to (C, c)$ by function composition

$$(\lambda n \to g_n \circ f_n) : (n : \mathbb{N}) \to A_n \to C_n$$

and pasting of commuting squares

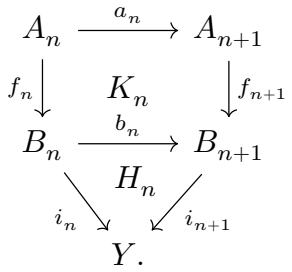
$$\begin{array}{ccc} A_n & \xrightarrow{a_n} & A_{n+1} \\ {\scriptstyle f_n}\downarrow & H_n & \downarrow{\scriptstyle f_{n+1}} \\ B_n & \xrightarrow[b_n]{} & B_{n+1} \\ {\scriptstyle g_n}\downarrow & K_n & \downarrow{\scriptstyle g_{n+1}} \\ C_n & \xrightarrow[c_n]{} & C_{n+1}. \end{array}$$

To construct a map $X \to Y$ between sequential colimits, we can use the universal property of X. That requires us to construct a cocone under X's diagram on Y.

Construction 3.2.25. Given a sequential diagram $\mathcal{B}$ and a cocone $c \doteq (i, H)$ on Y, define for every sequential diagram $\mathcal{A}$ the map

$$\text{precomp-hom}\mathbb{N}^{\mathcal{A}}_{c} : (\mathcal{A} \to \mathcal{B}) \to \text{cocone}\mathbb{N}(\mathcal{A}, Y)$$

which sends a morphism (f, K) to the cocone



Remark 3.2.26. This construction is in a sense dual to cocone$\mathbb{N}$-map — the function cocone$\mathbb{N}$-map extends a cocone by postcomposing a map $X \to Y$ on the right, and precomp-hom$\mathbb{N}$ extends a cocone by "precomposing" a morphism $\mathcal{B} \to \mathcal{A}$ on the left.

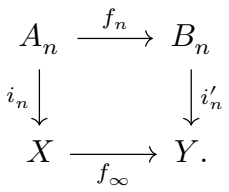
Construction 3.2.27. Given sequential diagrams $\mathcal{A}$ and $\mathcal{B}$, a sequential colimit $c : \text{cocone}\mathbb{N}(\mathcal{A}, X)$, and a cocone $c' : \text{cocone}\mathbb{N}(\mathcal{B}, Y)$, construct the map

$$\text{fmap-hom}\mathbb{N} : (\mathcal{A} \to \mathcal{B}) \to (X \to Y)$$

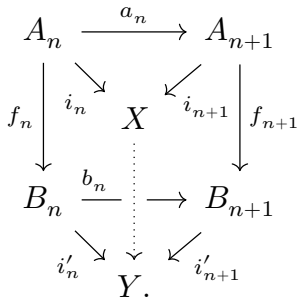
using the universal property of c, as the map taking a morphism $f : \mathcal{A} \to \mathcal{B}$ to the unique map induced by the cocone $\text{precomp-hom}\mathbb{N}_{c'}(f) : \text{cocone}(\mathcal{A}, Y)$.

We often write $f_\infty : X \to Y$ for the map induced by a morphism $f : \mathcal{A} \to \mathcal{B}$.

Lemma 3.2.28. *The map f_∞ fits into commuting squares*



which in turn fit into commuting prisms



Proof. The data is obtained from the computation rules stated in Lemma 3.2.8. The commuting squares are kept as-is, which causes the unexpected change of orientation — the computation rules provide a homotopy between the cocones $\text{cocone}\mathbb{N}\text{-map}_c(f_\infty)$ and $\text{precomp-hom}\mathbb{N}(f)$, not the other way around.

The type of prisms as above is equivalent to the type of coherences of homotopies $\text{cocone}\mathbb{N}\text{-map}_c(f_\infty) \sim \text{precomp-hom}\mathbb{N}(f)$ by mechanical homotopy algebra. □

Lemma 3.2.29. *The map* $\text{fmap-hom}\mathbb{N}$ *preserves identity morphisms. That is to say, given a sequential diagram* $\mathcal{A}$ *and its colimit* X*, the identity morphism* $\text{id} : \mathcal{A} \to \mathcal{A}$ *induces the map* $\text{id}_\infty : X \to X$*, which is homotopic to the identity map* $\text{id} : X \to X$*.*

Proof. By Lemma 3.2.8, the map id_∞ is the unique map such that the cocone $\text{cocone}\mathbb{N}\text{-map}(\text{id}_\infty)$ is homotopic to the cocone $\text{precomp-hom}\mathbb{N}(\text{id})$. Hence to show that $\text{id}_\infty \sim \text{id}$, it suffices to show that $\text{cocone}\mathbb{N}\text{-map}(\text{id})$ is homotopic to $\text{precomp-hom}\mathbb{N}(\text{id})$. In other words, the goal is to provide a homotopy

$$\begin{array}{ccc} A_n & \xrightarrow{a_n} & A_{n+1} \\ & {\scriptstyle i_n}\searrow \quad \swarrow{\scriptstyle i_{n+1}} & \\ & X & \\ & \downarrow{\scriptstyle \text{id}} & \\ & X & \end{array} \quad \sim \quad \begin{array}{ccc} A_n & \xrightarrow{a_n} & A_{n+1} \\ {\scriptstyle \text{id}}\downarrow & & \downarrow{\scriptstyle \text{id}} \\ A_n & \xrightarrow{a_n} & A_{n+1} \\ & {\scriptstyle i_n}\searrow \quad \swarrow{\scriptstyle i_{n+1}} & \\ & X. & \end{array}$$

The homotopy on maps is satisfied by $\text{refl-htpy} : i_n \sim i_n$, and for coherences we need to give

$$\alpha_n : ((\text{id} \cdot_l H_n) \bullet_h \text{refl-htpy}) \sim (H_n \bullet_h \text{refl-htpy}),$$

which follows from the left unit law of whiskering by id. □

Lemma 3.2.30. *The map* $\text{fmap-hom}\mathbb{N}$ *preserves composition, in the sense that for morphisms* $f : \mathcal{A} \to \mathcal{B}$ *and* $g : \mathcal{B} \to \mathcal{C}$*, colimits* $c \doteq (i, H) : \text{cocone}\mathbb{N}(\mathcal{A}, X)$ *and* $c' \doteq (i', H') : \text{cocone}\mathbb{N}(\mathcal{B}, Y)$ *and a cocone* $c'' \doteq (i'', H'') : \text{cocone}\mathbb{N}(\mathcal{C}, Z)$*, there is a homotopy* $(g \circ f)_\infty \sim (g_\infty \circ f_\infty)$*.*

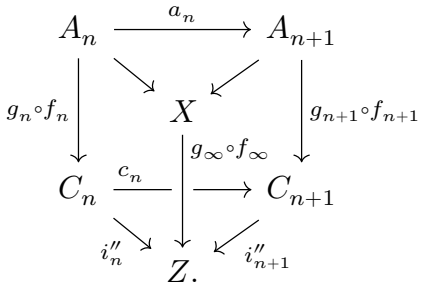
Proof. As in the identity case, it suffices to give a homotopy of cocones

$$\text{cocone}\mathbb{N}\text{-map}(g_\infty \circ f_\infty) \sim \text{precomp-hom}\mathbb{N}(g \circ f).$$

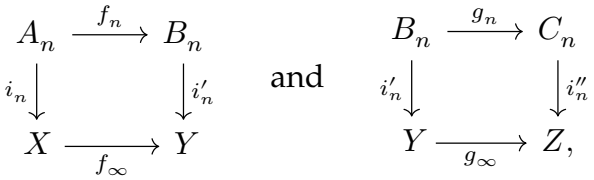
This is equivalent to providing a family of commuting squares

$$\begin{array}{ccc} A_n & \xrightarrow{g_n \circ f_n} & C_n \\ {\scriptstyle i_n}\downarrow & & \downarrow{\scriptstyle i''_n} \\ X & \xrightarrow[g_\infty \circ f_\infty]{} & Z \end{array}$$

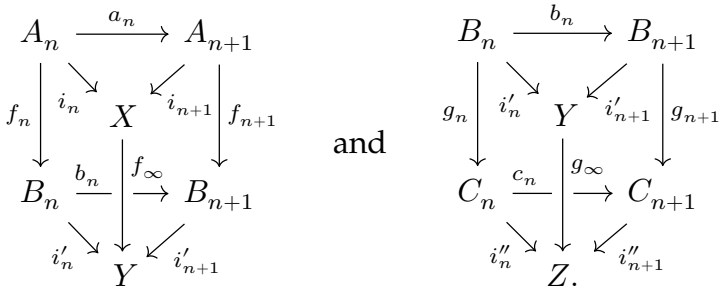
and fitting them into a family of commuting prisms



Since f_∞ and g_∞ are both constructed from morphisms of sequential diagrams, they come equipped with their respective homotopies



and the prisms



Putting the squares side-by-side and stacking the prisms atop each other gives the desired homotopy. □

The last property we will need is that taking a sequential colimit also extends to an action on homotopies.

Definition 3.2.31. Given two sequential diagrams $\mathcal{A} \doteq (A, a)$ and $\mathcal{B} \doteq (B, b)$, and two morphisms $f \doteq (i, H), g \doteq (i', H') : \mathcal{A} \to \mathcal{B}$, a **homotopy** between f and g is a pair (K, α) consisting of a family of homotopies

$$K : (n : \mathbb{N}) \to i_n \sim i'_n$$

and a family of commuting squares of homotopies indexed by $n : \mathbb{N}$

$$\begin{array}{ccc} b_n \circ i_n & \overset{b_n \cdot_l K_n}{\sim\sim\sim\sim} & b_n \circ i'_n \\ {\scriptstyle H_n}\big\| & & \big\|{\scriptstyle H'_n} \\ i_{n+1} \circ a_n & \underset{K_{n+1} \cdot_r a_n}{\sim\sim\sim\sim} & i'_{n+1} \circ a_n. \end{array}$$

We write $f \sim g$ for the type of homotopies between f and g.

Lemma 3.2.32. *For any two morphisms of sequential diagrams $f, g : \mathcal{A} \to \mathcal{B}$, there is an equivalence*

$$\text{htpy-eq-hom}\mathbb{N} : (f = g) \simeq (f \sim g).$$

Lemma 3.2.33. *Taking sequential colimits of sequential diagrams preserves homotopies. Specifically, given sequential diagrams $\mathcal{A}, \mathcal{B}$ and morphisms $f, g : \mathcal{A} \to \mathcal{B}$, there is a map*

$$\text{hmap-hom}_{\mathbb{N}} : (f \sim g) \to (f_\infty \sim g_\infty).$$

Proof. Turn the homotopy $H : f \sim g$ into an identification of morphisms of sequential diagrams $H' : f = g$, apply fmap-hom$_{\mathbb{N}}$ on the identification to get $H'' : f_\infty = g_\infty$, which induces a homotopy of type $f_\infty \sim g_\infty$. □

3.2.2 Colimits of shifted sequential diagrams

Sequential diagrams consist of an infinite amount of data, represented by an infinite sequence of types and maps between them. It is natural to ask how much individual vertices of that sequence influence the resulting colimit, and one might expect that removing a vertex from the sequence does not change the colimit at all. That is in fact true for any finite amount of vertices removed from the sequence. Here we limit ourselves to removing vertices from the beginning of the sequence, which is described by an operation called "shifting".

A shift of a sequential diagram $\mathcal{A}$ is the sequential diagram consisting of the types and maps shifted by one to the left. It is denoted $\mathcal{A}[1]$. This shifting can be iterated for any natural number k; then the resulting sequential diagram is denoted $\mathcal{A}[k]$.

Similarly, a shift of a morphism of sequential diagrams is a morphism from the shifted domain into the shifted codomain. In symbols, given a morphism $f : \mathcal{A} \to \mathcal{B}$, we have $f[k] : \mathcal{A}[k] \to \mathcal{B}[k]$.

We also define shifts of cocones and homotopies of cocones, which can additionally be "unshifted".

Importantly the type of cocones under a sequential diagram is equivalent to the type of cocones under its shift, as we will show by proving that shifting and unshifting are inverse operations. It follows that the sequential colimit of a shifted sequential diagram is equivalent to the colimit of the original diagram.

In the later chapters we only ever need to shift by one, but arbitrary shifts are used in the statement and proof of the main theorem of Sojakova; van Doorn; Rijke [17], which they use to prove connectivity and truncation results for sequential colimits, which in turn is necessary for proving some of the applications of the zigzag construction of identity types of pushouts, studied by Wärn [19].

Construction 3.2.34. Given a sequential diagram $\mathcal{A} \doteq (A, a)$, construct its **shift** by one as the diagram

$$A_1 \xrightarrow{a_1} A_2 \xrightarrow{a_2} \cdots.$$

Call this $\mathcal{A}[1]$.

Then construct arbitrary shifts by induction

$$\begin{aligned} \mathcal{A}[0] &:= \mathcal{A} \\ \mathcal{A}[k+1] &:= (\mathcal{A}[k])[1]. \end{aligned}$$

Remark 3.2.35. The constructions of shifts are defined by first defining a shift by one, and then recursively shifting by one according to the argument k. An alternative would be to shift all data using addition on the natural numbers.

However, in our setting addition computes only on one side, so we have a choice to make: given a number k to shift by, do we define the n-th level of the shifted structure to be the $(n+k)$-th, or $(k+n)$-th level of the original?

The former runs into issues already when defining the shifted sequence, since a_{n+k} has the type $A_{n+k} \to A_{(n+k)+1}$, but we need a function of type $A_{n+k} \to A_{(n+1)+k}$, which forces us to introduce a transport.

On the other hand, the latter requires transport when proving anything by induction on k and does not satisfy the judgmental equality $\mathcal{A}[0] \doteq \mathcal{A}$, because $A_{(k+1)+n}$ is not $A_{(k+n)+1}$ and A_{0+n} is not A_n, and it requires more infrastructure for working with horizontal compositions in sequential diagrams to be formalized in terms of addition.

To contrast, defining the operations by induction does satisfy $\mathcal{A}[0] \doteq \mathcal{A}$, it computes when proving properties by induction, which is the expected primary use-case, and no further infrastructure is necessary.

