View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

The Seifert–van Kampen Theorem in Homotopy Type Theory*

Kuen-Bang Hou (Favonia)[†1] **and Michael Shulman**[‡2]

1 **Department of Computer Science, Carnegie Mellon University, Pittsburgh, USA**
favonia@cs.cmu.edu

2 **Department of Mathematics and Computer Science, University of San Diego, USA**
shulman@sandiego.edu

Abstract

Homotopy type theory is a recent research area connecting type theory with homotopy theory by interpreting types as spaces. In particular, one can prove and mechanize type-theoretic analogues of homotopy-theoretic theorems, yielding "synthetic homotopy theory". Here we consider the Seifert–van Kampen theorem, which characterizes the loop structure of spaces obtained by gluing. This is useful in homotopy theory because many spaces are constructed by gluing, and the loop structure helps distinguish distinct spaces. The synthetic proof showcases many new characteristics of synthetic homotopy theory, such as the "encode-decode" method, enforced homotopy-invariance, and lack of underlying sets.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases homotopy type theory, fundamental group, homotopy pushout, mechanized reasoning

Digital Object Identifier 10.4230/LIPIcs.CSL.2016.22

1 Introduction

Homotopy type theory is an emerging research area which draws ideas from type theory, homotopy theory and category theory [18, 20, 2, 19, 10, 8, 7]. Most of the current research has focused on an extension of Martin-Löf type theory by Voevodsky's *univalence axiom* and *higher inductive types* [18], interpreting types as "spaces up to homotopy", and Martin-Löf's identification type as a space of "homotopical paths". One of the more intriguing applications of this theory is *synthetic homotopy theory*: proving and mechanizing type-theoretic versions of theorems in classical homotopy theory [18, 15, 12, 14, 13, 9]. Upon translation through the homotopy-theoretic semantics of type theory [10], these yield proofs of the corresponding classical results, sometimes involving significant new insights [17].

* We thank Ulrik Buchholtz, Floris van Doorn, Daniel R. Licata, and anonymous reviewers for their useful feedback on earlier drafts.
† Sponsored by the National Science Foundation under grant numbers CCF-1116703, W911-NF0910273 and W911-NF1310154; and Air Force Office of Scientific Research under grant numbers FA-95501210370 and FA-95501510053. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.
‡ Supported by the National Science Foundation under a postdoctoral fellowship and agreement No. 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.

© Kuen-Bang Hou (Favonia) and Michael Shulman;
licensed under Creative Commons License CC-BY

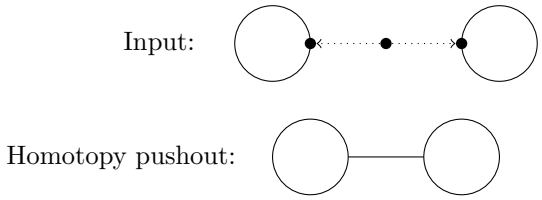

Figure 1 A homotopy pushout.

In this paper we advance the program of synthetic homotopy theory by proving and mechanizing the *Seifert–van Kampen theorem*, which computes the fundamental group of a homotopy pushout. The *fundamental group* $\pi_1(X)$ is a homotopy-theoretic invariant of a space X that measures intuitively "how many loops" it contains. For instance, the fundamental group of a circle is the integers $\mathbb{Z}$, because one can loop around the circle any number of times (in either direction); while the fundamental group of a torus (the surface of a donut) is $\mathbb{Z} \times \mathbb{Z}$, because one can either loop around the outer edge or through the hole (any number of times each).

A *homotopy pushout* is a way of gluing two spaces together to produce a new one, by specifying an inclusion of a common subspace that should be glued together. For instance, given two circles and a specified point on each, we can glue the two points together with a "bridge", producing a "barbell" shape; see Figure 1. Many of the spaces of interest to homotopy theory can be obtained by gluing together intervals, discs, and higher-dimensional discs (such gluings are called *cell complexes*); thus it is obviously of interest to calculate invariants of such gluings, such as the fundamental group.

The Seifert–van Kampen theorem tells us how to compute the fundamental group of a homotopy pushout; that is, it tells us how many loops there are in a glued space. In the example of Figure 1, a loop in the homotopy pushout can go around one circle any number of times (in either direction), then around the other circle any number of times, then back around the *first* circle some other number of times, and so on. More precisely, the fundamental group of the figure-eight or barbell is the *free product* of the fundamental groups of the two circles ($\mathbb{Z}$ and $\mathbb{Z}$), which is the coproduct in the category of groups. More generally, the Seifert–van Kampen theorem says that if we glue two spaces B and C together along a connected space A,[1] then the fundamental group $\pi_1(B \sqcup^A C)$ is the *amalgamated free product* (the pushout in the category of groups) of $\pi_1(B)$ and $\pi_1(C)$ over $\pi_1(A)$.

