HOMOTOPY LIMITS IN TYPE THEORY

JEREMY AVIGAD, KRZYSZTOF KAPULKIN, AND PETER LEFANU LUMSDAINE

Abstract. Working in homotopy type theory, we provide a systematic study of homotopy limits of diagrams over graphs, formalized in the Coq proof assistant. We discuss some of the challenges posed by this approach to formalizing homotopy-theoretic material. We also compare our constructions with the more classical approach to homotopy limits via fibration categories.

Contents

Introduction

Homotopy type theory is based on the discovery that formal dependent type theory has a natural homotopy-theoretic interpretation ([Voe06], [AW09]). Since a number of interactive proof assistants implement versions

Date: March 1, 2013.

of dependent type theory, these observations open the possibility of developing parts of homotopy theory formally with the help of such assistants; see [PW12] for a helpful overview.

In this spirit, we carry out a number of homotopy-theoretic constructions in a core system of homotopy type theory. In particular, we define and investigate (homotopy) pullbacks, equalizers, limits over graphs, pointed spaces, and fiber sequences. The entire development is formalized with the Coq interactive proof assistant. Besides the formalization itself, we also compare the semantics of type theory with fibration categories, a standard homotopy-theoretic setting for the construction of homotopy limits.

We assume some familiarity with type theory, but not specifically with the homotopical version; we do not assume any previous acquaintance with homotopy limits.

We should mention that many of the facts we present below are already known in folklore; in any case, none of them will be unexpected to researchers in the field. Egbert Rijke and Bas Spitters [RS13] have also independently investigated limits and colimits over graphs within a similar type theory. We hope it will prove useful, however, to have a systematic treatment of these basic results, fully formalized in Coq and available as a library for future use. We also hope that the practical lessons we learned during the formalization process may be useful to others.

Our Coq development builds on a library for homotopy type theory developed jointly by various people, under the leadership of Andrej Bauer, Lumsdaine, and Michael Shulman [HoT]. Another extensive library has been developed by Vladimir Voevodsky [Voe], and some of our verified results overlap his.

Outline. In Section 1, we set out the the formal framework of our work: the type theory under consideration, and its intended interpretation. Along with this, we very briefly review homotopy limits in the classical setting. In Section 2, we recall some key constructions from the type-theoretic development of homotopy theory, and use these to show that every categorical model of the theory carries the structure of a fibration category. Section 3 presents the main body of our formalization: a concise treatment of the content, in traditional mathematical prose. Finally, in Section 4 we share some reflections on practical aspects of the formalization process.

Our formal development in Coq can be found in the files associated with the journal publication of this paper, and also online at

`https://github.com/peterlefanulumsdaine/hott-limits/tree/v1`.

The Github version will be maintained for compatibility with Coq and the HoTT library.

References to the formal code are typeset in `a teletype font`. For brevity, we omit the `.v` extension from filenames, so that, for example, `Fundamentals.v` is cited as `Fundamentals`. Often a single lemma in the informal presentation below translates to a cluster of formal lemmas in our files, in which case we simply cite a representative element of that cluster.

Acknowledgements. Most of this work was carried out during the Special Year on Univalent Foundations at the Institute for Advanced Study, and all three authors are grateful to the Institute for its hospitality and support. We are also grateful to two anonymous referees for numerous helpful comments, corrections, and suggestions.

Avigad's work has been partially supported by NSF grant DMS-1068829 and AFOSR grant FA9550-12-1-0370. Kapulkin was supported by NSF Grant DMS-1001191 (P.I. Steve Awodey) and a grant from the Benter Foundation (P.I. Thomas Hales). Lumsdaine was supported by NSF grant DMS-1128155. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Kapulkin dedicates this work to his mother.

1. Background

In this section, we first lay out the specific logical system in which we will work. We then review its intended semantics, insofar as they are relevant to working within the theory, and fix the basic notation and terminology we will use, based on the intended semantics. Finally, we very briefly review classical homotopy limits.

1.1. Logical setting.

We assume familiarity with some form of dependent type theory. Specifically, we will work in the system of *predicative Martin-Löf type theory* [ML84]. The following types, and associated rules, form a minimal core to that system:

(1) dependent products $\Pi_{x:A} B$, and the associated introduction, elimination, and computation rules
(2) dependent sums $\Sigma_{x:A} B$, and the associated introduction, elimination, and computation rules
(3) identity types Id_A, and the associated introduction, elimination, and computation rules

Most developments in homotopy type theory add at least the following rule:

(4) function extensionality: for any type A, any type B depending on $x : A$, and functions $f, g : \Pi_{x:A} B$, if $fx = gx$ for every $x : A$, then $f = g$.

The system based on these rules is used in [AGS12], where it is denoted by $\mathcal{H}$; it also forms a sufficient basis for much of the present formalization. Some of our constructions, in addition, depend on:

(5) the type Nat of natural numbers, with the usual introduction, elimination, and computation rules,

from which the empty type, unit type, and other finite types can be defined; and finally, some definitions presuppose the existence of:

(6) a universe U of types, containing Nat, and closed under the formation of dependent products, sums, and identity types.

We use quantification over the universe to define the universal properties of pullbacks and limits, but also give equivalent formulations that do not make use of such a universe.

Besides these, the version of Coq we used implements η-conversion for functions, $\lambda x.fx = f$, as a built-in conversion rule. As a propositional equality, it is derivable from function extensionality, so we do not believe its use is essential; however, since it is unavoidably present in the proof assistant, we include it in our formal theory.

In sum, if we take axioms (1)–(3) to represent the core of Martin-Löf type theory, ML, it is then reasonable to denote our overall framework as

$$\mathsf{ML} + (\mathsf{funext}) + (\eta) + (\mathsf{Nat}) + (\mathsf{U}).$$

For brevity, we refer to this in the present paper as $\mathcal{H}'$; thus the formal content of our work is that the constructions and assertions of Sections 2 and 3 are consequences of this formal theory. As noted above, however, most of our results do not require Nat, and many do not require U.

We do not consider in the present work extra axioms such as Univalence, resizing, or higher inductive types.

One final comment about the formal verification: rather than providing Id, Nat, Bool, and so on individually, Coq provides a general mechanism for defining inductive types, which these are then defined as instances of. However, the resulting eliminators for these types correspond precisely to the rules for them described above. Coq also provides (dependent) record types, as syntactic sugar for certain inductive types; in some cases, using record types made type checking more efficient, and brought notational benefits as well. As these may be routinely translated into (iterated) Σ-types, their use has no bearing on the question of derivability in $\mathcal{H}'$.

1.2. **Semantics.**

1.2.1. *General algebraic semantics.* The fully general semantics of dependent type theories are, from a purely algebraic point of view, well-understood. Essentially, a model of any dependent type theory $\mathbf{T}$ with the same basic judgements and structural rules as $\mathcal{H}'$ may be defined as a *contextual category*—that is, a category equipped with structure sufficient to model the structural rules—along with further algebraic structure corresponding to the logical constructors and axioms of $\mathbf{T}$. For the details of this definition, see [Str91]; for brevity, we will refer to such a structure as a *categorical model of* $\mathbf{T}$.

The justification for calling such structures models comes from the fact that the syntax of the theory forms an initial such structure:

Definition 1.2.1. Given any dependent type theory $\mathbf{T}$, the *syntactic category* $\mathcal{C}(\mathbf{T})$ is given as follows:

- objects of $\mathcal{C}(\mathbf{T})$ are contexts $[x_1{:}A_1,\ \ldots,\ x_n{:}A_n]$ of $\mathbf{T}$, up to definitional equality and renaming of free variables;
- maps of $\mathcal{C}(\mathbf{T})$ are *context morphisms* (a.k.a. *substitutions*), again up to definitional equality and renaming of free variables. That is, a map

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

  is represented by a sequence of terms

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

Moreover, $\mathcal{C}(\mathbf{T})$ may naturally be given the structure of a contextual category; for each logical rule of $\mathbf{T}$, $\mathcal{C}(\mathbf{T})$ carries the corresponding algebraic structure.

Fact 1.2.2 ([Str91][1]). *$\mathcal{C}(\mathbf{T})$ is initial among categorical models of $\mathbf{T}$.*

Thus any other categorical model $\mathbf{C}$ has a canonical structure-preserving functor from $\mathcal{C}(\mathbf{T})$—that is, an interpretation function, interpreting the syntax of $\mathbf{T}$ in $\mathbf{C}$.

1.2.2. *Homotopical semantics.* Homotopy type theory is based on the realization ([HS98], [AW09], [vdBG12], [Voe10]) that various homotopy-theoretic settings give natural examples of such categorical models. Very roughly, a type A denotes a space; a family $B(x)$ of types, depending on some variable x of type A, denotes a fibration over A; a term $t(x)$ of type A', again dependent on a variable $x : A$, denotes a continuous map from A to A'; a term $t(x)$ of type $B(x)$, dependent on $x : A$, denotes a section of the corresponding fibration over A; and so on.

The main motivating interpretation, for us, is the model in simplicial sets—one of the most well-studied models of spaces in homotopy theory. The full details of this interpretation are rather technical, so since we never require them, we omit them here; see [KLV12] for a complete presentation of the simplicial set model, and [Shu14] for more general related models. We sketch here just the main ingredients of the interpretation, insofar as they justify the intuition and terminology for working within the theory.

In this model, closed types (and, more generally, contexts) are interpreted as Kan complexes; dependent types, as Kan fibrations. Most type formers—Π-types, Σ-types, Nat, etc.—are interpreted as in the more familiar topos

[1]Unfortunately, to our knowledge, no general form of this result exists in the literature; it is shown for certain specific type theories in [Str91] and elsewhere, and its extension to other combinations of the standard rules (such as $\mathcal{H}'$) is well-known in folklore.

logic: Π-types by the right adjoint to pullback, Σ-types by the left, Nat by the natural numbers object, and so on.

The main novelty, however, is the interpretation of the identity type $\mathsf{Id}_A(x, y)$ with variables x and y from A. In set- and topos-theoretic models, one would interpret it as the diagonal map $A \longrightarrow A \times A$. However, in simplicial sets (and other homotopy-theoretic settings) this map is hardly ever a fibration. It can, however, be replaced by a fibration $P(A) \longrightarrow A \times A$, where $P(A)$ is the *path object* of A; this is then used to interpret the identity type of A. Thinking of a simplicial set as a space, $P(A)$ represents the space of paths in A, with the fibration $P(A) \longrightarrow A \times A$ giving the indexing of paths over their endpoints. In particular contrast to the set-theoretic situation, for given $x, y : A$ the space $P(A)(x, y)$ of paths from x to y may be not a mere proposition, but a non-trivial space in its own right.