Construction 3.2.36. Given sequential diagrams $\mathcal{A}$, $\mathcal{B}$ and a morphism between them $F \doteq (f, H) : \mathcal{A} \to \mathcal{B}$, construct its **shift** by one as the morphism

$$\begin{array}{ccccc} A_1 & \xrightarrow{a_1} & A_2 & \xrightarrow{a_2} & \cdots \\ {\scriptstyle f_1}\downarrow & H_1 & \downarrow{\scriptstyle f_2} & & \cdots \\ B_1 & \xrightarrow[b_1]{} & B_2 & \xrightarrow[b_2]{} & \cdots, \end{array}$$

and call it $F[1] : \mathcal{A}[1] \to \mathcal{B}[1]$.

Construct further shifts by induction

$$\begin{aligned} F[0] &:\equiv F && : \mathcal{A}[0] \to \mathcal{B}[0] \\ F[k+1] &:\equiv (F[k])[1] && : \mathcal{A}[k+1] \to \mathcal{B}[k+1]. \end{aligned}$$

Construction 3.2.37. Given a cocone $c : \mathsf{cocone}_{\mathbb{N}}(\mathcal{A}, X)$, construct its **shift** by one as the cocone

$$\begin{array}{ccccccc} A_1 & \xrightarrow{a_1} & A_2 & \xrightarrow{a_2} & A_3 & \xrightarrow{a_3} & \cdots \\ & {\scriptstyle i_1}\searrow & H_1 \downarrow{\scriptstyle i_2} & & H_2 \swarrow{\scriptstyle i_3} & \cdots & \\ & & X, & & & & \end{array}$$

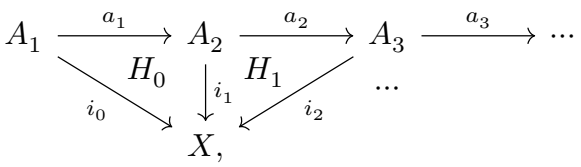
call it $c[1] : \mathsf{cocone}_{\mathbb{N}}(\mathcal{A}[1], X)$.

Construct arbitrary shifts by induction

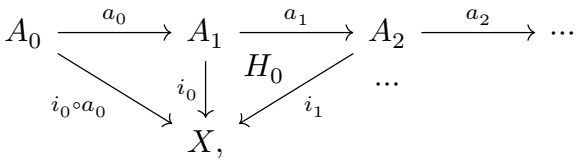
$$\begin{aligned} c[0] &:\equiv c && : \mathsf{cocone}_{\mathbb{N}}(\mathcal{A}[0], X) \\ c[k+1] &:\equiv (c[k])[1] && : \mathsf{cocone}_{\mathbb{N}}(\mathcal{A}[k+1], X). \end{aligned}$$

Cocones can be "unshifted" as well. If shifting corresponds to forgetting data, then unshifting corresponds to recovering forgotten data. The data can be recovered, because the missing data has the type $\Sigma(i : A_0 \to X).\, (i \sim i_0 \circ a_0)$, which is contractible at $(i_0 \circ a_0, \text{refl-htpy})$.

Construction 3.2.38. Given a cocone $c : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}[1], X)$, i.e. a diagram with the shape



construct its **unshift** by one as the cocone



where the left triangle is refl-htpy $: (i_0 \circ a_0) \sim (i_0 \circ a_0)$, and denote it $c[-1] : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X)$.

Then inductively define arbitrary unshifts

$$\begin{aligned} c[-0] &:= c && : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X) \quad \text{for } c : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}[0], X) \\ c[-(k+1)] &:= c[-1][-k] && : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X) \quad \text{for } c : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}[k+1], X). \end{aligned}$$

Remark 3.2.39. One might expect that, following the pattern of shifts, the inductive case should be $c[-k][-1]$. Note, however, that the construction only provides a way to unshift a cocone under $\mathcal{A}[n]$ by n; since the cocone c in the inductive case is under $\mathcal{A}[k][1]$, we first need to unshift by 1 to get $c[-1]$ under $\mathcal{A}[k]$, and only then we can unshift by k to get $c[-1][-k]$ under $\mathcal{A}$.

Shifting and unshifting homotopies will also be required to show that shifting and unshifting cocones are inverses to each other.

Construction 3.2.40. Given cocones $c \doteq (i, H)$ and $c' \doteq (i', H')$ under $\mathcal{A}$ on X, and a homotopy $\mathrm{K} \doteq (K, \alpha) : c \sim c'$, construct the **shift** by one of K to be the homotopy between $c[1]$ and $c'[1]$ consisting of (K', α'), where

$$\begin{aligned} K' &:= (\lambda n \to K_{n+1}) : i_{n+1} \sim i'_{n+1} \\ \alpha' &:= (\lambda n \to \alpha_{n+1}) \ : (H_{n+1} \bullet_h (K_{n+2} \cdot_r a_{n+1})) \sim (K_{n+1} \bullet_h H'_{n+1}). \end{aligned}$$

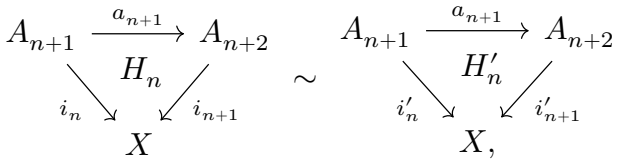
Denote it by $\mathrm{K}[1]$.

Then define other shifts by induction

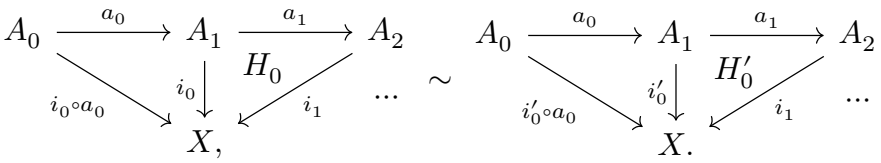
$$\begin{aligned} \mathrm{K}[0] &:= \mathrm{K} && : c[0] \sim c'[0] \\ \mathrm{K}[k+1] &:= (\mathrm{K}[k])[1] && : c[k+1] \sim c'[k+1]. \end{aligned}$$

Similarly to unshifting cocones, we can recover the first homotopy and coherence to unshift a homotopy of cocones.

Construction 3.2.41. Given cocones $c \doteq (i, H)$ and $c' \doteq (i', H')$ under $\mathcal{A}[1]$ on X, and a homotopy $\mathrm{K} \doteq (K, \alpha) : c \sim c'$, we construct the **unshift** by one of K, which is a homotopy $\mathrm{K}[-1] : c[-1] \sim c'[-1]$. The input data has the form



which we need to turn into a homotopy



Define $\mathrm{K}[-1] \doteq (K', \alpha')$ by case splitting on the index

$$\begin{aligned}
&K'_0 \coloneqq K_0 \cdot_r a_0 && : (i_0 \circ a_0) \sim (i'_0 \circ a_0)\\
&K'_{n+1} \coloneqq K_n && : i_n \sim i'_n\\
&\alpha'_0 \coloneqq \text{runit-htpy}^{-1} && : (\text{refl-htpy} \bullet_h (K_0 \cdot_r a_0)) \sim ((K_0 \cdot_r a_0) \bullet_h \text{refl-htpy})\\
&\alpha'_{n+1} \coloneqq \alpha_n && : (H_n \bullet_h (K_{n+1} \cdot_r a_n)) \sim (K_n \bullet_h H'_n).
\end{aligned}$$

Then define arbitrary unshifts by induction on k

$$\begin{aligned}
&\mathrm{K}[0] \coloneqq \mathrm{K} && : c[-0] \sim c'[-0]\\
&\mathrm{K}[-(k+1)] \coloneqq (\mathrm{K}[-1])[-k] && : c[-(k+1)] \sim c'[-(k+1)].
\end{aligned}$$

Lemma 3.2.42. *For every sequential diagram $\mathcal{A}$ and a natural number k, the map*

$$(-)[k] : \mathrm{cocone}^{\mathbb{N}}(\mathcal{A}, X) \to \mathrm{cocone}^{\mathbb{N}}(\mathcal{A}[k], X)$$

is an equivalence, with the inverse

$$(-)[-k] : \mathrm{cocone}^{\mathbb{N}}(\mathcal{A}[k], X) \to \mathrm{cocone}^{\mathbb{N}}(\mathcal{A}, X)$$

Proof. The goal is to show that for any k, we have $c[-k][k] = c$ and $c[k][-k] = c$, for appropriately typed cocones c.

First note that for any cocone $c : \mathrm{cocone}(\mathcal{A}[1], X)$, $c[-1][1]$ computes to the cocone c, because $c[-1]$ is the cocone c with synthesized data at the front, and $c[-1][1]$ forgets the new data. Inductively, we define the homotopy $c[-k][k] \sim c$ for all k. We have $c[-0][0] \sim c$ by the reflexive homotopy, and

$$\begin{aligned}
c[-(k+1)][k+1] &\doteq c[-1][-k][k][1]\\
&\sim c[-1][1] && \text{by shifting the induction hypothesis}\\
& && \mathrm{IH} : (c[-1])[-k][k] \sim c[-1]\\
&\doteq c.
\end{aligned}$$

Since homotopies of cocones characterize their identity types, we obtain the desired identifications $c[-k][k] = c$.

For the other direction, we begin by giving a homotopy $c \sim c[1][-1]$ for every cocone $c : \mathrm{cocone}^{\mathbb{N}}(\mathcal{A}, X)$. We choose this orientation of the homotopy, because

the first component of the homotopy now needs a proof of $i_0 \sim i_1 \circ a_0$, which we can supply directly by H_0. Define the homotopy of cocones by case splitting as

$$\begin{aligned}
K_0 &:= H_0 && : i_0 \sim i_1 \circ a_n \\
K_{n+1} &:= \text{refl-htpy} && : i_{n+1} \sim i_{n+1} \\
\alpha_0 &:= \text{refl-htpy} && : (H_0 \bullet_h \text{refl-htpy}) \sim (H_0 \bullet_h \text{refl-htpy}) \\
\alpha_{n+1} &:= \text{runit-htpy} && : (H_{n+1} \bullet_h \text{refl-htpy}) \sim (\text{refl-htpy} \bullet_h H_{n+1}).
\end{aligned}$$

Then extend the homotopy by induction to all k. We again have $c \sim c[0][-0]$ by the reflexivity homotopy, and in the inductive case we compose the homotopies

$$\begin{aligned}
c &\sim c[k][-k] && \text{by the inductive hypothesis} \\
&\sim c[k][1][-1][-k] && \text{by unshifting the homotopy } c[k] \sim c[k][1][-1] \text{ by } k \\
&\doteq c[k+1][-(k+1)].
\end{aligned}$$

This family of homotopies can be made into a family of identifications, and inverted to get the required $c[k][-k] = c$. □

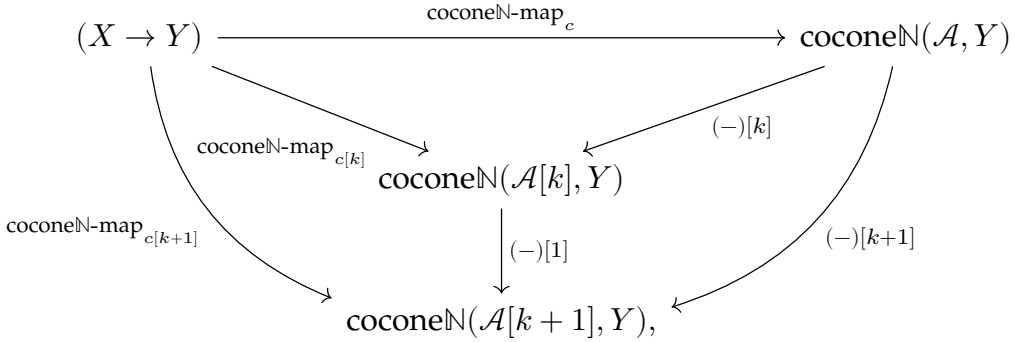
Theorem 3.2.43. *Given a sequential diagram $\mathcal{A}$ and its colimit $c : \text{cocone}^{\mathbb{N}}(\mathcal{A}, X)$, the cocone $c[k] : \text{cocone}^{\mathbb{N}}(\mathcal{A}[k], X)$ is a sequential colimit of the diagram $\mathcal{A}[k]$, for any natural number k.*

Proof. We construct a commuting triangle

$$\begin{array}{ccc}
(X \to Y) & \xrightarrow[\simeq]{\text{cocone}^{\mathbb{N}}\text{-map}_c} & \text{cocone}^{\mathbb{N}}(\mathcal{A}, Y) \\
& {\scriptstyle \text{cocone}^{\mathbb{N}}\text{-map}_{c[k]}} \searrow \quad \swarrow {\scriptstyle (-)[k]} \; \simeq & \\
& \text{cocone}^{\mathbb{N}}(\mathcal{A}[k], Y), &
\end{array}$$

where the right map is an equivalence by Lemma 3.2.42, and the top map is an equivalence by assumption. The it follows that the left map is an equivalence.

The triangle is constructed by induction. Note that it commutes by refl-htpy for the case $k = 0$, since then the right map is an identity and the cocone maps are the same map, and also for the case $k = 1$, since then both paths map a function $h : X \to Y$ to the cocone $(\lambda n \to h \circ i_n, \lambda n \to h \cdot_l H_n)$. To show that the triangle commutes for $k + 1$, compose it out of the smaller triangles



where the top one is the induction hypothesis, the left one is the case for $k = 1$, and the right one is the definition of $(-)[k + 1]$. □

To conclude this section, we show that there are inclusion morphisms of sequential diagrams $\mathcal{A} \to \mathcal{A}[k]$, which induce the identity map on the colimit.

Construction 3.2.44. Given a sequential diagram $\mathcal{A} \doteq (A, a)$, construct the morphism of sequential diagrams

$$\text{incl-hom}\mathbb{N}[1] : \mathcal{A} \to \mathcal{A}[1]$$

as the morphism

$$\begin{aligned} f_n &:= a_n && : A_n \to A_{n+1} \\ H_n &:= \text{refl-htpy} && : (a_{n+1} \circ a_n) \sim (a_{n+1} \circ a_n). \end{aligned}$$

Diagrammatically, the morphism can be drawn as

$$\begin{array}{ccccc} A_0 & \xrightarrow{a_0} & A_1 & \xrightarrow{a_1} & \cdots \\ {\scriptstyle a_0}\downarrow & & \downarrow{\scriptstyle a_1} & & \cdots \\ A_1 & \xrightarrow[a_1]{} & A_2 & \xrightarrow[a_2]{} & \cdots. \end{array}$$

Extend the morphism to

$$\text{incl-hom}\mathbb{N}[k] : \mathcal{A} \to \mathcal{A}[k]$$

for any natural number k by induction, where $\text{incl-hom}\mathbb{N}[0] : \mathcal{A} \to \mathcal{A}[0]$ is the identity morphism, and $\text{incl-hom}\mathbb{N}[k+1] : \mathcal{A} \to \mathcal{A}[k+1]$ is the composition of $\text{incl-hom}\mathbb{N}[k] : \mathcal{A} \to \mathcal{A}[k]$ and then $\text{incl-hom}\mathbb{N}[1] : \mathcal{A}[k] \to \mathcal{A}[k][1]$.