In fact, the full Seifert–van Kampen theorem applies also in the case when A is not connected. This requires replacing fundamental groups $\pi_1(X)$ by fundamental *groupoids* $\Pi_1 X$, which keep track of loops at different basepoints. If a space is not connected, such as the disjoint union of a circle with a point, then different numbers of loops may be possible depending on where we start. The fundamental groupoid actually records all *paths* between points, thus including loops starting at all points and also the information about which pairs of points are connected. The full SvKT says that the functor Π_1 takes homotopy pushouts to pushouts of groupoids.

Stating and proving this theorem in homotopy type theory is a bit subtle for several reasons. According to the homotopy-theoretic interpretation of type theory, types act like ∞-groupoids, and ordinary groupoids can be identified with the "1-truncated" types. Under

[1] The theorem is traditionally stated for a topological space X which is the union of two open subspaces U and V, but in homotopy-theoretic terms this is just a convenient way of ensuring that X is the homotopy pushout of U and V over $U \cap V$.

this interpretation, the Π_1 is represented by a type constructor called the 1-truncation, which is a left adjoint to the inclusion of 1-truncated types into all types. Since left adjoints preserve pushouts, the SvKT appears to follow trivially.

However, this form of SvKT is not actually particularly useful. We were originally interested in the fundamental *group* $\pi_1(X)$, which is a hom-set in the fundamental groupoid $\Pi_1 X$. If $\Pi_1 X$ is represented by a 1-truncated type, then its "hom-sets" are its path spaces (Martin-Löf's identity types), and so to find $\pi_1(X)$ when X is a pushout, we must compute the path space of a pushout. But computing the path space of a pushout is what the SvKT was supposed to do *for* us! So this version of the theorem has really just shifted the burden of calculation elsewhere.

To obtain a more calculationally useful version of SvKT, we represent groupoids more "analytically" with a type of objects and dependent types of morphisms. For $\Pi_1 X$, the type of objects is X itself, and the hom-set between $x, y : X$ is the 0-truncation (set of connected components) of the path-space from x to y. (The connection between the two fundamental groupoids is that in the language of [18, Chapter 9], the latter $\Pi_1 X$ is a "pregroupoid" whose "Rezk completion" [1] has the 1-truncation of X as its type of objects. Informally, this means that we force the type family $\Pi_1 X$ to coincide with the identity type.)

Now our goal is to calculate this truncated path space, given an expression of X as a homotopy pushout. For this we use the "encode-decode method" [15, 18]. The idea of this method is to define a type family code indexed by pairs of elements of X, using the recursion principle of X coming from its expression as a homotopy pushout; and then show (using the analogous induction principle of X, along with path induction) that this family of "codes for paths" is equivalent to the actual family of truncated path spaces.

We will do this in Section 3, after a brief review of homotopy type theory in Section 2. However, there is one further valuable refinement. The description of code in Section 3 is not maximally explicit in all cases, because it incorporates $\pi_1(X)$ through "homotopical magic". To rectify this, in Section 4 we prove an improved version of SvKT where X is equipped with an arbitrary type of "base points". Some example applications can be found at the ends of Section 3 and Section 4.

All the theorems of the paper have been mechanized in the proof assistant Agda [16]; the code can be found at `https://github.com/HoTT/HoTT-Agda/blob/1.0/Homotopy/VanKampen.agda`. We end in Section 5 with some remarks on the mechanization and how it differs from the informal treatment.

Note that most of this paper appeared previously in [18]; the relevant section §8.7 therein was in fact written by the present authors. However, since that book was self-published and never peer-reviewed, it should be regarded as a thesis or a preliminary report rather than a publication.

2 Homotopy Type Theory

As in [18], we will work in Martin-Löf type theory extended with Voevodsky's univalence axiom and some higher inductive types. For a type A and elements $x, y : A$, we write the identification type as $x =_A y$ or just $x = y$, and often refer to its elements as *paths*. The defining feature of homotopy type theory is that $x = y$ might have more than one element.

The induction principle for $x = y$, which we refer to as Id-induction or path induction, says that if $D : \prod_{(x,y:A)}(x = y) \to \mathcal{U}$, then to define $d : \prod_{(x,y:A)} \prod_{(p:x=y)} D(x, y, p)$ it suffices to define $r : \prod_{(x:A)} D(x, x, \mathsf{refl}_x)$. From this we can construct all the operations of a higher groupoid on A; for instance, given $p : x = y$ and $q : y = z$ we have their

concatenation $p \cdot q : x = z$ (identification is transitive), and for any $p : x = y$ we have its *inverse* or *opposite* $p^{-1} : y = x$ (identification is symmetric). We also have the operation of *transport* (a.k.a. substitution): given $C : A \to \mathcal{U}$ and $u : C(x)$, for any $p : x =_A y$ we have $\mathsf{transport}^C(p, u) : C(y)$. Finally, for any $f : A \to B$ and $p : x =_A y$, we can define $\mathsf{ap}_f(p) : f(x) =_B f(y)$ (functions respect identifications).