1.3. **Notation and terminology.**

Our choices of notation and terminology are guided by the homotopical interpretation. In particular, we will write $p : (x \rightsquigarrow y)$ for the identity type, to emphasize that we consider it as the type of paths from x to y. The Homotopy Type Theory library uses the notation `x = y` for this type, and in our Coq development, we stick with this. However, in the informal presentation below, we find it most natural to understand our constructions as constructions of paths, rather than equality proofs; and so we settle on the latter notation, and favor the word "path" over "equality."

In other respects, however, we have found it more convenient to leave the homotopy-theoretic interpretation implicit. For example, the natural definitions of pullbacks, equalizers, and limits in type-theoretic notation turn out to characterize homotopy pullbacks, homotopy equalizers, and homotopy limits in the homotopy-theoretic interpretation. Having kept the notion of "path" prominent, sprinkling the word "homotopy" everywhere seemed to impose an unnecessary burden; thus, both in code and in prose we refer just to "pullbacks," "equalizers," and "limits." (This is customary in higher category theory (see, e.g., [Lur09]), when one uses, for example, the word "limit" for an object that in strict terms is only a homotopy limit.)

For the sake of readability, we will use standard mathematical terminology and notation in Sections 2 and 3, rather than attempting to adhere closely to the notation used in the Coq code. Table 1 lists some of the basic notions of our development, comparing the notations used in our presentation here with those used in the Coq formalization.

As usual in homotopy type theory, we represent logic using *propositions-as-types*, with implication, conjunction, and universal and existential quantification interpreted in terms of function, product, Π-, and Σ-types respectively. Thus, for example, the functional extensionality axiom (Axiom 4 in

informal notion	mathematical notation	Coq notation
p is a path from x to y	$p : (x \rightsquigarrow y)$	`p : x = y`
identity path at x	$\mathsf{refl}(x)$	`idpath x`
concatenation of p and q	$p \centerdot q$	`p @ q`
inverse of p	$\overline{p}$	`!p`
B is a fibration over A	$B \twoheadrightarrow A$	`B : A -> Type`
total space of B over A	$\Sigma_{x:A} B(x)$	`{ x : A & B x }`
dependent product of B over A	$\Pi_{x:A} B(x)$	`forall x : A, B x`
e is an equivalence from A to B	$e \colon A \simeq B$	`e : A <~> B`
inverse of e	e^{-1}	`e^-1`
a universe of small types	U	`UU`
the natural numbers	Nat	`nat`

Table 1. Correspondence of notations

Section 1.1 above), is formally a constant of type:

$$\mathsf{funext} : \prod_{\substack{A:\mathsf{Type} \\ B:A\to\mathsf{Type}}} \; \prod_{f,g:\prod_{x:A} B(x)} \Big(\prod_{x:A} (fx \rightsquigarrow gx)\Big) \to (f \rightsquigarrow g).$$

Notice that Σ-types provide a useful way of "packaging" related pieces of data into a single type: to illustrate this, consider Definition 3.1.5 below. Formally, a *cospan* consists of types A, B, and C, and maps $f \colon A \longrightarrow C$, $g \colon B \longrightarrow C$. Given a type X, a *cone* over this cospan with vertex X consists of maps $h \colon X \longrightarrow A$ and $k \colon B \longrightarrow C$, and a family of paths $(f(hx) \rightsquigarrow g(kx))$ for each x in X. In other words, such a cone is an element of the type

$$\sum_{\substack{h:X\to A \\ k:X\to B}} \prod_{x:X} (f(hx) \rightsquigarrow g(kx)).$$

Thus our formal definition in Coq reads as follows:

```
Definition cospan_cone {A B C : Type} (f : A -> C)
  (g : B -> C) (X : Type)
:= { h : (X -> A) & { k : (X -> B)
   & forall x, paths (f(h x)) (g(k x)) }}.
```

The curly braces around the arguments `A`, `B`, and `C` indicate that these are treated as *implicit arguments*. This means that the user may write just `cospan_cone f g X`, leaving the system to infer `A`, `B`, and `C` from the types of `f` and `g`. Sometimes one needs to turn this feature off, and specify such arguments; writing `@cospan_cone A B C f g X` tells Coq to expect all the arguments of `cospan_cone` to be given explicitly.

1.4. **Classical homotopy limits.** For the reader unfamiliar with the classical theory of homotopy limits, we briefly survey here a few of its key points.

They may be seen as a solution to the problem that ordinary ("strict") limits are not invariant under homotopy equivalence: for instance, the two cospans below are homotopy equivalent, but their strict pullbacks are not.

$$\begin{tikzcd}
* \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & * \arrow[d] & & \emptyset \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & * \arrow[d, "1"] \\
* \arrow[r] & * & & * \arrow[r, "0"] & {[0,1]}
\end{tikzcd}$$

This may be resolved by instead defining the *homotopy pullback* $A \times_B^h C$, as the space of triples (a, c, p), where $a \in A$, $c \in C$, and p is a path in B from $f(a)$ to $g(c)$; the equalities in the definition of the strict pullback have been replaced by paths.

More generally, the homotopy limit of a functor $F \colon \mathcal{I} \longrightarrow \mathbf{Top}$ may be defined using the *end* formula $\int_{i \in \mathcal{I}} F(i)^{\mathbf{B}(\mathcal{I}/i)}$. This has the effect of replacing equalities by homotopies, in a coherent fashion; the coherence is encoded by the use of the classifying spaces $\mathbf{B}(\mathcal{I}/i)$. This generalizes to other settings, first by a similar concrete construction (in e.g. simplicial settings [BK72]), and more abstractly in terms of derived functors (for general homotopical categories [DHKS04]).

In the ∞-categorical setting, one may take an alternative approach, defining the (homotopy) limit by an ∞-categorical universal property directly generalizing that of ordinary 1-categorical limits (see e.g. [Lur09]). In Homotopy Type Theory, we do the same. It turns out, in fact, that at least for diagrams over graphs, what looks like the ordinary definition of a strict set-theoretic limit actually defines the homotopy limit—both as an explicit construction, and as a characterization via a universal mapping property.

2. Fibration categories from type theory

In this section and the next, we develop the basic theory of homotopy limits and related notions in $\mathcal{H}'$. We have already explained, in Section 1, how the basic ingredients are represented in the language of Coq, and complete details of the whole development can be found in the files comprising our formal verification. Especially in Section 3, therefore, we will generally only sketch most proofs, leaving out steps that are straightforward and routine (and even some that are not).

2.1. **Basic constructions.** Our formal work builds on the HoTT library [HoT] for homotopy theory developed by Bauer, Lumsdaine, Shulman, and others. We begin by summarizing some of the basic components of this library that are used throughout.

2.1.1. *Operations on paths.* Given any $x, y : X$, we write $p : (x \rightsquigarrow y)$ to denote that p is a path from x to y. For every x, there is an "identity path" $\mathsf{refl}(x) : (x \rightsquigarrow x)$. The central property characterizing the type of paths is its elimination principle, which says roughly that to construct an object of a type $C(x, y, p)$ depending on a path p from x to y, it suffices to construct an element of $C(x, x, \mathsf{refl}(x))$, in which p has been "contracted" to an identity path.

Paths admit various operations familiar from homotopy theory and higher category theory. Any two paths $p : (x \rightsquigarrow y)$ and $q : (y \rightsquigarrow z)$ can be concatenated, yielding a path $p \centerdot q : (x \rightsquigarrow z)$. Moreover, $\mathsf{refl}(x) : (x \rightsquigarrow x)$ is a unit element for this operation, and every path admits an inverse $\bar{p} : (y \rightsquigarrow x)$. These operations satisfy the groupoid laws, but, as in homotopy theory, only up to a higher path. For example, we can find an inhabitant of the type $(p \centerdot \bar{p} \rightsquigarrow \mathsf{refl}(x))$. In fact, every type, together with the tower of its paths, forms an ∞-groupoid of some sort; precise statements along these lines can be found in [vdBG11], [Lum09].

Moreover, the maps between types respect the paths and the structure on them. That is: given any $p : (x \rightsquigarrow y)$ in X and $f \colon X \longrightarrow Y$, we obtain a path $f[p] : (f(x) \rightsquigarrow f(y))$; and this is functorial, in the up-to-homotopy sense that there is, for example, an inhabitant of the type $(f[p \centerdot q] \rightsquigarrow f[p] \centerdot f[q])$.

2.1.2. *Equivalences and truncatedness.* The notion of paths allows us to recover several familiar notions from algebraic topology.

We can, for example, say that a type X is *contractible* if there is some $x_0 : X$, and a function giving for each $x : X$ a path $(x \rightsquigarrow x_0)$. Formally, the proposition "X is contractible" is defined as follows:[2]

$$\mathsf{isContr}(X) := \sum_{x_0 : X} \prod_{x : X} (x \rightsquigarrow x_0).$$

One can also construct the *homotopy fiber* of a map $f \colon X \longrightarrow Y$ over an element $y : Y$ by:

$$\mathsf{hfib}(f, y) := \sum_{x : X} (f(x) \rightsquigarrow y).$$

Given these we say that a map $f \colon X \longrightarrow Y$ is an *equivalence* if for all $y : Y$ the homotopy fiber of f over y is contractible. The HoTT library provides many crucial results on equivalences. For example, a map is an equivalence exactly if it has a two-sided inverse (up to homotopy), or alternatively two one-sided inverses.

[2]One might at first read this as a definition of connectedness—for each x, there exists some path from x to x_0—but remember that one should think of the function sending x to the path as *continuous*, so as giving a contraction of X to x_0. Precisely, in the simplicial and similar interpretations, the Π-type becomes a space of continuous functions, and so $\mathsf{isContr}$ gets interpreted as the property of contractibility; and moreover, working within the theory, the logic forces $\mathsf{isContr}$ to behave like contractibility, not like connectedness.

Another notion that smoothly transfers from algebraic topology to HoTT is the notion of an n-type. Classically, an n-type is a space whose homotopy groups vanish above dimension n. In HoTT we define an analogous hierarchy.

Precisely, *n-truncatedness* is defined by induction for $n \geq -2$. A type X is (-2)-truncated if it is contractible; and is $n+1$-truncated if for all $x, y : X$, the type $(x \rightsquigarrow y)$ of paths from x to y is of n-truncated. For short, we refer to n-truncated types as *n-types*. In particular, (-1)-types may be considered as propositions, carrying no more information than the fact of being inhabited; and 0-types as (up-to-homotopy) discrete sets. We call such types *propositions* (or *mere* propositions, for emphasis) and *sets* respectively.