These morphisms offer another way of unshifting cocones. Specifically, one can precompose a cocone $c : \text{cocone}\mathbb{N}(\mathcal{A}[k], X)$ with the above morphism $\text{incl-hom}\mathbb{N}[k] : \mathcal{A} \to \mathcal{A}[k]$ to get a cocone $c' : \text{cocone}\mathbb{N}(\mathcal{A}, X)$. We show that these two constructions result in homotopic cocones. We limit ourselves to the case $k = 1$, as we do not need the general case in further development.

Lemma 3.2.45. *Given a cocone $c \doteq (i, H) : \text{cocone}\mathbb{N}(\mathcal{A}[1], X)$, there is a homotopy of cocones*

$$c[-1] \sim \text{precomp-hom}\mathbb{N}^{\mathcal{A}}_{c}(\text{incl-hom}\mathbb{N}[1]).$$

Proof. We need to show a homotopy between the cocones

$$\begin{array}{ccccccc} A_0 & \xrightarrow{a_0} & A_1 & \xrightarrow{a_1} & A_2 & & \\ & {\scriptstyle i_0 \circ a_0}\searrow & \downarrow{\scriptstyle i_0} & \swarrow{\scriptstyle i_1} & \cdots & & \\ & & X & & & & \end{array} \quad \text{and} \quad \begin{array}{ccccc} A_n & \xrightarrow{a_n} & A_{n+1} & & \\ {\scriptstyle a_n}\downarrow & & \downarrow{\scriptstyle a_{n+1}} & & \\ A_{n+1} & \xrightarrow{a_{n+1}} & A_{n+2} & & \\ & {\scriptstyle i_n}\searrow & \swarrow{\scriptstyle i_{n+1}} & & \\ & & X. & & \end{array}$$

This homotopy can be constructed by induction on n as

$$\begin{aligned} K_0 &:= \text{refl-htpy} && : (i_0 \circ a_0) \sim (i_0 \circ a_0) \\ K_{n+1} &:= H_n && : i_n \sim (i_{n+1} \circ a_{n+1}) \\ \alpha_0 &:= \text{runit-htpy}^{-1} && : (\text{refl-htpy} \bullet_h H_0) \sim (\text{refl-htpy} \bullet_h H_n \bullet_h \text{refl-htpy}) \\ \alpha_{n+1} &:= H_n \cdot_l \text{runit-htpy}^{-1} && : (H_n \bullet_h (H_{n+1} \cdot_r a_{n+1})) \\ & && \sim (H_n \bullet_h (H_{n+1} \cdot_r a_{n+1}) \bullet_h \text{refl-htpy}). \end{aligned}$$

□

Corollary 3.2.46. *For any cocone $c : \text{cocone}_{\mathbb{N}}(\mathcal{A}, X)$, there is a homotopy*

$$c \sim \text{precomp-hom}_{\mathbb{N}}{}_{c[1]}(\text{incl-hom}_{\mathbb{N}}[1]).$$

Proof. Compose the homotopies $c \sim c[1][-1]$ from Lemma 3.2.42 and $c[1][-1] \sim \text{precomp-hom}_{\mathbb{N}}{}_{c[1]}(\text{incl-hom}_{\mathbb{N}}[1])$ from the above lemma applied to $c[1]$. □

Lemma 3.2.47. *Assume a sequential diagram $\mathcal{A}$ with its colimit $c : \text{cocone}_{\mathbb{N}}(\mathcal{A}, X)$. Then for any natural number k, the morphism $\text{incl-hom}_{\mathbb{N}}[k] : \mathcal{A} \to \mathcal{A}[k]$ induces a map out of X. If we consider $c[k] : \text{cocone}_{\mathbb{N}}(\mathcal{A}[k], X)$ as the colimit of $\mathcal{A}[k]$ by Theorem 3.2.43, this map's codomain is X. The induced map $\text{incl-hom}_{\mathbb{N}}[k]_\infty : X \to X$ is homotopic to the identity map $\text{id} : X \to X$.*

Proof. Proceed by induction on k. For $k = 0$, note that the morphism $\text{incl-hom}_{\mathbb{N}}[0] : \mathcal{A} \to \mathcal{A}[0]$ is the identity morphism, hence it is mapped to the identity by Lemma 3.2.29.

For the successor case $k+1$, the inclusion morphism computes to the composition $\mathcal{A} \to \mathcal{A}[k'] \to \mathcal{A}[k][1]$. By Lemma 3.2.30, passing to the colimit preserves composition, so there is a homotopy

$$\text{incl-hom}_{\mathbb{N}}[k+1]_\infty \sim \text{incl-hom}_{\mathbb{N}}[1]_\infty \circ \text{incl-hom}_{\mathbb{N}}[k]_\infty.$$

The map $\text{incl-hom}_{\mathbb{N}}[k]_\infty$ is homotopic to the identity map by the inductive hypothesis, so it remains to show that $\text{incl-hom}_{\mathbb{N}}[1] : \mathcal{A}[k] \to \mathcal{A}[k+1]$ induces the identity map. The map $\text{incl-hom}_{\mathbb{N}}[1]_\infty : X \to X$ is constructed using the universal property of $c[k]$ being the colimit of $\mathcal{A}[k]$, by the cocone $\text{precomp-hom}_{\mathbb{N}}{}_{c[k+1]}(\text{incl-hom}_{\mathbb{N}}[1])$. By Corollary 3.2.46 this cocone is homotopic to the cocone $c[k]$. Hence it suffices to show that the map induced by the cocone $c[k]$ using the universal property of $c[k]$ is the identity map, which is Lemma 3.2.9. □

3.2.3 Descent property and flattening lemma

We prove the flattening lemma phrased with descent data, which can be seen as an elementary case of the main theorem from [17]. The full theorem could reasonably be called "generalized flattening lemma", as it shows commutativity of taking the total space and sequential colimit of not just a type family induced by descent data, but a more general case where one may take an arbitrary dependent sequential diagram, and generate the descent data by taking colimits of increasingly more shifted total sequential diagrams.

Definition 3.2.48. Given a sequential diagram $\mathcal{A} \doteq (A, a)$, define the type of **descent data** over $\mathcal{A}$ to be the type of pairs (B, b), where B is a family of type families

$$B : (n : \mathbb{N}) \to A_n \to \mathcal{U}$$

and b is a family of fiberwise equivalences

$$b : (n : \mathbb{N})(x : A_n) \to B_n(x) \simeq B_{n+1}(a_n x).$$

We write $\text{DD}_{\mathbb{N}}(\mathcal{A})$ for the type of descent data over $\mathcal{A}$.

Remark 3.2.49. There is a principled way of looking at descent data over sequential diagrams — as *equifibered* sequential diagrams. A fibered (or dependent) sequential diagram consists of type families B_n over A_n's and connecting maps b_n over a_n's. An equifibered sequential diagram is one in which all connecting maps are equivalences.

In fact, all descent data arise as dependent diagrams with maps replaced by equivalences. Sometimes the structure may be simplified by replacing a span of fiberwise equivalences with a single fiberwise equivalence. To take the example of pushout, an equifibered span diagram consists of the data[1]

$$\begin{aligned}
&P_S : S \to \mathcal{U}\\
&P_A : A \to \mathcal{U}\\
&P_B : B \to \mathcal{U}\\
&P_f : (s : S) \to P_S(s) \simeq P_A(fs)\\
&P_g : (s : S) \to P_S(s) \simeq P_B(gs).
\end{aligned}$$

The type of equifibered span diagrams is equivalent to the type of descent data over span diagrams, since we can contract away the pair (P_S, P_f). The simplification reduces the amount of data we have to track and make coherent, at the expense of introducing an arbitrary direction — there is no reason to prefer the direction $P_A(fs) \simeq P_B(gs)$ over $P_B(gs) \simeq P_A(fs)$.

Sequential diagrams do not contain any spans, so there is no simplification to be made.

Construction 3.2.50. Given a sequential diagram $\mathcal{A} \doteq (A, a)$ and a cocone $c \doteq (i, H) : \mathsf{cocone}_{\mathbb{N}}(\mathcal{A}, X)$, construct the map

$$\mathsf{dd}_{\mathbb{N}}\text{-}\mathsf{fam}_c : (X \to \mathcal{U}) \to \mathsf{DD}_{\mathbb{N}}(\mathcal{A})$$

which sends a type family B to the descent data

$$\begin{aligned}
&(\lambda n, x \to B(i_n x)) \quad &&: (n : \mathbb{N}) \to A_n \to \mathcal{U}\\
&(\lambda n, x \to \mathsf{tr}_B(H_n x)) &&: (n : \mathbb{N})(x : A_n) \to B(i_n x) \simeq B(i_{n+1}(a_n x)).
\end{aligned}$$

Definition 3.2.51. Given a sequential diagram $\mathcal{A} \doteq (A, a)$ and a cocone $c \doteq (i, H) : \mathsf{cocone}_{\mathbb{N}}(\mathcal{A}, X)$, a **type family with descent data** is a triple (B_∞, B', e'), where $B_\infty : X \to \mathcal{U}$ is a type family, $B' \doteq (B, b) : \mathsf{DD}(\mathcal{A})$ is descent data over $\mathcal{A}$, and $e' \doteq (e, K)$ is an equivalence of descent data consisting of a family of equivalences

$$e : (n : \mathbb{N})(x : A_n) \to B_\infty(i_n x) \simeq B_n(x)$$

and a family of commuting squares indexed by n and $x : A_n$

$$\begin{array}{ccc}
B_\infty(i_n x) & \xrightarrow{e_n(x)} & B_n(x)\\
{\scriptstyle \mathsf{tr}_{B_\infty}(H_n x)}\downarrow & & \downarrow{\scriptstyle b_n(x)}\\
B_\infty(i_{n+1}(a_n x)) & \xrightarrow[e_{n+1}(a_n x)]{} & B_{n+1}(a_n x).
\end{array}$$

[1]There is a slight inaccuracy caused by universe levels — the type families P_A, P_B and P_S should be allowed to range over different universes $\mathcal{U}$, $\mathcal{V}$, $\mathcal{W}$. This can be formally rectified by appropriately raising to the common universe $\mathcal{U} \sqcup \mathcal{V} \sqcup \mathcal{W}$.

We write $e' : B_\infty \approx B'$ for the triple (B_∞, B', e').

Theorem 3.2.52 (Descent property of sequential colimits)**.** *Consider a sequential diagram $\mathcal{A} \doteq (A, a)$ and its sequential colimit c : cocone$\mathbb{N}(\mathcal{A}, X)$. Then the map* dd$\mathbb{N}$-fam$_c$ *is an equivalence.*

Proof. We construct a commuting triangle of maps

$$\begin{array}{ccc} (X \to Y) & \xrightarrow[\simeq]{\text{cocone}\mathbb{N}\text{-map}_c} & \text{cocone}\mathbb{N}(\mathcal{A}, \mathcal{U}) \\ & {\scriptstyle \text{dd}\mathbb{N}\text{-fam}_c} \searrow \quad \swarrow {\scriptstyle \simeq} & \\ & \text{DD}\mathbb{N}(\mathcal{A}). & \end{array}$$

The right equivalence sends (B, H) to $(B, \lambda n, x \to \text{equiv-eq}(H_n(x)))$, and the triangle commutes by function extensionality and Lemma 1.0.11. □

Corollary 3.2.53. *Given a sequential diagram $\mathcal{A} \doteq (A, a)$, its colimit $c \doteq (i, H)$ on X, and descent data $(B, b) : \text{DD}\mathbb{N}(\mathcal{A})$, there is a unique type family $B_\infty : X \to \mathcal{U}$ and an equivalence of descent data $e : B_\infty \approx (B, b)$.*

Construction 3.2.54. Given a sequential diagram $\mathcal{A} \doteq (A, a)$ and descent data $(B, b) : \text{DD}\mathbb{N}(\mathcal{A})$, take the **total sequential diagram** to be the diagram

$$\begin{aligned} &(\lambda n \to \Sigma A_n B_n) \quad : \mathbb{N} \to \mathcal{U} \\ &(\lambda n \to \text{tot}_{a_n}(b_n)) : (n : \mathbb{N}) \to (\Sigma A_n B_n) \to (\Sigma A_{n+1} B_{n+1}). \end{aligned}$$

Construction 3.2.55. Given a sequential diagram $\mathcal{A} \doteq (A, a)$, a cocone $c \doteq (i, H)$ on X, and a family with descent data $(e, K) : (B, b) \approx B_\infty$, construct the **total cocone** under the total sequential diagram

$$\begin{array}{ccc} \Sigma A_n B_n & \xrightarrow{\text{tot}_{a_n}(b_n)} & \Sigma A_{n+1} B_{n+1} \\ & {\scriptstyle \text{tot}_{i_n}(e_n)} \searrow \quad \swarrow {\scriptstyle \text{tot}_{i_{n+1}}(e_{n+1})} & \\ & \Sigma X B_\infty & \end{array}$$

which commutes by the homotopy H' given at $x : A_n$, $y : B_n(a)$ by

$$\begin{aligned} H'_1 &:= H_n(x) \quad : i_n(x) = i_{n+1}(a_n x) \\ H'_2 &:= K_n(x, y) : \text{tr}_{B_\infty}(H_n x)(e_n(y)) = e_{n+1}(b_n(x, y)). \end{aligned}$$

We proceed similarly to the proof of the flattening lemma with descent data for pushouts — we split the proof into two steps, one showing it holds for a type family $P : X \to \mathcal{U}$ and its induced descent data, and one generalizing it to arbitrary families with descent data.

The proofs lean on technical results regarding preservation of universal properties by equivalences of cocones and coforks (not to be confused with homotopies of cocones and coforks), which were introduced in the formalization but we do not cover them in the thesis text. The precise statements and proofs of those lemmas can be read off the attached Agda code.