2.1.3. *Functional extensionality.* Given two types X and Y, the type $X \to Y$ of maps between them can be equipped with the notion of a path (or rather, a "homotopy") in two different ways. First, for any $f, g \colon X \longrightarrow Y$, one can form $(f \rightsquigarrow g)$, in the usual way. On the hand, one can also compare two functions pointwise, asking for an element of $\prod_{x:X} (f(x) \rightsquigarrow g(x))$; we call such a function h a *homotopy* from f to g, and write $h : f \Rightarrow g$.

Given any $p : (f \rightsquigarrow g)$, we obtain by the elimination principle for paths an element of the type $f \Rightarrow g$. The functional extensionality axiom implies that this assignment is an equivalence; that is, that given a pointwise homotopy between two maps, we can always find a path between them in the function type inducing the original homotopy. More generally, functional extensionality implies this equivalence between paths and homotopies in *dependent* function types $\prod_{x:A} B(x)$.

2.1.4. *Dependent sums.* The interaction between dependent sums and paths is crucial in our work. Let $B(x)$ be a type depending on $x : A$. It is easy to see that then a path $p : (a \rightsquigarrow a')$ in A induces an equivalence $p_! \colon B(a) \longrightarrow B(a')$, which we call *transport* between fibers. As everything before, this commutes appropriately with the operations on paths; for example, for any $p : (a \rightsquigarrow a')$ and $q : (a' \rightsquigarrow a'')$, and $b : B(a)$ we have $((p \centerdot q)_! b \rightsquigarrow q_!(p_! b))$.

This also provides a means to construct paths between two elements of a Σ-type. Given a path $p : ((a, b) \rightsquigarrow (a', b'))$ in a $\sum_{x:A} B(x)$, we get a pair of paths: $p_1 : (a \rightsquigarrow a')$ and $p_2 : ((p_1)_! b \rightsquigarrow b')$; and conversely, given such a pair of paths, we can recover the original path p. This construction is ubiquitous in the formalization, since so many objects are defined using Σ-types; for more discussion, see Section 4.3 below.

2.2. **Fibration category structure.** In this section, we show that any categorical model of $\mathcal{H}'$ (so, in particular, its syntactic category) satisfies the axioms of a fibration category, following the lines of results such as [GG08], [Lum11]. After this, we look at how some standard properties of fibration categories translate in terms of the type theory.

The results follow from a combination of internal reasoning—proving certain statements in the type theory—and external (meta-theoretic), showing how in models, the internal statements translate into the desired axioms. Since we will be switching back and forth frequently between these two different logical settings, we use sans serif text in this section to distinguish the internal reasoning from the external. The internal portions are formalized in the file `Fundamentals`.

We start by recalling the definition of a fibration category (for more on which, see [Bro73], [Bau89]):

Definition 2.2.1. A *fibration category* is a category $\mathbf{C}$ together with two distinguished classes of maps, $\mathcal{W}$ (the *weak equivalences*) and $\mathcal{F}$ (the *fibrations*) satisfying the following conditions:

(1) Weak equivalences satisfy the 2-out-of-6 condition; i.e., given a composable triple of morphisms

$$W \xrightarrow{f} X \xrightarrow{g} Y \xrightarrow{h} Z,$$

if $g \cdot f$ and $h \cdot g$ are weak equivalences, then so are f, g, h, and $h \cdot g \cdot f$.

(2) $\mathcal{F}$ is closed under composition.

(3) Calling a map that is both a weak equivalence and a fibration an *acyclic fibration*, all isomorphisms are acyclic fibrations.

(4) $\mathbf{C}$ has a terminal object $\mathbf{1}$.

(5) Pullbacks along fibrations exist; fibrations and acyclic fibrations are stable under pullback.

(6) For any object $X \in \mathbf{C}$, the diagonal morphism $\Delta \colon X \longrightarrow X \times X$ can be factored as a weak equivalence followed by a fibration:

$$X \longrightarrow PX \longrightarrow X \times X.$$

(Such a factorization, and by abuse of language also the object PX, is called a *path object for* X.)

(7) Every object is *fibrant*; that is, the unique map $X \longrightarrow \mathbf{1}$ is a fibration, for any $X \in \mathbf{C}$.

Remark 2.2.2. This is slightly stronger than the original definition given by Brown, in that it requires the class $\mathcal{W}$ to satisfy the 2-out-of-6 axiom rather than just the more familiar 2-out-of-3. However, once $\mathbf{C}$ satisfies all the other axioms, the following conditions are equivalent (the result is due to Cisinski; see [RB06, Thm. 7.2.7]):

(1) $\mathcal{W}$ satisfies 2-out-of-6;

(2) $\mathcal{W}$ satisfies 2-out-of-3 and is *saturated*; that is, if a map w of $\mathbf{C}$ becomes an isomorphism in $\mathrm{Ho}(\mathbf{C})$, then $w \in \mathcal{W}$.

In this section we show that any categorical model of $\mathcal{H}'$ (in the sense of Section 1.2) carries the structure of a fibration category; and so, in particular, the syntactic category $\mathcal{C}(\mathcal{H}')$ does. From here on, fix some categorical model $\mathbf{C}$ of $\mathcal{H}'$.

For convenience of exposition, we also assume in this section strong η-rules for Σ-types, so that every context is isomorphic to (a context consisting of just) a single iterated Σ-type: for instance,

$$[x{:}A,\ y{:}B(x)] \cong [p : \sum_{x:A} B(x)].$$

This allows us to work just with types, rather than with general contexts. However, nothing here depends on that assumption; one may simply replace types with contexts and Σ-types with context extensions, in particular in the definition of the fibrations:

Definition 2.2.3 (Gambino–Garner [GG08])**.** A map of $\mathbf{C}$ is a *fibration* if it is isomorphic to some composite of first projections from Σ-types,

$$\sum_{x:A} B(x) \to A.$$

Denote the class of fibrations by $\mathcal{F}$.

(This is a slight simplification of Gambino and Garner's original definition, which also closes $\mathcal{F}$ under retracts.) Note that "isomorphic" here refers to the external notion of isomorphism in $\mathbf{C}$, involving definitional equality of maps; and so one cannot represent this definition internally in the type theory, since definitional equality is not represented by a type. Indeed, there is no way of defining these fibrations internally: every statement of the type theory respects equivalence, and we see in Lemma 2.2.11 below that every map is equivalent to a fibration.

Weak equivalences, by contrast, are defined first internally, as in Section 2.1 above:

Definition 2.2.4 (Voevodsky)**.** A map $f\colon A \longrightarrow B$ is an *equivalence* if for each $b : B$ the homotopy fiber $\mathsf{hfib}(f, b)$ is contractible.

(Note that this is simply a property of f, not extra structure, since being an equivalence is a proposition in the sense of Section 2.1.2.)

Take a map $f\colon A \longrightarrow B$ in $\mathbf{C}$ to be in $\mathcal{W}$ if "$(\lambda x.\ f(x))$ is an equivalence" holds in $\mathbf{C}$.

With these definitions, we are now ready for the main theorem of the section:

Theorem 2.2.5. *$\mathbf{C}$, with $\mathcal{W}$ and $\mathcal{F}$ as described above, is a fibration category.*

We consider the various axioms in turn.

Lemma 2.2.6. *$\mathcal{W}$ satisfies the 2-out-of-6 property.*

Proof. We first show the analogous statement internally (Lemmas `two_of_six_hgf`, `two_of_six_h`, `two_of_six_g`, and `two_of_six_f` in the formalization).

Let f, g, h be composable maps, and suppose $f \cdot g$ and $g \cdot h$ are equivalences. Then:

- $(g \cdot f)^{-1} \cdot g \cdot (h \cdot g)^{-1}$ gives a quasi-inverse for $h \cdot g \cdot f$;
- $(h \cdot g)^{-1} \cdot h$ and $f \cdot (g \cdot f)^{-1}$ give left and right inverses for g;
- $(g \cdot f)^{-1} \cdot g$ gives a quasi-inverse for f;
- $g \cdot (h \cdot g)^{-1}$ gives a quasi-inverse for h.

This immediately implies the desired external statement, since internal and external composition agree. □

Lemma 2.2.7. *Pullbacks of fibrations exist.*

Proof. The pullback of a dependent projection is given by substituting into the corresponding dependent type; that is, the following square is a pullback:

$$\begin{tikzcd} \sum_{x:A'} B(fx) \arrow[r] \arrow[d] & \sum_{x:A} B(x) \arrow[d] \\ A' \arrow[r, "f"] & A. \end{tikzcd}$$

The two pullbacks lemma implies that pullbacks of their composites then also exist. □

Note that this is an external statement: these really are strict pullbacks, in contrast to the internally defined pullbacks of Section 3.1, which from an external point of view become homotopy pullbacks.

Lemma 2.2.8 (`fiber_to_hfiber_equiv`)**.** *Let* $\pi_1 \colon \sum_{x:A} B(x) \longrightarrow A$ *be a fibration. Then for any* $a : A$*, we have* $B(a) \simeq \mathsf{hfib}(\pi_1, a)$*.*

Proof. Take any $a : A$. For the map $B(a) \longrightarrow \mathsf{hfib}(\pi_1, a)$, send $b : B(a)$ to $((a, b), \mathsf{refl}(a))$. Conversely, send $((a', b), p) : \mathsf{hfib}(\pi_1, a)$ (where $b : B(a')$ and $p : (a' \rightsquigarrow a)$) to the transported element $p_!(b) : B(a)$. The verification that these are mutually inverse is straightforward. □

Lemma 2.2.9. *Fibrations and acyclic fibrations are preserved under pullback.*

Proof. Preservation of fibrations is clear by construction from the proof of Lemma 2.2.7. For acyclicity, suppose $\pi = \pi_1 \colon \sum_{x:A} B(x) \longrightarrow A$ is an acyclic fibration, and $f \colon A' \longrightarrow A$ is a map. Write $f^*\pi$ for the pullback fibration $\pi_1 \colon \sum_{x:A'} B(f(x)) \longrightarrow A'$. Then for any $x : A'$,

$$\mathsf{hfib}(f^*\pi, x) \simeq B(f(x)) \simeq \mathsf{hfib}(\pi, f(x))$$

by Lemma 2.2.8; and $\mathsf{hfib}(\pi, f(x))$ is contractible by hypothesis, so since equivalence preserves contractibility, $\mathsf{hfib}(f^*\pi, x)$ is again contractible. So $f^*\pi$ is again acyclic, as required. □

Lemma `str_pullback_pres_acyclic_fib` provides the internal part of this argument.

Definition 2.2.10. The *path type* of a type $A \in \mathbf{C}$ is constructed from its identity types:

$$\mathsf{P}A := \sum_{x,y:A} (x \rightsquigarrow y)$$

It is equipped by construction with a fibration π to $A \times A$, and there is also a natural map $r\colon A \longrightarrow \mathsf{P}A$ sending a to $((a,a), r(a))$. Moreover, the map $\mathsf{P}A \longrightarrow A$ sending $((a,a'),p)$ to a (or to a') gives a quasi-inverse for r; so together, we have a factorization of the diagonal of A as a weak equivalence followed by a fibration:

$$\begin{tikzcd} & \mathsf{P}A \arrow[dr, "\pi"] & \\ A \arrow[ur, "r"] \arrow[rr, "\Delta"'] & & A \times A \end{tikzcd}$$

We have now amassed all the ingredients of a fibration category:

Proof of Theorem 2.2.5. Immediate from the preceding lemmas. □

Besides the basic structure, we consider how a few more useful constructions from the theory of fibration categories play out in $\mathbf{C}$:

Lemma 2.2.11 (Factorization Lemma, [GG08, Lem. 11])**.** *For every morphism $f\colon A \longrightarrow B$ in $\mathbf{C}$, there exists a factorization:*

$$\begin{tikzcd} & \mathsf{P}f \arrow[dr, "p_f"] & \\ A \arrow[ur, "\sigma_f"] \arrow[rr, "f"'] & & B \end{tikzcd}$$

with $\sigma_f \in \mathcal{W}$ and $p_f \in \mathcal{F}$.

Proof. We take

$$\mathsf{P}f := \sum_{y:B, x:A} (fx \rightsquigarrow y).$$

and

$$\sigma_f(x) := (fx, x, \mathsf{refl}(fx)).$$

By definition, p_f is in $\mathcal{F}$; and it is easy to check that $\sigma_f \in \mathcal{W}$. □

$(\mathcal{W}, \mathcal{F})$ factorizations may be constructed in this way in any fibration category. In the type-theoretic case, however, they crucially satisfy an additional property, corresponding to the Id-elimination rule: σ_f is weakly left-orthogonal to fibrations, and so fibrations form the right class of a weak factorization system. We will not however go into this point here; see [GG08] for details.

Lemma 2.2.12 (`right_properness`)**.** *The pullback of a weak equivalence along a fibration is again a weak equivalence:*

$$\begin{tikzcd}
\pi^* C \arrow[r, "\pi^* f"] \arrow[d] & \sum_{x:A} B(x) \arrow[d, "\pi"] \\
A' \arrow[r, "f \in \mathcal{W}"'] & A
\end{tikzcd}$$

Proof. The map $\pi^* f$ sends a pair (y, b) to $(f(y), b)$; so taking a quasi-inverse (g, η, ϵ) for f, we can construct a quasi-inverse for $\pi^* f$ by sending (x, b) to $(g(x), \overline{\eta(x)}_! b)$. □

(Again, this lemma holds in any fibration category.)

One may also define cofibrancy, for objects of any fibration category:

Definition 2.2.13. An object C of a fibration category $\mathbf{C}$ is *cofibrant* if for any acyclic fibration $p \colon B \longrightarrow A$ and map $f \colon C \longrightarrow A$, there is some lifting $\bar{f} \colon C \longrightarrow B$:

$$\begin{tikzcd}
 & B \arrow[d, "p"] \\
C \arrow[ur, dashed, "\bar{f}"] \arrow[r, "f"'] & A.
\end{tikzcd}$$

When $\mathbf{C}$ is a categorical model of $\mathcal{H}'$, we have:

Lemma 2.2.14. *Every object of* $\mathbf{C}$ *is cofibrant.*

Proof. Lemma 2.2.8 implies that every acyclic fibration $\pi_1 \colon \sum_{x:A} B(x) \longrightarrow A$ admits some section: take some family of contractions of the fibers $\mathsf{hfib}(\pi_1, x)$, and send $x : A$ to the image of the center of contraction $*_x : \mathsf{hfib}(\pi_1, x)$ under the equivalence $\mathsf{hfib}(\pi_1, x) \simeq B(x)$. Now, given f as above, take $\bar{f}$ to be the composite of f with this section. □

We conclude with a somewhat subtler question. Another condition often assumed for fibration categories is that for any $\mathbb{N}$-indexed sequence

$$A_0 \xleftarrow{f_0} A_1 \xleftarrow{f_1} A_2 \xleftarrow{f_2} \cdots ,$$

if each f_i is a fibration then the sequence has a limit, and moreover the projections from this limit are again fibrations.

This turns out not to be provable in the type theory—in particular, it fails in the syntactic category $\mathcal{C}(\mathcal{H}')$. However, appropriate internally-formulated versions of it do hold; this is analogous to the fact that an elementary topos may fail to be externally complete, while possessing all limits in the internal sense.

To see how it fails in $\mathcal{C}(\mathcal{H}')$, consider the sequence of projections

$$1 \xleftarrow{f_0} \mathsf{Nat} \xleftarrow{f_1} \mathsf{Nat}^2 \xleftarrow{f_2} \cdots$$

This sequence cannot have a limit, since such a limit would be a $\mathbb{N}$-fold product of copies of Nat, and as such would necessarily have uncountably many global elements, while $\mathcal{C}(\mathcal{H}')$ is countable.

However, an *internal* limit for the sequence exists, in the form of the object $\mathsf{Nat}^{\mathsf{Nat}}$ (working internally, it does not make sense to ask whether the projections are fibrations); and, in some models (e.g. the simplicial model) this object turns out to be interpreted as the external limit $\prod_{\mathbb{N}} \mathsf{Nat}$.

3. Limits and applications

3.1. **Pullbacks and equalizers.**

Before defining general limits over graphs, we start by investigating pullbacks; these serve both as a warmup and as a useful tool for subsequent material.

3.1.1. *The standard construction of a pullback.* We start by explicitly constructing the pullback of a cospan. The definitions and theorems in this section are found in `Pullbacks`.

Definition 3.1.1 (`pullback`)**.** Let $A \xrightarrow{f} C \xleftarrow{g} B$ be a cospan of types and functions. The *(standard) pullback* $\mathsf{Pb}(f,g)$ of this cospan is defined as:

$$\mathsf{Pb}(f,g) := \sum_{x:A,\, y:B} (fx \rightsquigarrow gy)$$

with the obvious maps:

$$\begin{tikzcd} \mathsf{Pb}(f,g) \arrow[r, "\pi_B"] \arrow[d, "\pi_A"'] & B \arrow[d, "g"] \\ A \arrow[r, "f"'] & C \end{tikzcd} .$$

(This definition may be recast to parallel a traditional construction of the homotopy pullback in fibration categories [Bro73, Lem 1.3]: first fibrantly replace f by p_f as in Lemma 2.2.11, obtaining

$$\begin{tikzcd} & \mathsf{P}f = \sum_{c:C} \mathsf{hfib}(f,c) = \sum_{c:C,a:A} (fa \rightsquigarrow c) \arrow[d, "p_f"] \\ A \arrow[r, "f"'] & C, \end{tikzcd}$$

and then secondly, take the strict pullback of $\mathsf{P}f$ along g as a fibration over C, obtaining $\sum_{b:B} \mathsf{hfib}(f, g(c)) = \sum_{b:B,a:A} (fa \rightsquigarrow gb)$, which is (strictly, externally) isomorphic to $\mathsf{Pb}(f,g)$ as defined above.)

Note that the pullback is symmetric (`pullback_symm`): there is an equivalence $\mathsf{Pb}(f,g) \simeq \mathsf{Pb}(g,f)$ commuting appropriately with the projections and canonical homotopies.

Moreover, the construction of the pullback should be functorial in (f,g). This requires a few extra definitions to state:

Definition 3.1.2 (`cospan_map`)**.** Given two cospans $A \xrightarrow{f} C \xleftarrow{g} B$ and $A' \xrightarrow{f'} C' \xleftarrow{g'} B'$, a *cospan map* h from (f,g) to (f',g') consists of maps h_A, h_B, h_C and homotopies h_f, h_g:

$$\begin{tikzcd}
 & B \arrow[dd, "g"'] \arrow[dr, "h_B"] & \\
 & & B' \arrow[dd, "g'"] \arrow[dl, Rightarrow, "h_g"] \\
A \arrow[r, "f"] \arrow[dr, "h_A"'] & C \arrow[dr, "h_C"] & \\
 & A' \arrow[r, "f'"'] \arrow[u, Rightarrow, "h_f"'] & C'
\end{tikzcd}$$

There is an identity map from any cospan to itself (`cospan_idmap`); also, there is an evident composition of cospan maps (`cospan_comp`).

Proposition 3.1.3 (`pullback_fmap`)**.** *A map of cospans $h\colon (f,g) \longrightarrow (f',g')$ induces a map of pullbacks $\mathsf{Pb}(f,g) \longrightarrow \mathsf{Pb}(f',g')$. Moreover, this preserves composition and identities.*

The most frequent application of this functoriality, in practice, is the invariance of pullbacks under equivalences — that, for instance, given a cospan $A \xrightarrow{f} C \xleftarrow{g} B$ and an equivalence $e\colon A' \simeq A$, there is an equivalence between the pullbacks $\mathsf{Pb}(f,g)$ and $\mathsf{Pb}(f \cdot e, g)$. This, and various other similar statements, are all easily obtained from the functoriality of Pb together with the lemma:

Lemma 3.1.4 (`cospan_equiv_inverse`)**.** *Suppose $h = (h_A, h_B, h_C, h_f, h_g)$ is a cospan map from (f,g) to (f',g'), and h_A, h_B, h_C are equivalences. Then there is a cospan map $h^{-1}\colon (f',g') \longrightarrow (f,g)$, inverse to h in that there are paths $(h \cdot h^{-1} \rightsquigarrow 1)$ and $(h^{-1} \cdot h \rightsquigarrow 1)$.*

An interesting technical point arises here: rather than proving this and other facts about cospan maps directly, we deduce them from the analogous facts about commutative squares (considered as maps between functions). These are developed in the file `CommutativeSquares`. Most immediately, this arrangement slightly simplifies the proofs in the present section, since one does not have to write each construction out separately for the left and right legs of the cospan. It also allows us to directly re-use the commutative squares material in Section 3.2, as the building blocks of the analogous facts about diagrams over general graphs.

3.1.2. *The universal property of pullbacks.* Above, we defined pullbacks by a specific construction. Alternatively, one can characterize them by a universal property. For the next few definitions, fix some cospan $A \xrightarrow{f} C \xleftarrow{g} B$.