Remark 3.2.56. The idea of those lemmas is that in the context of cocones under span diagrams, a commuting cube whose vertical maps are equivalences may be regarded as an equivalence of span diagrams and cocones under them. With this perspective, Lemma 2.3.1 says being a pushout is preserved by equivalences of cocones. Adapting it to coforks yields the concept of an equivalence of double arrows and coforks under them, and the property of being a coequalizer is preserved by such an equivalence, because equivalences of coforks induce equivalences of the associated cocones. Going one step further, we get a notion of equivalences of sequential diagrams and cocones under those, and being a sequential colimit is preserved by equivalences of cocones, because they induce equivalences of the associated coforks.

Lemma 3.2.57. *Consider a sequential diagram $\mathcal{A}$, its colimit $c : \mathrm{cocone}(\mathcal{A}, X)$ and a type family $B_\infty : X \to \mathcal{U}$. Then the total cocone of the family with descent data $\mathrm{id} : \mathrm{dd}\mathbb{N}\text{-}\mathrm{fam}_c(B_\infty) \approx B_\infty$ is a sequential colimit.*

Proof. Similarly to the proof of the flattening lemma for coequalizers, we leverage the fact that sequential colimits correspond to certain coequalizers.

Write $c \doteq (i, H)$. We construct an equivalence of coforks

$$\begin{array}{ccccc}
\Sigma(n:\mathbb{N})(x:A_n).\, B_\infty(i_n x) & \overset{\mathrm{tot}_{+1}(\mathrm{tot}_{a_{(-)}}(\mathrm{tr}_{B_\infty}(H_{(-)})))}{\underset{\mathrm{id}}{\rightrightarrows}} & \Sigma(n:\mathbb{N})(x:A_n).\, B_\infty(i_n x) & \xrightarrow{\text{ind-}\Sigma(\mathrm{tot}_{i_{(-)}}(\mathrm{id}))} & \Sigma X B_\infty \\
\Big\downarrow{\scriptstyle \text{assoc-}\Sigma}\,\simeq & & \simeq\,\Big\downarrow{\scriptstyle \text{assoc-}\Sigma} & & \simeq\,\Big\downarrow{\scriptstyle \mathrm{id}} \\
\Sigma((n,x):\Sigma\mathbb{N}A).\, B_\infty(i_n x) & \underset{\mathrm{tot}_{\mathrm{id}}(\mathrm{id})}{\overset{\mathrm{tot}_{\mathrm{tot}_{+1}(a_{(-)})}(\mathrm{tr}_{B_\infty}(H_{(-)}))}{\rightrightarrows}} & \Sigma((n,x):\Sigma\mathbb{N}A).\, B_\infty(i_n x) & \xrightarrow[\mathrm{tot}_{\text{ind-}\Sigma(i)}(\mathrm{id})]{} & \Sigma X B_\infty
\end{array}$$

where the top cofork is the cofork associated to the total cocone, and on the bottom is the total cofork of the cofork associated to c and B_∞. All the squares commute by refl-htpy, so the coherence is a combination of unit laws for concatenation with refl-htpy and whiskering by id.

Since the bottom cofork is a coequalizer by the flattening lemma for coequalizers, the top cofork is also a coequalizer, from which it follows that the total cocone is a sequential colimit. □

Theorem 3.2.58 (Flattening lemma for sequential colimits)**.** *Given a sequential colimit $c : \mathrm{cocone}\mathbb{N}(\mathcal{A}, X)$ and a family with descent data $(B, b) \approx B_\infty$, the total cocone is a sequential colimit.*

Proof. Put $\mathcal{A} \doteq (A, a)$ and $c \doteq (i, H)$. It suffices to show that there is an "equivalence of cocones"

$$\begin{array}{ccccc}
\Sigma A_n B_n & & \xrightarrow{\mathrm{tot}_{a_n}(b_n)} & & \Sigma A_{n+1} B_{n+1} \\
\Big\downarrow{\scriptstyle \mathrm{tot}(e_n)} & \searrow^{\mathrm{tot}_{i_n}(e_n)} & & \swarrow_{\mathrm{tot}_{i_{n+1}}(e_{n+1})} & \Big\downarrow{\scriptstyle \mathrm{tot}(e_{n+1})} \\
\Sigma A_n(B_\infty \circ i_n) & \xrightarrow{\mathrm{tr}_{B_\infty}(H_n)} & \Big\downarrow{\scriptstyle \mathrm{id}} & \xrightarrow{\quad} & \Sigma A_{n+1}(B_\infty \circ i_{n+1}) \\
& \searrow_{\mathrm{tot}_{i_n}(\mathrm{id})} & \Sigma X B_\infty, & \swarrow^{\mathrm{tot}_{i_{n+1}}(\mathrm{id})} &
\end{array}$$

where the bottom cocone is the total cocone of B_∞ and its induced descent data, and the top cocone is the total cocone of $(B, b) \approx B_\infty$. The vertical maps are equivalences, and the prisms commute by a homotopy algebra argument similar to Lemma 2.3.6. The bottom cocone is a sequential colimit by Lemma 3.2.57, so it follows that the top cocone is a sequential colimit. □

Chapter 4

Partial proof of correctness of the zigzag construction

Wärn [19] describes an explicit construction of identity types of pushouts. He does so by fixing an element $a_0 : A$, and then defining type families $a_0 \rightsquigarrow_\infty a$ and $a_0 \rightsquigarrow_\infty b$, such that for any $a : A$ and $b : B$, there are equivalences

$$(\mathrm{inl}(a_0) = \mathrm{inl}(a)) \simeq (a_0 \rightsquigarrow_\infty a)$$
$$(\mathrm{inl}(a_0) = \mathrm{inr}(b)) \simeq (a_0 \rightsquigarrow_\infty b).$$

The type families are defined by gradual approximations of the identity types, $a_0 \rightsquigarrow_t a$ and $a_0 \rightsquigarrow_{t+1} b$. If one thinks of the standard pushout $A \sqcup_S B$ as a coproduct $A + B$ with added "bridges" from $f(s)$ to $g(s)$, then $a_0 \rightsquigarrow_t a$ describes the type of identifications between $\mathrm{inl}(a_0)$ and $\mathrm{inl}(a)$, provided that we can pass from the A component to the B component and back t times, and similarly for $a_0 \rightsquigarrow_{t+1} b$. The full identity types are then constructed by removing the upper bound on the number of steps, by taking the sequential colimit.

The two type families are related — if one can get from $\mathrm{inl}(a_0)$ to $\mathrm{inl}(fs)$ in t crossings, then one can get from $\mathrm{inl}(a_0)$ to $\mathrm{inr}(gs)$ in $t + 1$ crossings, and similarly in reverse. We can formally encode this relationship in a structure called a "zigzag" between sequential diagrams. We begin by defining general zigzags of sequential diagrams and their behavior in the colimit. Then we define the type families of approximations of identity types, and a zigzag between them. At last, we present a partial proof that the construction satisfies the induction principle of identity systems of pushouts from section 2.4. One coherence proof remains unsolved. The construction and proof of correctness are presented with emphasis on their encoding in the Agda proof assistant [2], which due to its mutually inductive nature presented challenges to termination checking and computation.

4.1 Zigzags between sequential diagrams

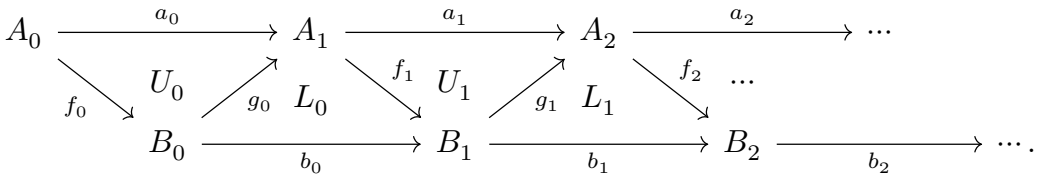
Definition 4.1.1. Given sequential diagrams $\mathcal{A} \doteq (A, a)$ and $\mathcal{B} \doteq (B, b)$, a **zigzag** between them is a quadruple (f, g, U, L), where f and g are families of maps

$$f : (n : \mathbb{N}) \to A_n \to B_n$$
$$g : (n : \mathbb{N}) \to B_n \to A_{n+1},$$

and U and L are families of coherences between them

$$U : (n : \mathbb{N}) \to a_n \sim (g_n \circ f_n)$$
$$L : (n : \mathbb{N}) \to b_n \sim (f_{n+1} \circ g_n).$$

A zigzag (f, g, U, L) can be visualized as a sequence of juxtaposed triangles



By forgetting the first triangle and turning the figure upside down, we get a new zigzag, this time between $\mathcal{B}$ and the shift $\mathcal{A}[1]$. This new zigzag is called a half-shift.

Construction 4.1.2. Given sequential diagrams $\mathcal{A} \doteq (A, a)$ and $\mathcal{B} \doteq (B, b)$, and a zigzag $z \doteq (f, g, U, L)$ between them, construct the **half-shift** of z as the zigzag (g, f', L, U') between $\mathcal{B}$ and $\mathcal{A}[1]$, where

$$\begin{aligned}
g &: (n : \mathbb{N}) \to B_n \to A_{n+1} \\
f' \coloneqq (\lambda n \to f_{n+1}) &: (n : \mathbb{N}) \to A_{n+1} \to B_{n+1} \\
L &: (n : \mathbb{N}) \to b_n \sim (f'_n \circ g_n) \\
U' \coloneqq (\lambda n \to U_{n+1}) &: (n : \mathbb{N}) \to a_{n+1} \sim (g_{n+1} \circ f'_n).
\end{aligned}$$

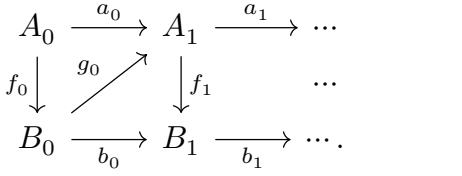
Remark 4.1.3. Half-shifts of zigzags provide a symmetry of the downward-going f maps and upward-going g maps. We exploit this symmetry in constructions and lemmas to follow, by formulating them for the downwards direction, and then applying them to the half-shift of a zigzag to get the constructions for the upward direction.

Repeating a half-shift twice gives a full shift, which shifts all the components by one.

Construction 4.1.4. Given a zigzag $z \doteq (f, g, U, L)$ between the sequential diagrams $\mathcal{A}$ and $\mathcal{B}$, define the **full shift** of z, denoted $z[1]$, as the zigzag between $\mathcal{A}[1]$ and $\mathcal{B}[1]$ obtained by taking the half-shift of the half-shift of z. Explicitly, it consists of the components (f', g', U', L'), where

$$\begin{aligned}
f' \coloneqq (\lambda n \to f_{n+1}) &: (n : \mathbb{N}) \to A_{n+1} \to B_{n+1} \\
g' \coloneqq (\lambda n \to g_{n+1}) &: (n : \mathbb{N}) \to B_{n+1} \to A_{n+2} \\
U' \coloneqq (\lambda n \to U_{n+1}) &: (n : \mathbb{N}) \to a_{n+1} \sim (g_{n+1} \circ f_{n+1}) \\
L' \coloneqq (\lambda n \to L_{n+1}) &: (n : \mathbb{N}) \to b_{n+1} \sim (f_{n+2} \circ g_{n+1}).
\end{aligned}$$

We can "shear" a zigzag to look at it from yet another perspective, as a morphism $F : \mathcal{A} \to \mathcal{B}$, where the necessary squares are constructed by pasting triangles. Diagrammatically, we have



This is the morphism of sequential diagrams associated to the zigzag.

Construction 4.1.5. Given a zigzag $z \doteq (f, g, U, L)$ between $\mathcal{A}$ and $\mathcal{B}$, construct the **associated morphism** of sequential diagrams from $\mathcal{A}$ to $\mathcal{B}$ to be the morphism (f, H), where

$$\begin{aligned} f &: (n : \mathbb{N}) \to A_n \to B_n \\ H &:= (L_n \cdot_r f_n) \bullet_h (f_{n+1} \cdot_l U_n^{-1}) : (n : \mathbb{N}) \to (b_n \circ f_n) \sim (f_{n_1} \circ a_n). \end{aligned}$$

For sequential colimits $c : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}, X)$ and $c' : \mathrm{cocone}_{\mathbb{N}}(\mathcal{B}, Y)$, write $f_\infty : X \to Y$ for the induced map of colimits.

By taking the associated morphism of a half-shift of a zigzag, we get the associated inverse morphism.

Construction 4.1.6. Given a zigzag z between $\mathcal{A}$ and $\mathcal{B}$, define the **associated inverse morphism** to be the morphism $\mathcal{B} \to \mathcal{A}[1]$ associated to the half-shift of z.

For sequential colimits $c : \mathrm{cocone}_{\mathbb{N}}(\mathcal{B}, Y)$ and $c' : \mathrm{cocone}_{\mathbb{N}}(\mathcal{A}[1], X)$, write $g_\infty : Y \to X$ for the induced map of colimits.

It deserves the moniker "inverse", because we will show that the induced map g_∞ is an inverse of f_∞. The last prerequisite to showing that the induced maps are inverses is a lemma relating zigzags and the shift inclusion morphisms $\text{incl-hom}_{\mathbb{N}}[1]$.

Lemma 4.1.7. *Given a zigzag between sequential diagrams $\mathcal{A}$ and $\mathcal{B}$, the inclusion morphism* $\text{incl-hom}_{\mathbb{N}}[1] : \mathcal{A} \to \mathcal{A}[1]$ *is homotopic to the composition of the associated morphism $\mathcal{A} \to \mathcal{B}$ and the inverse morphism $\mathcal{B} \to \mathcal{A}[1]$.*

Proof. Write (f, g, U, L) for the zigzag, and $\mathcal{A} \doteq (A, a)$ and $\mathcal{B} \doteq (B, b)$.