Definition 3.1.5 (`cospan_cone`)**.** Let X be any type. A *cone* μ over (f, g) with vertex X consists of functions μ_A, μ_B, and a homotopy $\mu_C : f \cdot \mu_A \Rightarrow g \cdot \mu_B$:

$$\begin{tikzcd} X \arrow[rrd, "\mu_B"] \arrow[rdd, "\mu_A"'] & & \\ & & B \arrow[d, "g"] \\ & A \arrow[r, "f"] \arrow[ru, Rightarrow, "\mu_C"] & C \end{tikzcd}$$

Write $\mathsf{Cone}(X; f, g)$ for the type of cones over (f, g) with vertex X.

$\mathsf{Cone}(X; f, g)$ should be contravariantly functorial in X. We do not show this in full; but in particular, a map $f\colon X' \longrightarrow X$ induces a map

$$\mathsf{Cone}(X; f, g) \longrightarrow \mathsf{Cone}(X'; f, g),$$

given by precomposing the components of the cone with f. For a cone μ, we denote this as $\mu \circ f$. Fixing a cone $\mu : \mathsf{Cone}(X; f, g)$ thus induces for any type X' a map

$$(\mu \circ -)\colon (X' \to X) \longrightarrow \mathsf{Cone}(X'; f, g).$$

This allows us to define the universal property of pullbacks:

Definition 3.1.6 (`is_pullback_cone`)**.** A cone μ over (f, g), with vertex P, is an *(abstract) pullback* for (f, g) if for every small type $X : \mathsf{U}$, the map $(\mu \circ -)$ gives an equivalence $(X \to P) \simeq \mathsf{Cone}(X; f, g)$.

One can of course ask whether $(\mu \circ -)$ is an equivalence for an arbitrary type X, not necessarily small; but to quantify over types, one must restrict to some universe. Even doing so, the resulting property of "being a pullback" is (a priori) as large as the universe used. It is, however, a mere proposition, since being an equivalence is one.

(For an investigation of left universal properties of inductive types, defined along similar lines, see [AGS12].)

Proposition 3.1.7 (`pullback_universal`)**.** *The evident cone from the standard pullback* $\mathsf{Pb}(f, g)$ *(3.1.1) to* (f, g) *is an abstract pullback.*

Proof. By direct construction: any cone from some X to (f, g) induces a map $X \longrightarrow \mathsf{Pb}(f, g)$, and by functional extensionality, this construction is inverse to composition with the standard cone. □

Proposition 3.1.8 (`abstract_pullback_unique`)**.** *If* $\mu : \mathsf{Cone}(X; f, g)$ *and* $\nu : \mathsf{Cone}(Y; f, g)$ *are both pullbacks for* (f, g)*, then the unique map* $f\colon Y \longrightarrow X$ *such that* $(\mu \circ f \rightsquigarrow \nu)$ *(provided by the universal property of* μ*) is an equivalence.*

Conversely, if $\mu : \mathsf{Cone}(X; f, g)$ *is any cone, and* $f\colon X \simeq Y$ *an equivalence, then setting* $\nu := \mu \circ f$*,* μ *is a pullback if and only if* ν *is.*

Proof. The following diagram commutes, and the maps $X \longrightarrow (1 \to X)$, $Y \longrightarrow (1 \to X)$ are equivalences:

$$\begin{tikzcd}
Y \arrow[r] \arrow[d, "f"'] & (1 \to Y) \arrow[dr, bend left, "(\nu \circ -)"] & \\
& & \mathsf{Cone}(1; f, g) \\
X \arrow[r] & (1 \to X) \arrow[ur, bend right, "(\mu \circ -)"'] &
\end{tikzcd}$$

It follows by 2-out-of-3 that if any two of f, $(\mu \circ -)$, $(\nu \circ -)$ are equivalences, so is the third. □

Corollary 3.1.9 (`is_pullback_cone'`). *A cone $\mu : \mathsf{Cone}(X; f, g)$ is a pullback cone if and only if the induced map $X \longrightarrow \mathsf{Pb}(f, g)$ is an equivalence.*

Since any two interderivable propositions are necessarily equivalent, this property could be used as an alternative definition of μ being a pullback cone, with the advantage (compared to our previous definition) of yielding again a small type, since it does not quantify over the universe.

3.1.3. *Two pullbacks lemmas.* We have introduced pullbacks in two different ways: via a concrete construction, and via a universal property. For each of these, one can give a version of the classical two pullbacks lemma.

Proposition 3.1.10 (`two_pullbacks_equiv`). *For all f, g, h as in the diagram below, the induced comparison map $\mathsf{Pb}(g^*f, h) \longrightarrow \mathsf{Pb}(f, g \cdot h)$ is an equivalence:*

$$\begin{tikzcd}
\mathsf{Pb}(f, g \cdot h) \arrow[dr] \arrow[drrr, bend left] \arrow[ddr, bend right] & & & \\
& \mathsf{Pb}(g^*f, h) \arrow[r] \arrow[d] & \mathsf{Pb}(f, g) \arrow[r] \arrow[d, "g^*f"] & A \arrow[d, "f"] \\
& B_2 \arrow[r, "h"] & B_1 \arrow[r, "g"] & C
\end{tikzcd}$$

Proposition 3.1.11 (`abstract_two_pullbacks_lemma`, in `Pullbacks3`). *Suppose that in a rectangle*

$$\begin{tikzcd}
P_2 \arrow[r] \arrow[d] & P_1 \arrow[r] \arrow[d, "k"] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\
B_2 \arrow[r, "h"] & B_1 \arrow[r, "g"] & C
\end{tikzcd}$$

the right square is a pullback. Then the left square is a pullback if and only if the outer rectangle is a pullback.

Proof. Write μ for the cone from P_1 to (f, g), μ' for the cone from P_2 to (g^*f, h), and μ'' for the cone from P_2 to $(f, g \cdot h)$. Then for any X, the following triangle commutes:

$$\begin{tikzcd} & \mathsf{Cone}(X; g^* f, h) \arrow[dd] \\ (X \to P_2) \arrow[ur, "(\mu' \cdot -)"] \arrow[dr, "(\mu'' \circ -)"'] & \\ & \mathsf{Cone}(X; f, g \cdot h) \end{tikzcd}$$

Here the vertical map denotes the composition of a cone on $(g^* f, h)$ with μ; and this can be shown (by direct construction) to be an equivalence. Hence by 2-out-of-3, $(\mu' \circ -)$ is an equivalence if and only if $(\mu'' \circ -)$ is. □

It should be noted that the arguments involved in showing the equivalence $\mathsf{Cone}(X; g^* f, h) \simeq \mathsf{Cone}(X; f, g \cdot h)$ are necessarily more involved than in the 1-categorical setting, since they depend on comparing paths in types; in terms of the classical theory, this is more analogous to the corresponding lemma for quasi-pullbacks in a bicategory.

3.1.4. *Equalizers.* The formal definitions and theorems corresponding to the remainder of Section 3.1 are found in the file `Pullbacks2`, except for the next definition, which appears in `Equalizers`.

Definition 3.1.12 (`equalizer`)**.** Let $f, g \colon A \longrightarrow B$. The *equalizer* of f and g is defined as the type:

$$\mathsf{Eq}(f, g) := \sum_{x:A} (fx \rightsquigarrow gx).$$

together with the projection $\pi \colon \mathsf{Eq}(f, g) \longrightarrow A$.

As in classical category theory, pullbacks and equalizers can be defined in terms of each other.

Proposition 3.1.13 (`eq_as_pb_equiv`)**.** *The equalizer of any parallel pair $f, g \colon A \longrightarrow B$ is equivalent to the pullback of the paired map $\langle f, g\rangle \colon A \longrightarrow B \times B$ and the diagonal Δ_B:*

$$\begin{tikzcd} \mathsf{Eq}(f, g) \simeq \mathsf{Pb}(\Delta_B, \langle f, g\rangle) \arrow[r] \arrow[d] & A \arrow[d, "\langle f,g\rangle"] \\ B \arrow[r, "\Delta_B"'] & B \times B \end{tikzcd}$$

Conversely, the pullback of any cospan $A \xrightarrow{f} C \xleftarrow{g} B$ *is equivalent to the equalizer of the pair*

$$f \cdot \pi_1, g \cdot \pi_2 \colon A \times B \longrightarrow C.$$

3.1.5. *Homotopy fibers and loop spaces.* We next consider a couple of examples which bring out the homotopical character of the theory—examples which in classical 1-category theory, and in the type theory with UIP[3], would be trivial, but which in the un-truncated type theory become non-trivial, corresponding to the classical theory of *homotopy* pullbacks.

We first need one piece of notation. Given a type B and an element $b : B$, write $\ulcorner b \urcorner \colon 1 \longrightarrow B$ for the map sending the unique element $* : 1$ to b.

Example 3.1.14 (`hfiber_to_pullback_equiv`)**.** Given a map $f \colon A \longrightarrow B$ and an element $b : B$, the homotopy fiber of f over b may equivalently be given as a pullback:

$$\begin{tikzcd}
\mathsf{hfib}(f,b) \simeq \mathsf{Pb}(\ulcorner b \urcorner, f) \arrow[r] \arrow[d] & A \arrow[d, "f"] \\
1 \arrow[r, "\ulcorner b \urcorner"'] & B
\end{tikzcd}$$

Example 3.1.15 (`Omega_to_pullback_equiv`)**.** Given a type B and an element $b : B$, the space of loops in B based at b, $\Omega(B,b) := (b \rightsquigarrow_B b)$ may be given as a pullback:

$$\begin{tikzcd}
\Omega(B,b) \simeq \mathsf{Pb}(\ulcorner b \urcorner, \ulcorner b \urcorner) \arrow[r] \arrow[d] & 1 \arrow[d, "\ulcorner b \urcorner"] \\
1 \arrow[r, "\ulcorner b \urcorner"'] & B.
\end{tikzcd}$$

This last example in particular exemplifies the fact that our pullbacks correspond, in the classical setting, to *homotopy* pullbacks.

3.1.6. *Properties of pullbacks.* Various nice properties of maps are preserved under pullback. In proving such preservation properties, the following lemma is rather useful:

Proposition 3.1.16 (`hfiber_of_pullback`)**.** *Given* $A \xrightarrow{f} C \xleftarrow{g} B$ *, the homotopy fiber of the map* f^*g *over a point* $a : A$ *is equivalent to the homotopy fiber of* g *over* $f(a)$*.*

Proof. Immediate from Example 3.1.14 together with the concrete two pullbacks lemma, Proposition 3.1.10. □

Corollary 3.1.17 (`pullback_preserves_equiv`)**.** *Equivalences are stable under pullback. That is, if* $g \colon B \longrightarrow C$ *is an equivalence, then for any* $f \colon A \longrightarrow B$*, the pullback* $f^*g \colon A \times_B C \longrightarrow A$ *is again an equivalence.*

Proof. Each fiber of f^*g is equivalent to some fiber of g, so is contractible. □

[3]"Uniqueness of Identity Proofs": the axiom that every identity type $(x \rightsquigarrow_X y)$ is a mere proposition [Str91], [War08].

More generally, any property of maps defined or characterized fiberwise, using an equivalence-invariant property of types, is itself stable under pullback (`pullback_preserves_fiberwise_properties`).

3.2. **Limits.** Generalizing the constructions above of pullbacks and equalizers, we move to limits for diagrams over arbitrary graphs. Unless otherwise noted, the formal definitions and theorems that follow are found in `Limits`.

3.2.1. *Graphs and diagrams.*

Definition 3.2.1 (`graph`). A *graph* G consists of:

- a type G_0 (the *vertices* or *objects* of G); and
- for each $i, j : G_0$, a type $G_1(i, j)$ (the *edges* or *arrows* from i to j).[4]

Definition 3.2.2 (`diagram`). A *diagram* D on a graph G consists of:

- for each vertex $i \colon G_0$, a type $D_0(i)$;
- for each arrow $g : G_1(i, j)$ of G, a map $D_1(g) \colon D(i) \longrightarrow D(j)$.

For both graphs and diagrams, we will often suppress the subscripts when they are clear from context.

Example 3.2.3 (`cospan_graph`, in `Limits2`). To recover cospans as an example of these diagrams, one can define a graph by taking G_0 to be the type with three elements, $\{l, m, r\}$ and let G_1 be given by:

- $G(l, m) := 1$,
- $G(r, m) := 1$,
- $G(i, j) := \emptyset$ otherwise.

A diagram D over this graph corresponds precisely to a cospan:

$$\begin{tikzcd} & D(r) \arrow[d] \\ D(l) \arrow[r] & D(m) \end{tikzcd}$$

3.2.2. *The universal property of limits.*

Definition 3.2.4 (`graph_cone`). Given a diagram D on a graph G, a *cone* μ on D with vertex X consists of:

- for each $i : G_0$, a map $\mu^0_i \colon X \longrightarrow D_0(i)$;
- for each arrow $g : G_1(i, j)$, a homotopy $\mu^1_g \colon D_1(g) \cdot \mu^0_i \Rightarrow \mu^0_j$.

Write $\mathsf{Cone}(X; D)$ for the type of cones on D with vertex X.

Again, we usually suppress the subscripts, writing just μ_i, μ_f.

As with cones over cospans, $\mathsf{Cone}(X; D)$ is functorial in X: a function $f \colon X' \longrightarrow X$ and a cone $\mu : \mathsf{Cone}(X; D)$ may be composed to give a cone $\mu \circ f : \mathsf{Cone}(X; D)$. This lets us generalize the definition of the universal property:

[4]Note that we do not assume truncatedness for any of the types involved; we do not need to, essentially since the definiton doesn't posit any paths within them. Cf. Section 3.2.4.

Definition 3.2.5 (`is_limit_cone`)**.** Let D be a diagram on the graph G. A cone μ over D, with vertex L, is an *(abstract) limit* for D if for every small type $X : \mathsf{U}$, the map $(\mu \circ -) \colon (X \to L) \longrightarrow \mathsf{Cone}(X; D)$ is an equivalence.

By abuse of notation, we often speak of L being the limit of D, when the cone μ is implicit.

Most of the theorems of the preceding section generalize immediately. In particular,

Proposition 3.2.6. *Given any two limit cones for the same diagram, the canonical map between their vertices is an equivalence; conversely, the composition of any limit cone with an equivalence is again a limit cone.*

Again as in the previous section, there is a standard construction of the limit:

Definition 3.2.7 (`limit`)**.** Let D be a diagram over a graph G. The *(standard) limit* $\mathsf{Lim}D$ is the type of pairs (x, α), where

- $x : \prod_{i:G_0} D(i)$;
- $\alpha : \prod_{i,j:G_0,\, g:G(i,j)} ((D(g)(x_i) \rightsquigarrow x_j))$.

There is an evident cone from $\mathsf{Lim}D$ to D (`limit_graph_cone`), and as one would hope,

Proposition 3.2.8 (`limit_universal`)**.** $\mathsf{Lim}(D)$ *is an abstract limit for* D.

Proposition 3.2.9 (`is_limit_cone'`)**.** *A cone μ from X to some diagram D is a limit for D if and only if the map $X \longrightarrow \mathsf{Lim}D$ induced by μ is an equivalence.*

One again, we may define maps of diagrams (`diagram_map`), and show that Lim is functorial in such maps, and in particular, is functorial in equivalences (`limit_fmap_equiv`). Since graphs, diagrams, and limits are all simply built up from arrows, these definitions and results follow straightforwardly once one has given the basic case of commutative squares, seen as maps between functions. (This is handled in the file `CommutativeSquares`.)

3.2.3. *Examples and properties.*

Example 3.2.10 (`pb_as_lim_equiv`, in `Limits2`)**.** In Example 3.2.3 above, we saw that cospans correspond to diagrams over a certain graph. Then cones over these diagrams correspond to cones over the cospans, as originally defined; and a diagram-cone is a limit exactly if the corresponding cospan-cone is a pullback.

Example 3.2.11 (`lim_as_eq`)**.** Just as in the classical 1-categorical theory, the limit over a diagram D may be constructed as an equalizer of maps between products:

$$\prod_{i,j:G,\, g:G(i,j)} D(i) \rightrightarrows \prod_{i:G} D(i)$$

Various useful facts are also straightforward to deduce from the standard construction; for instance,

Proposition 3.2.12 (`trunc_limits_preserve_trunc`, in `Limits2`)**.** *If D is a diagram on some graph, and each type $D(i)$ is an n-type, then $\mathsf{Lim}D$ is an n-type; hence via the canonical equivalence, so is any other limit for D.*

3.2.4. *Why not categories?* One might reasonably ask here: why have we considered limits only over graphs, not over categories as is usual in the 1-categorical theory?

The problem—as ever in homotopical settings—is one of *coherence*. Defining a category internally is roughly analogous to defining an $(\infty, 1)$-category externally; that is, it requires not only identity, composition, associativity, and the like, but also higher-dimensional data ensuring the coherence of the paths witnessing the associativity axioms, and so on in arbitrarily high dimensions. While we hope that this will eventually be possible in the type theory, it is currently far from clear how to present it.

In defining categories, this problem can be avoided by assuming truncatedness of the types of morphisms; see [AKS13] for a development of the resulting theory. However, to talk about diagrams of arbitrary types over such categories would once again require an infinite family of coherence conditions, essentially since one is presenting an ∞-functor into the $(\infty, 1)$-category of all types, which is not generally n-truncated for any n.

However, working with graphs avoids these issues entirely: a map out of a graph (or equivalently, out of the free category thereon) consists purely of 0- and 1-dimensional data, with no coherence required. (More generally, one could use a similar approach to describe diagrams over finite-dimensional computads or semi-simplicial objects without confronting coherence issues.)

3.3. **Pointed types and fiber sequences.**

3.3.1. *Definitions.* The formal definitions and theorems described in this section are found in `PointedTypes`.

Definition 3.3.1 (`pointed_type`)**.** A *pointed type* (A, a_0) is a type A, together with an element $a_0 : A$, the *basepoint*. (We will often refer to both the pointed type and its underlying type as A, and write $\mathsf{pt}(A)$ for the basepoint.)

Definition 3.3.2 (`pointed_map`)**.** A *map of pointed types* (or *pointed map*) $(f, p)\colon (A, a_0) \longrightarrow (B, b_0)$ consists of a function $f\colon A \longrightarrow B$, together with a path $p : (f(a_0) \rightsquigarrow_B b_0)$. (Again, we will often write f for the whole pointed map, and $\mathsf{pt}(f)$ for its associated path.)

The loop space construction Ω lifts naturally to a map from pointed types to pointed types, setting $\Omega A := ((\mathsf{pt}\, A \rightsquigarrow \mathsf{pt}\, A), \mathsf{refl}(\mathsf{pt}\, A))$. One can therefore iterate it, giving the n-fold loop spaces $\Omega^n A$ of a pointed type. Moreover, this has an associated action on maps. A pointed map $f\colon A \longrightarrow B$

induces a pointed map $\Omega(f)\colon \Omega A \longrightarrow \Omega B$, with underlying map sending $q : (\mathsf{pt}\,A \rightsquigarrow \mathsf{pt}\,A)$ to $\overline{\mathsf{pt}\,f} \centerdot f[q] \centerdot \mathsf{pt}\,f : (\mathsf{pt}\,B \rightsquigarrow \mathsf{pt}\,B)$.

Similarly, the homotopy fiber construction hfib lifts naturally to the pointed world. Given a pointed map $f\colon A \longrightarrow B$, write $\mathsf{hfib}(f)$ for the pointed type given by $\mathsf{hfib}(f, \mathsf{pt}\,B)$, with basepoint $(\mathsf{pt}\,A, \mathsf{pt}\,f)$; and the inclusion $\mathsf{hfib}(f) \longrightarrow A$ is again a pointed map.

3.3.2. *The long exact sequence of a pointed map.* As an application of the above tools, we can now recover the long exact sequence associated to a pointed map. This sequence is a basic but powerful computational tool in classical homotopy theory, and promises to be so also in homotopy type theory: [Uni13, 8.5], for instance, gives a type-theoretic version of the classical proof that $\pi_3(S^2) \cong \mathbb{Z}$, using the long exact sequence of the Hopf fibration. Similarly, one can straightforwardly reconstruct the classical theory of covering spaces, as families of *sets* varying over a type, and conclude that they induce isomorphisms of higher homotopy groups.

Definition 3.3.3 (`hfiber_ptd`)**.** A *fiber sequence* consists of a pair $F \xrightarrow{g} E \xrightarrow{f} B$ of pointed maps, together with an equivalence $F \simeq \mathsf{hfib}(f)$ commuting with the inclusion $\mathsf{hfib}(f) \longrightarrow E$.

Note that up to canonical equivalence, a fiber sequence is determined simply by the single pointed map $E \longrightarrow B$.

The following theorem is found in `LongExactSequences`.