We need to show that the morphism

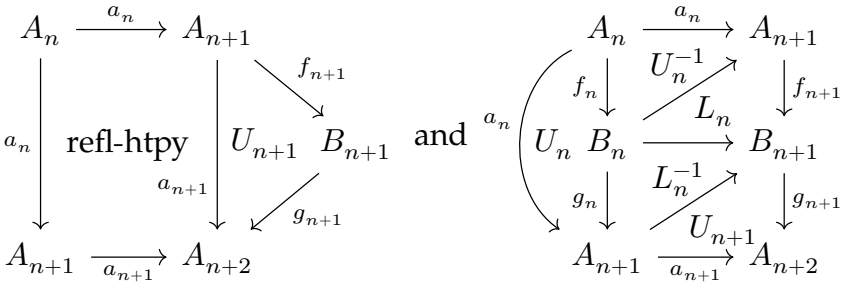
$$\begin{array}{ccccccc} A_0 & \xrightarrow{a_0} & A_1 & \xrightarrow{a_1} & \cdots \\ {\scriptstyle a_0}\downarrow & & \downarrow{\scriptstyle a_1} & & \cdots \\ A_1 & \xrightarrow[a_1]{} & A_2 & \xrightarrow[a_2]{} & \cdots. \end{array}$$

and

$$\begin{array}{ccccccc} A_0 & \xrightarrow{a_0} & A_1 & \xrightarrow{a_1} & \cdots \\ {\scriptstyle f_0}\downarrow & \nearrow_{g_0} & \downarrow{\scriptstyle f_1} & & \cdots \\ B_0 & \xrightarrow[b_0]{} & B_1 & \xrightarrow[b_1]{} & \cdots \\ {\scriptstyle g_0}\downarrow & \nearrow_{f_1} & \downarrow{\scriptstyle g_1} & & \cdots \\ A_1 & \xrightarrow[a_1]{} & A_1 & \xrightarrow[a_2]{} & \cdots \end{array}$$

are homotopic.

The first component of the homotopy is a family of homotopies of maps $K_n : a_n \sim g_n \circ f_n$. We take the triangles U_n for the homotopies of maps. Then we need to show that the homotopies



are themselves homotopic. With some effort the pairs U_n and U_n^{-1}, and L_n and L_n^{-1} cancel out, and we end up with U_{n+1} on both sides. □

Theorem 4.1.8. *Consider sequential diagrams $\mathcal{A}$ and $\mathcal{B}$, their respective colimits $c : \text{cocone}_{\mathbb{N}}(\mathcal{A}, X)$ and $c' : \text{cocone}_{\mathbb{N}}(\mathcal{B}, Y)$, and a zigzag $z \doteq (f, g, U, L)$ between them. Then the associated morphism to z induces a map $f_\infty : X \to Y$, and when we take $c[1] : \text{cocone}_{\mathbb{N}}(\mathcal{A}[1], X)$ to be the colimit of $\mathcal{A}[1]$, the associated inverse morphism induces a map $g_\infty : Y \to X$. Then the two maps are mutually inverse equivalences.*

Proof. We first show that with the above assumptions, f_∞ is a section of g_∞, i.e. $g_\infty \circ f_\infty \sim \text{id}$. We can prove it by concatenating the following homotopies:

$$\begin{aligned} g_\infty \circ f_\infty &\sim (g \circ f)_\infty && \text{by Lemma 3.2.30} \\ &\sim (\text{incl-hom}_{\mathbb{N}}[1])_\infty && \text{by Lemma 4.1.7 and Lemma 3.2.33} \\ &\sim \text{id} && \text{by Lemma 3.2.47} \end{aligned}$$

Then consider the half-shift of z. The premises of the theorem are fulfilled by the sequential diagrams $\mathcal{B}$ and $\mathcal{A}[1]$, the colimits c' and $c[1]$, and the half-shift. By the first half of the proof, we get that g_∞ is a section of $f[1]_\infty$, in other words there is a triangle $f[1]_\infty \circ g_\infty \sim \text{id}$. Thus we found a section and a retraction of g_∞, so by definition it is an equivalence. Then since f_∞ is a section of an equivalence, it is itself an equivalence, and it is an inverse of g_∞. □

4.2 The zigzag construction of identity types

The following construction is a variation of the original zigzag construction of Wärn [19]. It differs from Wärn's version in representation of span diagrams — Wärn represents span diagrams as a pair of types A, B with a type-valued relation $R : A \to B \to \mathcal{U}$. We use the same representation as in the rest of the thesis, i.e. a triple of types S, A, B equipped with a pair of maps $f : S \to A$, $g : S \to B$. These two representations are equivalent: a relation R can be seen as the spanning type $\Sigma(a : A)(b : B).\, R(a, b)$ with the first and second projections, and conversely a spanning type S with maps f, g can be seen as the relation $\lambda a, b \to \Sigma(s : S).\, (fs = a) \times (gs = b)$. Adapting Wärn's construction involves reconstructing a relation from a span diagram and removing contractible pairs.

This version of the construction is the original type-theoretic one. Wärn later published a categorical version [20]. Formalization of the categorical version is not attempted in this thesis. It requires different infrastructure from the one that had already been built by the time of publication of the categorical version, and it is also not as straightforward to recover the concrete equivalences between identity types and the resulting type families.

In the rest of the thesis we assume existence of standard pushouts and sequential colimits. The pushout cocone of a span diagram is written pushout($\mathcal{S}$). Its left and right inclusions are denoted inl and inr, respectively, and its homotopy is denoted glue. The unique dependent map corresponding to a dependent cocone d is called dep-cogap(d). The sequential colimit cocone of a sequential diagram $A_\bullet$ is denoted A_∞, and we abuse notation to use the same symbol for underlying type of the colimit. The inclusion maps into the sequential colimits are denoted ι_n, and its homotopies are denoted κ_n.

We begin by describing the construction informally, and then discuss necessary modifications to encode it in a proof assistant.

For the remainder of this section, assume a span diagram $\mathcal{S} \doteq (f, g)$ and a basepoint $a_0 : A$. The data we need to construct is a pair of type families

$$P_A^n : A \to \mathcal{U}$$
$$P_B^n : B \to \mathcal{U}$$

and a pair of connecting maps

$$- \bullet_n s : P_A^n(fs) \to P_B^{n+1}(gs)$$
$$- \bullet_n \overline{s} : P_B^n(gs) \to P_A^n(fs),$$

all of which are indexed by $n : \mathbb{N}$. The construction proceeds by induction on n, with various interdependencies between definitions of the above data.

Take $P_A^0(a)$ to be the identity type $(a_0 = a)$, $P_B^0(b)$ to be the empty type $\mathbb{0}$, and $- \bullet_0 \overline{s}$ to be the unique map out of the empty type[1]. Note that we cannot yet define $- \bullet_0 s$, because its intended codomain $P_B^1(gs)$ is not defined. Then construct the types $P_A^{n+1}(a)$, $P_B^{n+1}(b)$ and the connecting maps $- \bullet_{n+1} \overline{s}$ and $- \bullet_n s$ together as the pushouts and their inclusion maps

$$\begin{array}{ccc} \Sigma(s : S)(r : (a = fs)).\ P_A^n(a) & \xrightarrow{\mathrm{tot}(\mathrm{tot}(-\bullet'_n s))} & \Sigma(s : S)(r : (a = fs)).\ P_B^{n+1}(gs) \\ {\scriptstyle \mathrm{pr}_3}\downarrow & \ulcorner & \downarrow{\scriptstyle -\bullet_{n+1}\overline{s} :\equiv \mathrm{inr}} \\ P_A^n(a) & \longrightarrow & P_A^{n+1}(a) \end{array}$$

for $a : A$, and

$$\begin{array}{ccc} \Sigma(s : S)(r : (b = gs)).\ P_B^n(b) & \xrightarrow{\mathrm{tot}(\mathrm{tot}(-\bullet_n \overline{s}))} & \Sigma(s : S)(r : (b = gs)).\ P_A^n(fs) \\ {\scriptstyle \mathrm{pr}_3}\downarrow & \ulcorner & \downarrow{\scriptstyle -\bullet_n s :\equiv \mathrm{inr}} \\ P_B^n(b) & \longrightarrow & P_B^{n+1}(b) \end{array}$$

for $b : B$. The top maps $\mathrm{tot}(\mathrm{tot}(-\bullet_n s))$ and $\mathrm{tot}(\mathrm{tot}(-\bullet_n \overline{s}))$ additionally transport in the type families P_A^n and P_B^n using the available identifications $r : (a = fs)$ and $r : (b = gs)$ respectively, to make the types line up.

The data dependencies resulting from these definitions are summarized in Figure 4.1.

[1]Since the types A, B and S can all live in different universes, we also formally have to raise the identity type and the empty type to the smallest universe containing all of them. This is done in the formalization, but it is omitted from the thesis text.

$$\cdot \vdash P_B^0 \qquad P_B^n, P_A^n, - \bullet_n \overline{s} \vdash P_B^{n+1}$$
$$\cdot \vdash P_A^0 \qquad P_A^n, P_B^{n+1}, - \bullet_n s \vdash P_A^{n+1}$$
$$\cdot \vdash - \bullet_0 \overline{s} \qquad P_A^{n+1} \vdash - \bullet_{n+1} \overline{s}$$
$$P_B^1(b) \vdash - \bullet_0 s \qquad P_B^{n+1} \vdash - \bullet_n s$$

Figure 4.1: Dependencies between definitions in the zigzag construction.

Formally, we induct on the stage $n : \mathbb{N}$, so we need a type family over $\mathbb{N}$ to induct into. Writing down the type of $- \bullet_n s$ poses a challenge already, because to define it at stage n, it needs to know what the type family P_B is at the next stage $n + 1$. One possibility is to not refer to its codomain as $P_B^{n+1}(gs)$, but instead inline its definition as the appropriate pushout, because all the necessary data to construct $P_B^{n+1}(gs)$ is available at stage n. Then after performing the construction, its codomain will be judgmentally equal to $P_B^{n+1}(gs)$. This approach has two disadvantages. The first one is duplication of code — one would need to construct the exact span diagram and its pushout to both state the type, and provide an inhabitant of one of its components. Additionally, the code for the definition of the cases $- \bullet_0 s$ and $- \bullet_{n+1} s$ would be identical. The other issue is computational: in contrast to $- \bullet_n \overline{s}$, the map $- \bullet_n s$ is defined as the right inclusion map of a pushout at every stage n. But this is invisible to computation, because it is defined together with the other data by induction on n, so the definition only computes when it is applied to either of the constructors 0 or $n' + 1$.

The preferred definition of the type family, which we chose to formalize, removes the $- \bullet_n s$ component altogether. In the construction itself it is only used to define P_A^{n+1}, where it can be replaced by a direct reference to the right inclusion map of the pushout $P_B^{n+1}(gs)$, which is already defined by the time we need to define P_A^{n+1}. Then $- \bullet_n s$ can be defined after the construction as the right inclusion map at every stage, without induction, removing code duplication and giving it the right computational behavior. We also want to refer to the span diagrams defining P_B^{n+1} and P_A^{n+1} later in the code, hence we also remember those in the construction.

Definition 4.2.1. Given a natural number n, define the type of **zigzag construction data** at stage n to be the type of quadruples $(P_B^n, P_A^n, - \bullet_n \overline{s}, D)$, where

$$P_B^n : B \to \mathcal{U}$$
$$P_A^n : A \to \mathcal{U}$$

are type families,

$$- \bullet_n \overline{s} : P_B^n(gs) \to P_A^n(fs)$$

is a family of maps indexed by $s : S$, and D is an element of the unit type if $n = 0$, or of the type of pairs $(\mathcal{T}_B^n, \mathcal{T}_A^n)$ where $\mathcal{T}_B^n$ is a family of span diagrams indexed by B, and $\mathcal{T}_A^n$ is a family of span diagrams indexed by A if n is a successor.

This type can be inhabited for all $n : \mathbb{N}$, using the construction described above.

Construction 4.2.2. Construct an inhabitant of the type of zigzag construction data for every stage n by induction.

For the zero case, use

$$\begin{aligned} P_B^0 &:= (\lambda b \to \mathbb{0}) && : B \to \mathcal{U} \\ P_A^0 &:= (\lambda a \to (a_0 = a)) && : A \to \mathcal{U} \\ - \bullet_n \overline{s} &:= \text{ex-falso} && : P_B^0(gs) \to P_A^0(fs) \\ D^0 &:= \star && : \mathbb{1}. \end{aligned}$$

For the successor case $n+1$, first construct the families of span diagrams $\mathcal{T}_B^{n+1}$. For an element $b : B$, define $\mathcal{T}_B^{n+1}(b)$ to be the span diagram

$$P_B^n(b) \xleftarrow{\text{pr}_3} \Sigma(s : S)(r : b = gs).\, P_B^n(b) \xrightarrow{\varphi} \Sigma(s : S)(r : b = gs).\, P_A^n(fs),$$

where φ sends (s, r, p) to $(s, r, (\text{tr}_{P_B^n}(r, p)) \bullet_n \overline{s})$. Take $P_B^{n+1}(b)$ to be the standard pushout of this diagram, and to denote its homotopy glue_B^n. Analogously, for an element $a : A$, define $\mathcal{T}_A^{n+1}(a)$ to be the span diagram

$$P_A^n(a) \xleftarrow{\text{pr}_3} \Sigma(s : S)(r : a = fs).\, P_A^n(a) \xrightarrow{\psi} \Sigma(s : S)(r : a = fs).\, P_B^{n+1}(gs)$$

where the map ψ takes (s, r, p) to $(s, r, \text{inr}(s, \text{refl}, \text{tr}_{P_A^n}(r, p)))$, using the right inclusion inr into the pushout $P_B^{n+1}(gs)$. Then define $P_A^{n+1}(a)$ to be the standard pushout of $\mathcal{T}_A^{n+1}(a)$, and denote its homotopy glue_A^n. Finally, define $p \bullet_{n+1} \overline{s}$ to be $\text{inr}(s, \text{refl}, p)$ using the right inclusion map into $P_A^{n+1}(fs)$.

We keep using the names P_B^n, P_A^n, $- \bullet_n s$, $\mathcal{T}_B^n$ and $\mathcal{T}_A^n$ for the corresponding elements of this canonical construction. Note that the span diagrams $\mathcal{T}_B^n(b)$ and $\mathcal{T}_A^n(a)$ are not defined when n is zero; they are the defining span diagrams of $P_B^n(b)$ and $P_A^n(a)$, respectively, which are only pushouts in the successor case.

Construction 4.2.3. For every stage $n : \mathbb{N}$ and element $s : S$, define the map

$$- \bullet_n s : P_A^n(fs) \to P_B^{n+1}(gs)$$

to send p to $\text{inr}(s, \text{refl}, p)$, where inr is the right inclusion of the defining pushout of $P_B^{n+1}(gs)$.

We may also construct the sequential diagrams of approximations of the type families $(\text{inl}(a_0) = \text{inr}(b))$ and $(\text{inl}(a_0) = \text{inl}(a))$.

Construction 4.2.4. Given an element $b : B$, define the sequential diagram $P_B^\bullet(b)$ to be the diagram

$$\mathbb{0} \xrightarrow{\text{incl}_B^0} P_B^1(b) \xrightarrow{\text{incl}_B^1} P_B^2(b) \xrightarrow{\text{incl}_B^2} \cdots,$$

where the maps incl_B^n are the left inclusions inl of the pushouts defining $P_B^{n+1}(b)$.

Denote its sequential colimit $P_B^\infty(b)$, with inclusion maps ι_B^n and coherences κ_B^n.

Construction 4.2.5. Given an element $a : A$, define the sequential diagram $P_A^\bullet(a)$ to be the diagram

$$(a_0 = a) \xrightarrow{\mathrm{incl}_A^0} P_A^1(a) \xrightarrow{\mathrm{incl}_A^1} P_A^2(a) \xrightarrow{\mathrm{incl}_A^2} \cdots,$$

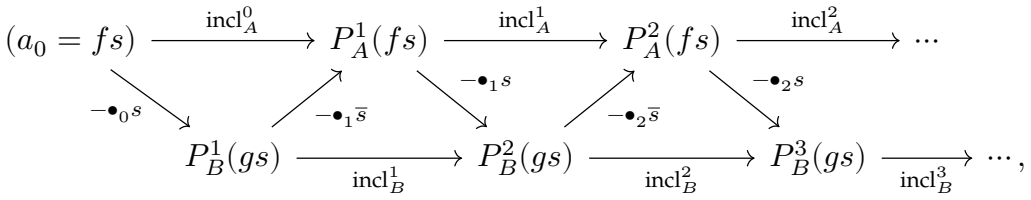
where the maps incl_A^n are the left inclusions inl of the pushouts defining $P_A^{n+1}(a)$.

Denote its sequential colimit $P_A^\infty(a)$, with inclusion maps ι_A^n and coherences κ_A^n.

While we define the full sequential diagram starting with $P_B^0(b)$, we only do so for uniformity of the zigzag construction. This way exactly the type families with a non-zero index are pushouts. When working with the construction we drop the first vertex and compute only with $P_B^{n+1}(b)$, which are all pushouts. This is accomplished by passing to the shift $P_B^\bullet(b)[1]$, which has the same colimit $P_B^\infty(b)$.

When constrained to $P_B^\bullet(gs)$ and $P_A^\bullet(fs)$, the two sequential diagrams admit a zigzag between them.

Construction 4.2.6. Given an element $s : S$, construct the zigzag Z between $P_A^\bullet(fs)$ and the shift $P_B^\bullet(gs)[1]$ as



where the triangles are the gluing homotopies

$$\mathrm{glue}_A^n(s, \mathrm{refl}) : \mathrm{incl}_A^n \sim (- \bullet_n s) \bullet_{n+1} \overline{s}$$
$$\mathrm{glue}_B^n(s, \mathrm{refl}) : \mathrm{incl}_B^n \sim (- \bullet_n \overline{s}) \bullet_n s.$$

of the defining pushouts of $P_A^{n+1}(fs)$ and P_B^{n+1}, respectively.

Construction 4.2.7. Define the **zigzag descent data** $(P_A^\infty, P_B^\infty, - \bullet_\infty s)$, where the type families are constructed by taking sequential colimits of $P_A^\bullet(a)$ and $P_B^\bullet(b)[1]$, respectively, and the family of equivalences

$$- \bullet_\infty s : P_A^\infty(fs) \simeq P_B^\infty(gs)$$

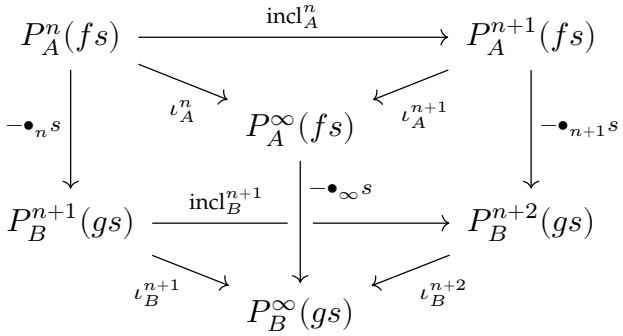
is induced by the zigzag between $P_A^\bullet(fs)$ and $P_B^\bullet(gs)[1]$ for an element $s : S$, using Theorem 4.1.8.

Additionally, this descent data is pointed with $\iota_A^0(\mathrm{refl}_{a_0}) : P_A^\infty(a_0)$, which we call refl_∞.

Lemma 4.2.8. *For every element $s : S$, there is a family of commuting square of maps indexed by $n : \mathbb{N}$*

$$\begin{array}{ccc} P_A^n(fs) & \xrightarrow{-\bullet_n s} & P_B^{n+1}(gs) \\ {\scriptstyle \iota_A^n}\downarrow & & \downarrow{\scriptstyle \iota_B^{n+1}} \\ P_A^\infty(fs) & \xrightarrow[-\bullet_\infty s]{} & P_B^\infty(gs), \end{array}$$

call them $C^n(s)$, which fit in a commuting prism



as the front squares, with the triangles being κ_A^n and κ_B^{n+1}, and the back square a composition of $(\mathrm{glue}_A^n)^{-1}$ and glue_B^{n+1}.

Proof. The data is obtained from the morphism associated to the zigzag Z using Lemma 3.2.28. □

4.3 Partial proof of correctness

We write down a partial proof that the zigzag descent data $(P_A^\infty, P_B^\infty, - \bullet_\infty s)$ is an identity system at refl_∞. Incompleteness of the proof is due to Conjecture 4.3.11, which remains an unproven coherence condition at the time of writing.

To show that the zigzag descent data is an identity system, we assume pointed descent data Q over its total span diagram, and the rest of the thesis is dedicated to providing a section of Q.

Explicitly, in the remainder of this section assume type families

$$\begin{aligned} Q_{\Sigma A} &: (a : A) \to P_A^\infty(a) \to \mathcal{U} \\ Q_{\Sigma B} &: (b : B) \to P_B^\infty(b) \to \mathcal{U}, \end{aligned}$$

a family of equivalences

$$Q_{\Sigma S} : (s : S)(p : P_A^\infty(a)) \to Q_{\Sigma A}(fs, p) \simeq Q_{\Sigma B}(gs, p \bullet_\infty s),$$

and a point $q_0 : Q_{\Sigma A}(a_0, \mathrm{refl}_\infty)$. The goal is to conjure a section, i.e. define a pair of dependent functions

$$\begin{aligned} t_A &: (a : A)(p : P_A^\infty(a)) \to Q_{\Sigma A}(a, p) \\ t_B &: (b : B)(p : P_B^\infty(b)) \to Q_{\Sigma B}(b, p) \end{aligned}$$

and a family of identifications

$$t_S : (s : S)(p : P_A^\infty(fs)) \to Q_{\Sigma S}(s, p, t_A(fs, p)) = t_B(gs, p \bullet_\infty s).$$

We may occasionally omit some arguments when writing down expressions, and we pass freely between the curried and uncurried forms, to aid readability and clarity of intent.

In each case the construction proceeds by induction on the argument p. By the dependent universal property of sequential colimits, it suffices to define the data

$$\begin{aligned}
t_A^n &: (a : A)(p : P_A^n(a)) \to Q_{\Sigma A}(a, \iota_A^n(p)) \\
K_A^n &: (a : A)(p : P_A^n(a)) \to \mathrm{tr}_{Q_{\Sigma A}(a)}(\kappa_A^n(p), t_A^n(a,p)) = t_A^{n+1}(a, \mathrm{incl}_A^n\, p) \\
t_B^n &: (b : B)(p : P_B^n(b)) \to Q_{\Sigma B}(b, \iota_B^n(p)) \\
K_B^n &: (b : B)(p : P_B^n(b)) \to \mathrm{tr}_{Q_{\Sigma B}(b)}(\kappa_B^n(p), t_B^n(b,p)) = t_B^{n+1}(b, \mathrm{incl}_B^n\, p)
\end{aligned}$$

for the maps, and

$$\begin{aligned}
t_S^n &: (s : S)(p : P_A^n(fs)) \to Q_{\Sigma S}(s, \iota_A^n(p), t_A(fs, \iota_A^n(p))) = t_B(gs, (\iota_A^n(p)) \bullet_\infty s) \\
K_S^n &: (s : S)(p : P_A^n(fs)) \to \\
&\quad \mathrm{tr}_{\lambda q \to (Q_{\Sigma S}(s,q,t_A(fs,q)) = t_B(gs, q \bullet_\infty s))}(\kappa_A^n(p), t_S^n(s,p)) = t_S^{n+1}(s, \mathrm{incl}_A^n\, p)
\end{aligned}$$

for the coherence.

We begin with the maps t_B^n and t_A^n, which are defined together. We first present their construction informally, and then explain the modifications necessary for Agda to accept them in code.

The maps t_B^n and t_A^n are defined by induction on n. In the zero case, the domain of $t_B^0(b)$ is the empty type, so it is defined by ex-falso. The domain of $t_A^0(a)$ is the identity type $(a_0 = a)$, and inducting on it gives us the goal of inhabiting $Q_{\Sigma A}(a_0, \iota_A^0(\mathrm{refl}))$, which we can do by the basepoint q_0. In the successor case $n+1$, we eliminate from the pushouts $P_B^{n+1}(b)$ and $P_A^{n+1}(a)$, so we use the dependent universal property of pushouts. This means that we need to give definitions of the maps on point constructors and a coherence over the path constructor, which defines a dependent cocone. Let us begin with the point constructor cases.

To define t_B^{n+1}, the behavior on point constructors consists of maps

$$\begin{aligned}
\overline{t_B^{n+1}}(b, \mathrm{incl}_B^n(-)) &: (p : P_B^n(b)) \to Q_{\Sigma B}(b, \iota_B^{n+1}(\mathrm{incl}_B^n\, p)) \\
\overline{t_B^{n+1}}(gs, - \bullet_n s) \quad &: (p : P_A^n(fs)) \to Q_{\Sigma B}(gs, \iota_B^{n+1}(p \bullet_n s)).
\end{aligned}$$

We need to distinguish between t_B^{n+1} and $\overline{t_B^{n+1}}$, because they have different computational properties. Since we work in a type theory without judgmental computation rules for pushouts, once we combine the data into the map t_B^{n+1}, its behavior on path constructors will only hold up to an identification. In contrast, $\overline{t_B^{n+1}}$ is a pair of functions which compute judgmentally, but their applications are only well-formed when they are syntactically applied to a point constructor. The statement of the second function is in a form after inducting on the term $r : b = gs$ in the context. The maps are defined by

$$\begin{aligned}
\overline{t_B^{n+1}}(b, \mathrm{incl}_B^n(p)) &:= \mathrm{tr}_{Q_{\Sigma B}(b)}(\kappa_B^n(p), t_B^n(b,p)) \\
\overline{t_B^{n+1}}(gs, p \bullet_n s) &:= \mathrm{tr}_{Q_{\Sigma B}(gs)}(C^n(s,p), Q_{\Sigma S}(s, \iota_A^n(p), fs, t_A^n(fs,p))).
\end{aligned}$$

Similarly for the definition of t_A^{n+1} on point constructors, we need to give

$$\overline{t_A^{n+1}}(a, \mathsf{incl}_A^n(-)) \ : (p : P_A^n(a)) \to Q_{\Sigma A}(a, \iota_A^{n+1}(\mathsf{incl}_A^n\, p))$$
$$\overline{t_A^{n+1}}(fs, - \bullet_{n+1} \overline{s}) : (p : P_B^{n+1}(gs)) \to Q_{\Sigma A}(fs, \iota_A^{n+1}(p \bullet_{n+1} \overline{s})).$$

The behavior on incl_A^n is analogous to the one of t_B^{n+1}, specifically

$$\overline{t_A^{n+1}}(a, \mathsf{incl}_A^n(p)) :\equiv \mathsf{tr}_{Q_{\Sigma A}(a)}(\kappa_A^n(p), t_A^n(a, p)).$$

For the second map, we would expect to use the inverse $Q_{\Sigma S}(s, -)^{-1}$. It is not that straightforward, because $t_B^{n+1}(gs, p)$ gives an element of $Q_{\Sigma B}(gs, \iota_B^{n+1}(p))$, and $Q_{\Sigma S}(s, -)^{-1}$ can only undo an application of $- \bullet_\infty s$ under $Q_{\Sigma B}(gs)$, not add an application of $- \bullet_\infty \overline{s}$ under $Q_{\Sigma A}(fs)$. One might try to fix this by recalling that $- \bullet_\infty s$ and $- \bullet_\infty \overline{s}$ are inverse equivalences, and using the homotopy

$$\mathsf{id} \sim (- \bullet_\infty \overline{s}) \bullet_\infty s.$$

This approach leads to complications in proving further coherences, so instead we define a helper function which we can later compute with.