Theorem 3.3.4 (`hfiber_sequence`, `Omega_to_hfiber_seq_0`, et seq.)**.** *Given a pointed map $f\colon E \longrightarrow B$, there is a sequence of maps*

$$\ldots \longrightarrow \Omega^2 B \longrightarrow \Omega F \longrightarrow \Omega E \longrightarrow \Omega B \longrightarrow F \longrightarrow E \longrightarrow B$$

in which every pair of consecutive maps forms a fiber sequence.

Proof. Taking $F := \mathsf{hfib}(f)$, it is sufficient to prove that the homotopy fiber of the inclusion $F \longrightarrow E$ is pointed-equivalent to ΩB; subsequent stages follow by iteration. One can prove this equivalence by direct construction; alternatively, the results of Section 3.1 allow us to give a rather more conceptual proof, due originally to Mather [Mat76, Lem. 32]:

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d] & F \arrow[r] \arrow[d] & 1 \arrow[d, "\ulcorner \mathsf{pt}\,B \urcorner"] \\
1 \arrow[r, "\ulcorner \mathsf{pt}\,E \urcorner"] & E \arrow[r, "f"] & B
\end{tikzcd}$$

By the two pullbacks lemma, the pullback of the left-hand square is equivalent to the pullback of the whole rectangle. But by Examples 3.1.14 and 3.1.15, these pullbacks are respectively equivalent to the homotopy fiber of $F \longrightarrow E$, and to the loop space ΩB. □

4. Reflections on the formal verification

Formalizing the constructions of Sections 2 and 3 was often straightforward: many of the definitions are very naturally expressed in the language of type theory, and verifying their properties is often just a matter of unpacking definitions and applying straightforward logical manipulations and background facts. Sometimes, however, additional effort was required. In this section, we survey some of the practical lessons learned during the formalization.

4.1. **Limitations.**

One fundamental challenge that arises comes from working purely in the type theory. In classical approaches to homotopy theory, one always has an extra external scaffolding available, with (in particular) strict, on-the-nose equality on all types of objects. One typically expects the main results and constructions to respect appropriate notions of equivalence, but one is free to use intermediate constructions that do not.

Developing the homotopy theory in HoTT, we are constrained to work entirely in a homotopy-invariant manner, rendering some classical techniques unavailable. In most cases, some fully invariant approach is reasonably apparent; but sometimes, one is not. We saw such a case in Section 3.2: we do not know how to represent the notion of a diagram over an arbitrary category, and so restricted attention to (diagrams and limits over) graphs.

4.2. **Proof-relevance.**

Another difficulty lies in getting used to thinking of proofs of equalities as *constructions* that one might need to prove things about later on.

In traditional formalizations, equality is *proof-irrelevant*: different proofs of the same equality are not logically distinguishable. In Coq, for instance, one could safely end them with the keyword `Qed`, which renders them *opaque*, meaning that one cannot later access their contents. In traditional mathematics, this makes sense; once one has an equality, one only needs the fact that it holds, treating the proof as a black box.

In HoTT, however, equality is proof-relevant: a path type may have multiple logically distinct inhabitants. When constructing equality proofs in this setting, one typically needs to end an equality proof with the keyword `Defined`, allowing the user to unfold that definition later on. The specifics of the proof matter; one tries to keep proofs as clean and short as possible, using lemmas and constructions with known, previously proven properties. Unfortunately, this means that several of Coq's powerful tactics (notably the `rewrite` family) are somewhat unsatisfactory in our setting: the paths they produce are difficult to reason about later.

On the other hand, some important statements remain proof-irrelevant. If a type has been shown to be a proposition, one knows that any two elements of it are canonically equal; so one may make such an element opaque without losing any logical content. Even so, it is often convenient to leave such objects transparent, to retain their *computational* content.

For instance, for a function `f`, the type `IsEquiv f` (the property that `f` is an equivalence) is a proposition; so in principle one may safely render a proof of this opaque. However, one often uses such a proof to produce an inverse for `f`; if the proof was transparent, then the resulting inverse will retain computational properties from its construction, whereas if the proof is opaque, one must reason explicitly about the action of the inverse. We formed no clear convention on this: sometimes it turned out more convenient to keep such proofs transparent, for easier reduction in later proofs; in other case, this was unnecessary, and making the proofs opaque gave more efficient compilation.

4.3. **Constructing paths.** The most fundamental type constructor in homotopy type theory is the type of paths, and the most challenging parts of proofs usually involved constructing paths between complex objects. Given the subject matter, we never had to pass beyond the 2-categorical level, constructing paths between paths; but even so, this required a good deal of care, and facility with path algebra.

One recurring situation was the construction of paths between elements of a dependent sum, or elements of a record type with dependencies between components. For example, if (a, b) and (a', b') are elements of a type $\sum_{x:A} B(x)$, constructing a path between these two elements involves constructing a path p from a to a', and then constructing a path q from the transport of b along p to b'. Thus in general we have:

```
Lemma total_paths {A : Type} {B : A -> Type}
  {s s' : total B}
  (p : paths (pr1 s) (pr1 s'))
  (q : paths (p # (pr2 s)) (pr2 s'))
: s = s'.
```

where `pr1` and `pr2` denote the projections from the total space $\sum_{x:A} B(x)$. For interactive, tactic-based proofs, we generally found it useful to bundle the arguments p, q into a single structure:

```
Lemma total_paths' {A : Type} {B : X -> Type}
  {s s' : total B}
: { p : pr1 s = pr1 s' & p # pr2 s = pr2 s' } -> s = s'.
```

Recall that here `{ p : pr1 s = pr1 s' & p # pr2 s = pr2 s' }` is notation for a dependent sum, denoting the type of pairs (p, q) as above. When constructing a path between elements of a dependent sum, even when p is explicitly available, applying `(total_paths p)` sometimes fails to infer implicit arguments. Instead, applying `total_paths'` leaves the goal of providing the pair (p, q), providing the user explicitly with their required types. The tactic `exists p` can then be used to give the first component, leaving the goal of constructing the second path q interactively.

The problem is that `transport` is rather difficult to work with. There are many library lemmas about how its behaviour depends on the dependent type B, which in principle allow one to work with transported terms; but we found it more convenient to directly give tailored variants of `total_paths` for each specific Σ- and record type.

For example, taking a cospan $f\colon A \longrightarrow C$, $g\colon B \longrightarrow C$, the standard pullback of f and g is given by the type $\sum_{x:A,y:B}(fx \rightsquigarrow gy)$. Using `total_path` to provide a path in this type between triples `(x;(y;p))`, `(x';(y';p'))` would require three paths `q : x = x'`, `r : q # y = y'`, and `s : r # q # p = p'`. Notice, however, that in this case the second component, y, does not depend on x, so the transport is trivial; and moreover, the doubly-transported third component can be explicitly described as a composite. Thus, one can provide the following lemma to construct a path between two elements of the standard pullback:

```
Definition pullback_path' {A B C : Type} {f : A -> C} {g :
    B -> C}
  (u u' : pullback f g)
: { p : pullback_pr1 u = pullback_pr1 u'
    & {q : pullback_pr2 u = pullback_pr2 u'
    & (ap f p)^ @ (pullback_comm u) @ (ap g q)
      = pullback_comm u' } }
  -> u = u'.
```

The process of analyzing the canonical data for presenting a path between elements of a complex type, and writing lemmas to construct and work with such paths, was crucial to the formalization.

To consider one last example of this sort, recall that a cospan cone, that is, a diagram on the data f, g above, consists of a space, X, and maps h and k from X to A and B, respectively, making the diagram commute.

```
Definition cospan_cone {A B C : Type} (f : A -> C)
  (g : B -> C) (X : Type)
:= { h : (X -> A) &
     { k : (X -> B) & forall x, (f(h x)) = (g(k x)) }}.
```

A path between two such cones involves, in particular, a path between the family of paths in the third component:

```
Definition cospan_cone_path
  {A B C : Type} {f : A -> C} {g : B -> C} {X : Type}
  {Phi1 Phi2 : cospan_cone f g X}
  (p : cospan_cone_map1 Phi1 = cospan_cone_map1 Phi2)
  (q : cospan_cone_map2 Phi1 = cospan_cone_map2 Phi2)
  (r : forall x:X,
    cospan_cone_comm Phi1 x = (ap f (ap10 p x)) @
        cospan_cone_comm Phi2 x @ (ap g (ap10 q x))^)
  : Phi1 = Phi2.
```

Here, `cospan_cone_map1`, `cospan_cone_map2`, and `cospan_cone_comm` refer to the three components of a cospan cone in the preceding definition. As with `total_paths`, we also give a version `cospan_cone_path'` that packages the required components into a dependent sum, and is often more convenient in interactive proofs.

The advantage to these formulations is that it is comparatively straightforward (using lemmas from the HoTT library) to reason about transport operations their interactions with each other, as well as with path operations such as concatenation and inversion.

Returning to the question of the path-algebra itself, we found the formalization to require significant facility with such calculations. The HoTT library has a number of tactics for automating common manipulations and simplifications, but we found these tactics generally slowed down the proof-checker significantly. So, for the most part, we ended up giving such calculations by hand, building them explicitly from basic lemmas.

4.4. **General strategies.** We found it important to develop our theories and proofs in a modular way. The value of modularity in interactive theorem proving is well understood (see, for example, [GAA+13]), but in the context of homotopy type theory, it takes on additional significance. For one thing, many statements involving paths can only be proved when stated in full generality (to make available the elimination for Id-types). As a consequence, some facts cannot be derived in the course of a proof, on the fly, but have to be expressed independently. The fact that one often needs to reason about the construction of paths provides an additional reason to construct such proofs out of individually-named component lemmas: doing so allows one derive properties of the components individually, and then invoke these properties later on. In other words, reasoning about a modularly-constructed proof allows one to work with the individual lemmas and unpack their contents selectively, as needed. In contrast, the failure to modularize can result in formal terms that are overwhelming in complexity.

Perhaps the most important lesson we learned was not to expect too much from an interactive theorem prover. Although homotopy type theory provides a powerful framework to support homotopy-theoretic reasoning, one still needs a thorough understanding of the relevant mathematics. To get some of the more complex proofs and constructions to work, we found it vitally important to find the right definitions, the right way of formulating assertions, the right supporting infrastructure, and the right proof strategies. This required thinking carefully about the mathematical content, avoiding the temptation to simply dive in and hack.

This should not suggest that Coq was no help at all. Indeed, Coq was excellent for helping us keep track of definitions and formulate statements correctly. Especially for more complex path-constructions, applying standard rules to unwrap and reduce the contents of a goal type was an extremely useful aid to finding the term required. In practice, we found ourselves going back and forth between the blackboard and Coq, using Coq to negotiate the inevitable syntactic bureaucracy, and then returning to the blackboard to recoup intuitions and plan proof-strategies. In this way, Coq earned its keep, serving as a "proof assistant" in a very real sense.