Construction 4.3.1. For any element $s : S$ and natural number n, define the map

$$\Phi^n(s) : (p : P_B^{n+1}(gs)) \to Q_{\Sigma B}(gs, \iota_B^{n+1} p) \to Q_{\Sigma B}(gs, (\iota_A^{n+1}(p \bullet_{n+1} \overline{s})) \bullet_\infty s)$$

as the composition of transports

$$\begin{array}{c} Q_{\Sigma B}(gs, \iota_B^{n+1} p) \\ \Big\downarrow \mathsf{tr}_{Q_{\Sigma B(gs)}(\kappa_B^{n+1} p)} \\ Q_{\Sigma B}(gs, \iota_B^{n+2}(\mathsf{incl}_B^{n+1}\, p)) \\ \Big\downarrow \mathsf{tr}_{Q_{\Sigma B(gs, \iota_B^{n+2}(-))}(\mathsf{glue}_B^{n+1}(s, \mathsf{refl}, p))} \\ Q_{\Sigma B}(gs, \iota_B^{n+2}((p \bullet_{n+1} \overline{s}) \bullet_{n+1} s)) \\ \Big\downarrow \mathsf{tr}_{Q_{\Sigma B(gs)}(C^{n+1}(s, p \bullet_{n+1} \overline{s}))^{-1}} \\ Q_{\Sigma B}(gs, (\iota_A^{n+1}(p \bullet_{n+1} \overline{s})) \bullet_\infty s). \end{array}$$

With this function in hand, we can define

$$\overline{t_A^{n+1}}(fs, p \bullet_{n+1} \overline{s}) :\equiv (Q_{\Sigma S}(s, \iota_A^{n+1}(p \bullet_{n+1} \overline{s})))^{-1}(\Phi^n(s, p, t_B^{n+1}(gs, p))).$$

The path constructor cases will be less similar between t_B^{n+1} and t_A^{n+1}. The coherence of t_B^{n+1} asks us to inhabit the type

$$(p : P_B^n(gs)) \to \mathsf{tr}_{Q_{\Sigma B(gs, \iota_B^{n+1}(-))}}(\mathsf{glue}_B^n(p), \overline{t_B^{n+1}}(\mathsf{incl}_B^n(p))) = \overline{t_B^{n+1}}((p \bullet_n \overline{s}) \bullet_n s).$$

Here we once again need to case split on n. In the zero case we eliminate from $P_B^0(gs) \doteq \mathbb{0}$, so we inhabit the coherence by ex-falso. In the successor case, the right-hand side computes to

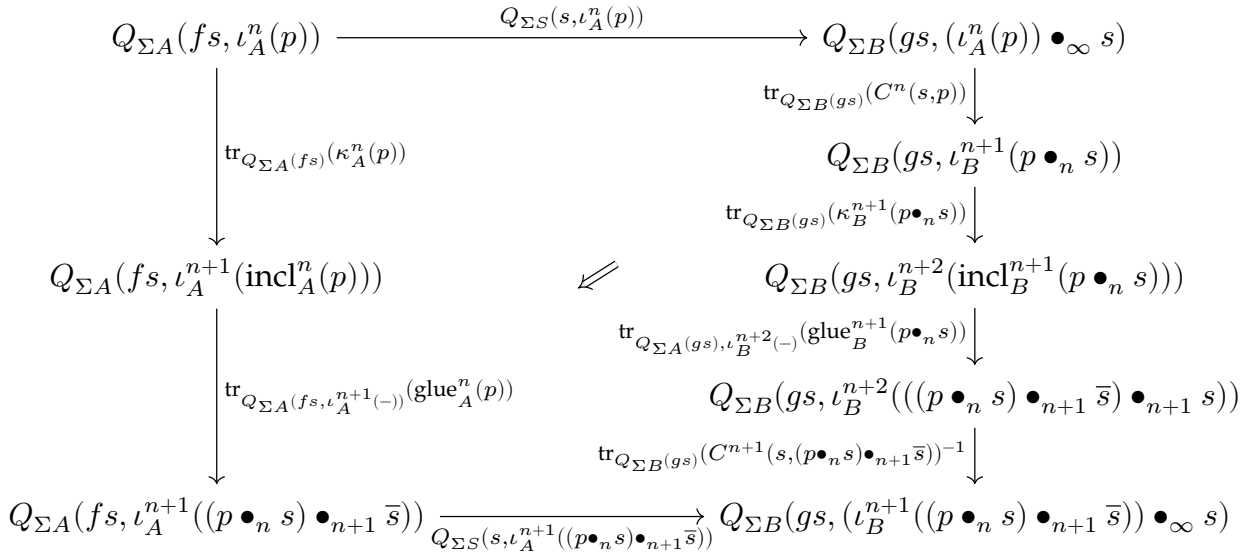
$$\mathsf{tr}_{Q_{\Sigma B}(gs)}(C^{n+1}(s, p \bullet_{n+1} \overline{s}), Q_{\Sigma S}(s, \iota_A^{n+1}(p \bullet_{n+1} \overline{s}), t_A^{n+1}(p \bullet_{n+1} \overline{s}))),$$

where we can use a computation rule of t_A^{n+1} and further expand Φ^n in the resulting expression. Then the two sides are identified, because the equivalence $Q_{\Sigma S}(s, \iota_A^{n+1}(p \bullet_{n+1} \overline{s}))$ cancels out with its inverse, and so does $\mathrm{tr}_{Q_{\Sigma B}(gs)}(C^{n+1}(s, p \bullet_{n+1} \overline{s}))$.

The coherence for t_A^{n+1} has the type

$$(p : P_A^n(fs)) \to \mathrm{tr}_{Q_{\Sigma A}(fs, \iota_A^{n+1}(-))}(\mathrm{glue}_A^n(p), \overline{t_A^{n+1}}(\mathrm{incl}_A^n(p))) = \overline{t_A^{n+1}}((p \bullet_n s) \bullet_{n+1} \overline{s}).$$

We can expand the $\overline{t_A^{n+1}}$, the Φ^n, and also use the computation rule of t_B^{n+1} to compute $t_B^{n+1}(p \bullet_n s)$. By transposing $Q_{\Sigma S}(s, \iota_A^{n+1}((p \bullet_n s) \bullet_{n+1} \overline{s}))^{-1}$ to the other side of the identification, we see that the goal is to make the diagram



commute from right to left — it is rotated only to fit on the page. Commutativity can be established using the following lemma.

Lemma 4.3.2. *Given types A, B and a function $f : A \to B$, type families $P : A \to \mathcal{U}$ and $Q : B \to \mathcal{V}$, a fiberwise function $h : (a : A) \to P(a) \to Q(fa)$, identifications $p : x = y$ of elements in A and $q : fx = fy$, and a coherence $\alpha : \mathrm{ap}_f(p) = q$, there is a commuting square*

$$\begin{array}{ccc} P(x) & \xrightarrow{\mathrm{tr}_P(p)} & P(y) \\ {\scriptstyle h(x)}\downarrow & & \downarrow{\scriptstyle h(y)} \\ Q(fx) & \xrightarrow[\mathrm{tr}_Q(q)]{} & Q(fy). \end{array}$$

Proof. By induction on α and p, it suffices to provide a homotopy $h(x) \sim h(x)$, which we can fulfill with refl-htpy. □

By computing transports of the form $\mathrm{tr}_{(P \circ f)}(p)$ to $\mathrm{tr}_P(\mathrm{ap}_f p)$ (Lemma 1.0.5) and applying distributivity of concatenation and transport (Lemma 1.0.2), we can collect both sides of the diagram into one transport each, at which point we may apply the above lemma. The coherence α is obtained from the prism

Lemma 4.2.8 by straightforward homotopy algebra. This completes the informal construction of the maps t_B^n and t_A^n.

In the formalization, we are again met with computation issues. Since t_B^n and t_A^n are defined together by induction on n, and the definition of t_B^{n+1} does one more case split on n, we end up with three cases for the induction, namely 0, 1 and $n+2$. As a consequence, $\overline{t_A^{n+1}}(p \bullet_{n+1} \overline{s})$ does not have a uniform definition, because it is also defined by cases 0, 1 and $n+2$. But we rely on its definition when computing the coherence of t_B^{n+1}. If we naïvely try to case split on n again to analyze the cases t_A^1 and t_A^{n+2} separately, we would end up defining everything in terms of the cases 0, 1, 2 and $n+3$. Instead, during the definition of t_B^{n+1} and t_A^{n+1} we carry a proof that $\overline{t_A^{n+1}}(p \bullet_{n+1} \overline{s})$ is identical to the desired composition of $Q_{\Sigma S}^{-1}$ and Φ. This component will be satisfied by refl at all stages.

Furthermore, during induction we need to compute with t_B^{n+1} and t_A^{n+1} as maps defined by the dependent universal property, meaning that we need to carry around their defining dependent cocones. Rather than defining together the maps, the dependent cocones, and proofs that the maps are defined by the respective dependent cocones, we prefer to construct only the dependent cocones during induction, materializing their induced maps t_B^{n+1} and t_A^{n+1} only when necessary.

Definition 4.3.3. Given a natural number n, the type of **section cocones** at stage n is the type of triples (d_B^n, d_A^n, R^n), where

$$d_B^n : (b : B) \to \text{dep-cocone}(\text{pushout}(\mathcal{T}_B^{n+1}(b)), Q_{\Sigma B}(b, \iota_B^{n+1}(-)))$$
$$d_A^n : (a : A) \to \text{dep-cocone}(\text{pushout}(\mathcal{T}_A^{n+1}(a)), Q_{\Sigma A}(a, \iota_A^{n+1}(-)))$$

are families of dependent cocones over the standard pushout cocones of the span diagrams defining $P_B^{n+1}(b)$ and $P_A^{n+1}(a)$, respectively, and R^n is an identification between the vertical map of $d_A^n(fs)$ applied to $p : P_B^{n+1}(gs)$ and the element

$$(Q_{\Sigma S}(s, \iota_A^{n+1}(p \bullet_{n+1} \overline{s})))^{-1}(\Phi^n(s, p, \text{dep-cogap}(d_B^n(gs), p))).$$

Construction 4.3.4. For any natural number n, construct a section cocone at stage n. Begin by case splitting on n. Define the left map and coherence of d_B^0 by ex-falso, and the right map by pattern matching on $p : P_A^0(fs)$, and filling the goal with $\text{tr}_{Q_{\Sigma B}(gs)}(C^0(s, \text{refl}), Q_{\Sigma S}(s, \text{refl}_\infty)(q_0))$. For the successor case, define the left and right maps of d_B^{n+1} like in the informal description, replacing $t_B^{n+1}(b)$ by dep-cogap$(d_B^n(b))$ and $t_A^{n+1}(a)$ by dep-cogap$(d_A^n(a))$. Similarly, define the left and right maps and coherences of d_A^0 and d_A^{n+1} following the informal description, replacing calls to t_B^{n+1} and t_A^n with the cogap maps of the appropriate dependent cocones. Use the reflexive homotopy for the witnesses R^0 and R^{n+1}.

Finally, construct the coherence of d_B^{n+1} using the informal description, following the computation rule of dep-cogap(d_A^{n+1}) by the identification R^n to get to the desired shape of the right-hand side of the coherence.

The dependent cocones induce maps t_B^{n+1} and t_A^{n+1}, for which we can add the base cases to get the maps t_B^n and t_A^n.

Construction 4.3.5. Construct the maps

$$t_B^n : (b : B)(p : P_B^n(b)) \to Q_{\Sigma B}(b, \iota_B^n(p))$$
$$t_A^n : (a : A)(p : P_A^n(a)) \to Q_{\Sigma A}(a, \iota_A^n(p))$$

by induction on n.

In the zero case, define

$$t_B^0(b) :\equiv \text{ex-falso}$$
$$t_A^0(a_0, \text{refl}) :\equiv q_0,$$

and in the successor case, define

$$t_B^{n+1}(b) :\equiv \text{dep-cogap}(d_B^n(b))$$
$$t_A^{n+1}(a) :\equiv \text{dep-cogap}(d_A^n(a)).$$

The coherences K_B^n and K_A^n can be recovered from the computation rules of t_B^{n+1} and t_A^{n+1}, respectively.

Construction 4.3.6. Define the family of coherences

$$K_B : (n : \mathbb{N})(b : B)(p : P_B^n(b)) \to \text{tr}_{Q_{\Sigma B}(b)}(\kappa_B^n(p), t_B^n(b, p)) = t_B^{n+1}(b, \text{incl}_B^n\, p)$$

by case analysis on n. In the zero case, define

$$K_B^0(b) :\equiv \text{ex-falso},$$

and in the successor case, unfold the definition of $t_B^{n+2}(b)$ as the dependent cogap of the cocone $d_B^{n+1}(b)$, which comes equipped with the computation rule

$$t_B^{n+2}(\text{incl}_B^{n+1}(p)) = \text{tr}_{Q_{\Sigma B}(b)}(\kappa_B^{n+1}(p), t_B^{n+1}(b, p)),$$

which may be inverted to get the desired identification.

Construction 4.3.7. Define the family of coherences

$$K_A : (n : \mathbb{N})(a : A)(p : P_A^n(a)) \to \text{tr}_{Q_{\Sigma A}(a)}(\kappa_A^n(p), t_A^n(a, p)) = t_A^{n+1}(a, \text{incl}_A^n\, p)$$

by case analysis on n. In both cases, the identification $K_A^\bullet(a, p)$ is the inverse of the computation rule of t_A^{n+1} as the dependent cogap of $d_A^n(a)$.

The maps and coherences fit together to define dependent cocones under the sequential diagrams $P_B^\bullet(a)$ and $P_A^\bullet(b)$, which induce dependent maps out of their respective colimits.

Construction 4.3.8. Define the maps

$$t_A : (a : A)(p : P_A^\infty(a)) \to Q_{\Sigma A}(a, p)$$
$$t_B : (b : B)(p : P_B^\infty(b)) \to Q_{\Sigma B}(b, p)$$

using the dependent universal property of sequential colimits of $P_A^\infty(a)$ and P_B^∞, from the families of dependent cocones

$$(\lambda a \to (\lambda n \to t_A^n, \lambda n \to K_A^n)) : (a : A) \to \text{dep-cocone}\mathbb{N}(P_A^\bullet(a), Q_{\Sigma A}(a))$$
$$(\lambda b \to (\lambda n \to t_B^n, \lambda n \to K_B^n)) : (b : B) \to \text{dep-cocone}\mathbb{N}(P_B^\bullet(b), Q_{\Sigma B}(b)).$$

It remains to define the families of identifications t_S^n and coherences K_S^n.

Lemma 4.3.9. *For all natural numbers n, elements $s : S$ and $p : P_A^n(fs)$, there is an identification*

$$t_B^{n+1}(gs)(p \bullet_n s) = \mathrm{tr}_{Q_{\Sigma B}(gs)}(C^n(s,p), Q_{\Sigma S}(s, \iota_A^n(p), fs, t_A^n(p))).$$

Proof. Doing case analysis on n allows the underlying dependent cocones to compute, and the identification holds by the computation rule of dep-cogap. □

Construction 4.3.10. Given a natural number n, construct the family of identifications

$$t_S^n : (s : S)(p : P_A^n(fs)) \to Q_{\Sigma S}(s, \iota_A^n(p), t_A(fs, \iota_A^n(p))) = t_B(gs, (\iota_A^n(p)) \bullet_\infty s)$$

by concatenating

$$\begin{array}{c} Q_{\Sigma S}(s, \iota_A^n(p), t_A(fs, \iota_A^n(p))) \\ \| \\ Q_{\Sigma S}(s, \iota_A^n(p), t_A^n(fs, p)) \\ \| \\ t_B(gs, (\iota_A^n(p)) \bullet_\infty s), \end{array}$$

where the top identification is obtained from the computation rule for maps out of sequential colimits defined by dependent cocones, and the bottom identification follows from the observation that there is an identification

$$\begin{array}{c} \mathrm{tr}_{Q_{\Sigma B}(gs)}(C^n(s,p), Q_{\Sigma S}(s, \iota_A^n(p), t_A^n(fs, p))) \\ \| \\ t_B^{n+1}(gs, p \bullet_n s) \\ \| \\ t_B(gs, \iota_B^{n+1}(p \bullet_n s)) \\ \| \\ \mathrm{tr}_{Q_{\Sigma B}(gs)}(C^n(s,p), t_B(gs, (\iota_A^n(p)) \bullet_\infty s)), \end{array}$$

where the top identification is Lemma 4.3.9, the middle identification is the computation rule of t_B, and the bottom identification is the inverse of $\mathrm{apd}_{t_B(gs)}(C^n(s,p))$. We may "unapply" the transport from the composite, because transporting is an equivalence.

At the time of writing, the coherences K_S^n have not been informally nor formally constructed, neither here nor in available published literature by Wärn or anyone else. We leave their existence as a conjecture, and claim only a partial proof of correctness.

Conjecture 4.3.11. *The coherences K_S^n exist.*

Construction 4.3.12. Construct the family of coherences

$$t_S : (s : S)(p : P_A^\infty(fs)) \to Q_{\Sigma S}(s, p, t_A(fs, p)) = t_B(gs, p \bullet_\infty s)$$

using the dependent universal property of $P_A^\infty(fs)$, from the families of dependent cocones

$$(\lambda s \to (\lambda n \to t_S^n, \lambda n \to K_S^n)) :$$
$$(s : S) \to \text{dep-cocone}\mathbb{N}(P_A^\bullet(fs), \lambda p \to (Q_{\Sigma S}(s, p, t_A(fs, p)) = t_B(gs, p \bullet_\infty s)))$$

Theorem 4.3.13. *Assuming Conjecture 4.3.11 holds, the descent data $(P_A^\infty, P_B^\infty, - \bullet_\infty s)$ is an identity system at* $\text{refl}_\infty(a_0)$.

Proof. The data (t_A, t_B, t_S) form an element of $\text{sect}(Q_{\Sigma A}, Q_{\Sigma B}, Q_{\Sigma S})$, which makes $(P_A^\infty, P_B^\infty, - \bullet_\infty s)$ an identity system by Lemma 2.4.9. □

Corollary 4.3.14. *Assuming Conjecture 4.3.11 holds, there are families of equivalences*

$$e_A : (a : A) \to (\text{inl}(a_0) = \text{inl}(a)) \simeq P_A^\infty(a)$$
$$e_B : (b : B) \to (\text{inl}(a_0) = \text{inr}(b)) \simeq P_B^\infty(b)$$

satisfying $e_A(a_0, \text{refl}) = \text{refl}_\infty$, *and for every $s : S$ a commuting square*

$$\begin{array}{ccc} (\text{inl}(a_0) = \text{inl}(fs)) & \xrightarrow{e_A(fs)} & P_A^\infty(fs) \\ {\scriptstyle -\bullet(Hs)}\downarrow & & \downarrow{\scriptstyle -\bullet_\infty s} \\ (\text{inl}(a_0) = \text{inr}(gs)) & \xrightarrow[e_B(gs)]{} & P_B^\infty(gs). \end{array}$$

Proof. By Theorem 2.4.11 and Theorem 4.3.13, there is a unique equivalence of descent data $e : (I_A, I_B, I_S) \simeq (P_A^\infty, P_B^\infty, - \bullet_\infty -)$ sending refl to refl_∞. □

Chapter 5

Conclusion

The goal of the thesis was to improve existing infrastructure for working with homotopy pushouts in the agda-unimath library, and formalize existing results from the literature. Specifically the descent property, the flattening lemma, and the induction principle of identity types of pushouts were successfully formalized, properly documented, and integrated into the library. In an attempt to formalize the zigzag construction of identity types of pushouts, infrastructure for coequalizers and sequential colimits was developed. The zigzag construction was formalized, but it has not been shown to satisfy the induction principle of identity types of pushouts. Progress was made, but one coherence condition remains unproven.

Future work

There is an obvious step to continue the presented work, which is to complete the proof of correctness of the zigzag construction. Additional insight into strategies of its proof might be necessary, for example using the calculus of dependent identifications to abstract the transports, or formulating general lemmas about type families over sequential diagrams connected by a zigzag. This abstraction has not been explored in the thesis for its perceived low applicability.

The inductive definition of d_B^n and d_A^n could be improved by removing contractible data, i.e. the right map of d_A^n and the witness R^n.

The thesis also does not attempt to formalize applications of the zigzag construction. Wärn presents truncatedness and connectivity results as applications of the construction. For reasoning about the zigzag construction, more results about sequential colimits might be necessary, particularly the full "generalized flattening lemma" [17]. While the paper presents a proof in Homotopy Type Theory, it does so in a context with judgmental computation rules for sequential colimits, which are not present in our setting.

Bibliography

1. ACZEL, Peter. On Voevodsky's Univalence Axiom. In: *Mathematical Logic: Proof Theory, Constructive Mathematics*. European Mathematical Society - EMS - Publishing House GmbH, 2012, vol. 8, pp. 2963–3002. No. 4. ISSN 1660-8941. Available from DOI: 10.4171/owr/2011/52 (cit. on p. 10).
2. AGDA DEVELOPERS. *Agda*. 2024. Version 2.6.4. Available also from: https://agda.readthedocs.io/ (cit. on pp. v, 3, 63).
3. AWODEY, Steve; GAMBINO, Nicola; SOJAKOVA, Kristina. Inductive Types in Homotopy Type Theory. In: *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*. New Orleans, Louisiana: IEEE Computer Society, 2012, pp. 95–104. LICS '12. ISBN 9780769547695. Available from DOI: 10.1109/LICS.2012.21 (cit. on p. 19).
4. AWODEY, Steve; GARNER, Richard; MARTIN-LÖF, Per; VOEVODSKY, Vladimir. Mini-Workshop: The Homotopy Interpretation of Constructive Type Theory. *Oberwolfach Reports*. 2011, vol. 8, no. 1, pp. 609–638. ISSN 1660-8941. Available from DOI: 10.4171/owr/2011/11 (cit. on p. 3).
5. BRUNERIE, Guillaume. *On the homotopy groups of spheres in homotopy type theory*. 2016. Available from arXiv: 1606.05916 [math.AT]. PhD thesis. Laboratoire Jean-Alexandre Dieudonné (cit. on pp. v, 3).
6. HURKENS, Antonius J. C. A Simplification of Girard's Paradox. In: *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*. Berlin, Heidelberg: Springer-Verlag, 1995, pp. 266–278. TLCA '95. ISBN 354059048X (cit. on p. 5).
7. KRAUS, Nicolai; von RAUMER, Jakob. Path spaces of higher inductive types in homotopy type theory. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science*. Vancouver, Canada: IEEE Press, 2019. LICS '19 (cit. on pp. v, 28, 33).
8. LICATA, Daniel R.; SHULMAN, Michael. Calculating the Fundamental Group of the Circle in Homotopy Type Theory. In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. 2013, pp. 223–232. ISSN 1043-6871. Available from DOI: 10.1109/LICS.2013.28 (cit. on p. 10).
9. MARTIN-LÖF, Per. An Intuitionistic Theory of Types: Predicative Part. In: ROSE, H.E.; SHEPHERDSON, J.C. (eds.). *Logic Colloquium '73*. Elsevier, 1975, vol. 80, pp. 73–118. Studies in Logic and the Foundations of Mathematics. ISSN 0049-237X. Available from DOI: 10.1016/S0049-237X(08)71945-1 (cit. on p. 3).

10. NORDSTRÖM, Bengt; PETERSSON, Kent; SMITH, Jan M. *Programming in Martin-Löf's type theory: an introduction*. USA: Clarendon Press, 1990. ISBN 0198538146 (cit. on p. 3).

11. PAULIN-MOHRING, Christine. Inductive Definitions in the system Coq - Rules and Properties. In: *Proceedings of the International Conference on Typed Lambda Calculi and Applications*. Berlin, Heidelberg: Springer-Verlag, 1993, pp. 328–345. TLCA '93. ISBN 3540565175 (cit. on p. 5).

12. RIJKE, Egbert. *Classifying Types*. 2019. Available from arXiv: 1906.09435 [math.LO]. PhD thesis. Carnegie Mellon University (cit. on pp. v, 3, 19).

13. RIJKE, Egbert. *Introduction to Homotopy Type Theory*. 2022. Available from arXiv: 2212.11082 [math.LO] (cit. on pp. 3, 5, 83).

14. RIJKE, Egbert. *Introduction to Homotopy Type Theory*. 2022. Available also from: https://github.com/martinescardo/HoTTEST-Summer-School/blob/06dab2ca8ea0c760ac2dcb40006c280d619e5368/HoTT/hott-intro.pdf. Unpublished preprint used as study material for the HoTTEST Summer School 2022 (cit. on pp. 18, 19).

15. RIJKE, Egbert; STENHOLM, Elisabeth; PRIETO-CUBIDES, Jonathan; BAKKE, Fredrik; ŠTĚPANČÍK, Vojtěch, et al. *The agda-unimath library*. 2024. Available also from: https://github.com/UniMath/agda-unimath/. Accessed on 2024-17-07 (cit. on pp. v, 3).

16. SOJAKOVA, Kristina. Higher Inductive Types as Homotopy-Initial Algebras. *SIGPLAN Not.* 2015, vol. 50, no. 1, pp. 31–42. ISSN 0362-1340. Available from DOI: 10.1145/2775051.2676983 (cit. on p. 19).

17. SOJAKOVA, Kristina; van DOORN, Floris; RIJKE, Egbert. Sequential Colimits in Homotopy Type Theory. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. Saarbrücken, Germany: Association for Computing Machinery, 2020, pp. 845–858. LICS '20. ISBN 9781450371049. Available from DOI: 10.1145/3373718.3394801 (cit. on pp. v, 3, 42, 47, 51, 57, 79, 84).

18. THE UNIVALENT FOUNDATIONS PROGRAM. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013 (cit. on pp. 3, 5, 9).

19. WÄRN, David. *Path Spaces of Pushouts*. 2023. Available also from: https://dwarn.se/po-paths.pdf. Accessed on 2023-30-09 (cit. on pp. v, 3, 4, 51, 63, 66).

20. WÄRN, David. *Path Spaces of Pushouts*. 2024. Available from arXiv: 2402.12339 [math.AT] (cit. on pp. 3, 66).

Appendix A

List of attachments

Attachments consist of Git diff files, most of which were submitted as pull requests to the agda-unimath GitHub repository. They may be found in the `attachments` directory in the thesis source repository, and should also be included with any distribution of the thesis.

- `535-refactor-descent-circle.diff` (PR) updates formalization of the descent property of the circle to be consistent with its description in [13].
- `709-descent-circle.diff` (PR) documents the above and extends the formalization with characterization of other kinds of type families over the circle.
- `724-foundation-precomp-concat-squares.diff` (PR) shows commutativity of pasting and exponentiation of commuting squares.
- `725-foundation-horizontal-paste-pushouts.diff` (PR) formalizes horizontal pasting of pushouts squares.
- `755-foundation-vertical-paste-pushouts.diff` (PR) formalizes vertical pasting of pushouts squares.
- `758-foundation-unpaste-pushouts.diff` (PR) formalizes the inverse direction of pasting properties of pushout squares.
- `764-flattening-pushouts-family.diff` (PR) formalizes the flattening lemma for pushouts, without descent data.
- `792-coequalizers.diff` (PR) introduces coequalizers to the extent described in the thesis.
- `816-flattening-pushouts-descent-data.diff` (PR) formalizes the flattening lemma for pushouts, using descent data.
- `831-flattening-coequalizers.diff` (PR) refactors the flattening lemma for coequalizers to be derived from the flattening lemma for pushouts.
- `841-sequential-colimits.diff` (PR) defines and documents basic infrastructure of sequential diagrams and their colimits in terms of universal properties.

- `919-functoriality-sequential-colimits.diff` (PR) adds commuting prisms and formalizes functoriality of taking the standard sequential colimit, to the extent described in the thesis.
- `972-flattening-sequential-colimits-family.diff` (PR) formalizes the flattening lemma for sequential colimits, without descent data.
- `978-refactor-functoriality-sequential-colimits.diff` (PR) generalizes functoriality of sequential colimits from only the standard ones to arbitrary cocones satisfying the universal property.
- `988-refactor-lifts-families.diff` (PR) refactors and documents formalization of lifts of families of elements, which underpin the equivalence of the dependent and non-dependent pullback properties of pushouts.
- `1070-shifts-sequential-colimits.diff` (PR) formalizes shifts of sequential diagrams and related concepts, and its preservation of sequential colimits.
- `1098-refactor-coequalizers.diff` (PR) adds infrastructure for double arrows, and adopts coequalizers to use it.
- `1109-flattening-sequential-colimits-descent-data.diff` (PR) adds descent data over sequential diagrams and the flattening lemma for sequential colimits, using descent data.
- `1117-equivalences-sequential-comp.diff` (PR) formalizes the fact that equivalences are closed under sequential composition.
- `1129-zigzags-sequential-diagrams.diff` (PR) formalizes zigzags of sequential diagrams, to the extent described in the thesis.
- `1144-sequential-colimits-overview.diff` (PR) adds a summary page referencing formalization of sequential colimits following the outline of the paper by Sojakova, van Doorn and Rijke [17].
- `1145-descent-pushouts.diff` (PR) refactors, extends and documents the descent property of pushouts.
- `1148-characterization-families-pushouts.diff` (PR) adds characterizations of families of functions and families of equivalences over pushouts, and the family of based identity types.
- `1150-identity-systems-descent-pushouts.diff` (PR) formalizes the induction principle of identity types of pushouts, using descent data.
- `zigzag-construction-identity-types.diff` formalizes the zigzag construction and its partial proof of correctness, to the extent of the thesis. May be applied on top of latest `master` branch of agda-unimath as of writing, commit `98119d71307593f533f2539882430f157330c07e`.