4.5. **A case study: the two pullbacks lemma.** We close with a discussion of the abstract two pullbacks lemma, Proposition 3.1.11, by way of illustration. Somewhat to our surprise, this turned out to be the most difficult proof in our formalization. In the end, we tried three substantially different approaches before finding one satisfactory. All three can be found in `Pullbacks3_alt`.

Consider for now just the forward direction of Proposition 3.1.11, which states that if both squares have the universal property of pullbacks, then so does the composite. Let $f\colon A \longrightarrow C$, $g\colon B_1 \longrightarrow C$, $h\colon B_2 \longrightarrow C$, and $k\colon P_1 \longrightarrow B_1$ denote the maps so labeled in the diagram there. Our first approach invoked the concrete two pullbacks lemma, Proposition 3.1.10, which states that

$$\mathsf{Pb}(g^*f, h) \simeq \mathsf{Pb}(f, g \cdot h).$$

We then derived the following chain of equivalences, using the fact that cones from X to the cospan (f, g) are equivalent to maps from X to the standard pullback:

$$\begin{aligned}
(X \to P_2) &\simeq \mathsf{Cone}(X; k, h) \\
&\simeq (X \to \mathsf{Pb}(k, h)) \\
&\simeq (X \to \mathsf{Pb}(g^*f, h)) \\
&\simeq (X \to \mathsf{Pb}(f, g \cdot h)) \\
&\simeq \mathsf{Cone}(X; f, g \cdot h).
\end{aligned}$$

Here the second and last equivalences are just the universal properties of the concrete pullbacks. The notation g^*f in the third equivalence denotes the pullback of f along g according to the concrete pullback construction; this equivalence relies on the fact that any abstract pullback is equivalent to the concrete one, and the fact that the concrete pullback construction is functorial. The fourth equivalence is just (post-composition with) the concrete two pullbacks equivalence, Proposition 3.1.10.

The equivalence of the left- and right-hand sides of the chain above *almost* gives what we want: however, the universal property for the outer pullback square requires not just that an equivalence exists, but that the *canonical* map from $X \longrightarrow P_2$ to $\mathsf{Cone}(X; f, g \cdot h)$ is an equivalence.

What remains is thus to show that the map we have just constructed is homotopic to the canonical one! This, however, turned out to be extremely difficult. The problem was a failure of modularity: all we could do was unwrap the long, complicated term, and calculate. We managed to do this, but although the tactic engine declared the effort successful, we were unable to get it past the type-checker (presumably because the resulting term was too large).

Our second approach involved constructing the desired inverse by hand. Any cone $\mu : \mathsf{Cone}(X; f, g \cdot h)$ over the outer cospan can be reinterpreted as a cone $\mu' : \mathsf{Cone}(X; f, g)$ over the right cospan. Applying the universal property of the cone from P_1, we obtain a map $m_1 \colon X \longrightarrow P_1$ inducing μ'; we can then take m_1 as the first leg of a cone $\mu'' : \mathsf{Cone}(X; k, h)$ on the left cospan. Applying the universal property of the cone from P_2 then gives a map $m_2 \colon X \longrightarrow P_2$, as desired. However, the task of proving that this construction is indeed a two-sided inverse for $(\mu \circ -)$ turned out to be difficult. For example, the first task requires one to show that, starting with a cone $\mu : \mathsf{Cone}(X; f, g \cdot h)$, carrying out the procedure above to obtain a map from X to P_2 and then taking the induced cone, the resulting cone $\nu : \mathsf{Cone}(X; f, g)$ is connected by a path to the original μ. As described in Section 4.3, this involves showing not only that the component maps agree, but also that the resulting families of equality proofs agree as well; this turns out to be an interesting but laborious exercise in bicategorical path-algebra.

We finally settled on the approach described in Section 3.1.3, which establishes both directions of Proposition 3.1.11 simultaneously. Showing that the type $\mathsf{Cone}(X; k, h)$ of cones on the left cospan is equivalent to the type $\mathsf{Cone}(X; f, g \cdot h)$ of cones on the outer cospan required some effort, but the result was still considerably cleaner than either of the previous proofs. With that in hand, all that remained was to show that the triangle depicted in the proof of Proposition 3.1.11 in Section 3.1.3 commutes. To our very pleasant surprise, this fact had a one-line proof in Coq:

```
Lemma two_pullback_triangle_commutes {P1 : Type}
    (C1 : cospan_cone f g P1)
    {P2 : Type} (C2 : cospan_cone (cospan_cone_map2 C1) h P2)
    {X : Type} (m : X -> P2)
: top_cospan_cone_to_composite C1 (map_to_cospan_cone C2 X
    m)
= map_to_cospan_cone (top_cospan_cone_to_composite C1 C2) X
    m.
Proof.
  exact 1.
Defined.
```

In other words, the left- and right-hand sides are definitionally equal.

References

[AGS12] Steve Awodey, Nicola Gambino, and Kristina Sojakova, *Inductive types in homotopy type theory*, 2012 27th Annual IEEE Symposium on Logic in Computer Science (LICS), IEEE, 2012, pp. 95–104.

[AKS13] Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman, *Univalent categories and Rezk completion*, submitted, 2013.

[AW09] Steve Awodey and Michael A. Warren, *Homotopy theoretic models of identity types*, Math. Proc. Cambridge Philos. Soc. **146** (2009), no. 1, 45–55, arXiv:0709.0248, doi:10.1017/S0305004108001783.

[Bau89] Hans Joachim Baues, *Algebraic homotopy*, Cambridge Studies in Advanced Mathematics, vol. 15, Cambridge University Press, Cambridge, 1989, doi:10.1017/CBO9780511662522.

[BK72] Aldridge K. Bousfield and Daniel M. Kan, *Homotopy limits, completions and localizations*, Lecture Notes in Mathematics, Vol. 304, Springer-Verlag, Berlin, 1972.

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

[DHKS04] William G. Dwyer, Philip S. Hirschhorn, Daniel M. Kan, and Jeffrey H. Smith, *Homotopy limit functors on model categories and homotopical categories*, Mathematical Surveys and Monographs, vol. 113, American Mathematical Society, Providence, RI, 2004.

[GAA$^+$13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry, *A machine-checked proof of the odd order theorem*, Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings (Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, eds.), Lecture Notes in Computer Science, vol. 7998, Springer, 2013, pp. 163–179.

[GG08] Nicola Gambino and Richard Garner, *The identity type weak factorisation system*, Theoret. Comput. Sci. **409** (2008), no. 1, 94–109, arXiv:0803.4349, doi:10.1016/j.tcs.2008.08.030.

[HoT] HoTT group, *Homotopy type theory repository*, ongoing Coq development, https://github.com/HoTT/HoTT.

[HS98] Martin Hofmann and Thomas Streicher, *The groupoid interpretation of type theory*, Twenty-five years of constructive type theory (Venice, 1995), Oxford Logic Guides, vol. 36, Oxford Univ. Press, New York, 1998, pp. 83–111.

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

[Lum09] Peter LeFanu Lumsdaine, *Weak ω-categories from intensional type theory (conference version)*, Typed lambda calculi and applications (Berlin), Lecture Notes in Computer Science, vol. 5608, Springer, 2009, pp. 172–187.

[Lum11] ———, *Model structures from higher inductive types*, unpublished note, December 2011, http://www.mathstat.dal.ca/~p.l.lumsdaine/research/Lumsdaine-Model-strux-from-HITs.pdf.

[Lur09] Jacob Lurie, *Higher topos theory*, Annals of Mathematics Studies, vol. 170, Princeton University Press, Princeton, NJ, 2009.

[Mat76] Michael Mather, *Pull-backs in homotopy theory*, Can. J. Math **28** (1976), no. 2, 225–263.

[ML84] Per Martin-Löf, *Intuitionistic type theory*, Studies in Proof Theory. Lecture Notes, vol. 1, Bibliopolis, Naples, 1984.

[PW12] Álvaro Pelayo and Michael Warren, *Homotopy type theory and Voevodsky's univalent foundations*, preprint, 2012, arXiv:1210.5658.

[RB06] Andrei Radulescu-Banu, *Cofibrations in homotopy theory*, preprint, 2006, arXiv:0610009.

[RS13] Egbert Rijke and Bas Spitters, *Limits and colimits in Homotopy Type Theory*, in preparation, 2013.

[Shu14] Michael Shulman, *The univalence axiom for inverse diagrams and homotopy canonicity*, Mathematical Structures in Computer Science (2014), to appear, arXiv:1203.3253.

[Str91] Thomas Streicher, *Semantics of type theory*, Progress in Theoretical Computer Science, Birkhäuser Boston Inc., Boston, MA, 1991, Correctness, completeness and independence results, With a foreword by Martin Wirsing.

[Uni13] The Univalent Foundations Program, *Homotopy type theory: Univalent foundations of mathematics*, Tech. report, Institute for Advanced Study, 2013.

[vdBG11] Benno van den Berg and Richard Garner, *Types are weak ω-groupoids*, Proc. Lond. Math. Soc. (3) **102** (2011), no. 2, 370–394, arXiv:0812.0298, doi:10.1112/plms/pdq026.

[vdBG12] ———, *Topological and simplicial models of identity types*, ACM Trans. Comput. Log. **13** (2012), no. 1, Art. 3, 44, arXiv:1007.4638v1, doi:10.1145/2071368.2071371.

[Voe] Vladimir Voevodsky, *Univalent foundations repository*, ongoing Coq development, https://github.com/vladimirias/Foundations.

[Voe06] ———, *A very short note on homotopy λ-calculus*, notes from seminars given at Stanford University, 2006, http://math.ucr.edu/home/baez/Voevodsky_note.ps.

[Voe10] ———, *Notes on type systems*, ongoing unpublished manuscript, 2010, http://www.math.ias.edu/~vladimir/Site3/Univalent_Foundations_files/expressions_current.pdf.

[War08] Michael A. Warren, *Homotopy theoretic aspects of constructive type theory*, Ph.D. thesis, Carnegie Mellon University, 2008.

(Jeremy Avigad) Carnegie Mellon University, Pittsburgh
E-mail address: avigad@cmu.edu

(Krzysztof Kapulkin) University of Pittsburgh
E-mail address: krk56@pitt.edu

(Peter LeFanu Lumsdaine) Institute for Advanced Study, Princeton
E-mail address: plumsdaine@ias.edu