A type A is called a *mere proposition* (or simply a *proposition*) if it has at most one element, i.e. $\prod_{x,y:A} x = y$. It is called a *set* if it satisfies Uniqueness of Identity Proofs, i.e. if $\prod_{x,y:A} \prod_{p,q:x=y} p = q$; or equivalently if each type $x = y$ is a proposition. Propositions and sets are the first two rungs on an infinite ladder of "n-types" [18, Chapter 7], and as such are also called (-1)-*types* and 0-*types* respectively.

If we have two types A and B and functions $f : A \to B$ and $g : B \to A$ such that $\prod_{a:A} g(f(a)) = a$ and $\prod_{b:B} f(g(b)) = b$, then A and B are *equivalent*, written $A \simeq B$. (This is not the definition of "equivalence" — see [18, Chapter 4] for that — but it is how we generally produce equivalences.) Since types are elements of a universe type, we also have the path type $A = B$, with a canonical map $(A = B) \to (A \simeq B)$ since identified types are equivalent; the *univalence axiom* says that this canonical map is itself an equivalence, so that equivalent types are identified.

Higher inductive types (HITs) are a generalization of inductive types that allow constructors which generate new identifications (paths) in addition to elements. For instance, the circle $\mathbb{S}^1$ is a HIT generated by a point $b : \mathbb{S}^1$ and a path $l : b = b$. Note that the path l is "new" and not identified with refl_b (at least, not *a priori* — proving that it is definitely unequal to refl_b is a significant theorem [15]).

The central HIT for us will be the *pushout* $B \sqcup^A C$ of two functions $f : A \to B$ and $g : A \to C$, which is generated by the following constructors:

- $i : B \to B \sqcup^A C$,
- $j : C \to B \sqcup^A C$, and
- for all $x : A$, a path $h(x) : i(f(x)) = j(g(x))$.

As in Fig. 1, the paths $h(x)$ form the "glue", or the "handle" of the barbell. We thus have a commutative diagram

$$\begin{array}{ccc} A & \xrightarrow{\quad g \quad} & C \\ {\scriptstyle f}\downarrow & & \vdots\,{\scriptstyle j} \\ B & \xrightarrow[\quad i \quad]{} & B \sqcup^A C \end{array}$$

that is universal, in the category-theoretic sense. This follows from the type-theoretic induction principle of $B \sqcup^A C$, which says (slightly informally) that given a family $D : B \sqcup^A C \to \mathcal{U}$, to define $d : \prod_{(p:B\sqcup^A C)} D(p)$ it suffices to define $m : \prod_{(b:B)} D(i(b))$ and $n : \prod_{(c:C)} D(j(c))$ which "agree over $h(x)$" for all $x : A$. Details can be found in [18, Chapter 6]. In particular, the "recursion principle" (the case where D is non-dependent) says that to define a map $d : B \sqcup^A C \to D$, it suffices to give maps $m : B \to D$ and $n : C \to D$ and a path $mf = ng$; this is the "existence" part of the universal property of a pushout.

Other important HITs are the *propositional truncation* and the *set-truncation*. The propositional truncation of a type A is a type $\|A\|_{-1}$ that is a proposition, together with a map $|-|_{-1} : A \to \|A\|_{-1}$ that is universal among maps from A to propositions. Informally, $\|A\|_{-1}$ is "$\mathbf{0}$ if A is empty and $\mathbf{1}$ if A is inhabited". Similarly, the set-truncation of A is a type $\|A\|_0$ that is a set, together with a map $|-|_0 : A \to \|A\|_0$ that is universal among maps from A to sets; we think of it as "the set of connected components of A". See [18, Chapter 7] for

how to construct these truncations as HITs, as well as a generalization to the n-truncation (the 1-truncation was mentioned in the introduction).

Given a function $f : A \to B$ and a point $b : B$, the *fiber* of f over b is $\mathsf{fib}_f(b) :\equiv \sum_{(a:A)} f(a) = b$. We say f is an *embedding* if $\mathsf{fib}_f(b)$ is a proposition for all $b : B$, and 0-*truncated* if $\mathsf{fib}_f(b)$ is a set for all $b : B$. Dually, we say f is *surjective* if each $\|\mathsf{fib}_f(b)\|_{-1}$ is contractible (equivalent to $\mathbf{1}$), and *connected* (or 0-connected for emphasis) if each $\|\mathsf{fib}_f(b)\|_0$ is contractible. In particular, a type A is *connected* if the unique function $A \to \mathbf{1}$ is connected, which is equivalent to saying that $\|A\|_0$ is contractible — that is, A has exactly one connected component.

The set-truncation is also how we define the fundamental group and the fundamental groupoid. Given a type A and a point $a : A$, we write $\pi_1(A, a) :\equiv \|a = a\|_0$. (Often one writes simply $\pi_1(A)$, since many types have canonical "basepoints" a. Moreover, if A is connected, we have $\|a = b\|_{-1}$ for all $a, b : A$, and hence $\|\pi_1(A, a) \simeq \pi_1(A, b)\|_{-1}$ as well; so up to "propositional equivalence" the choice of a doesn't matter.) And given just a type A, for any points $x, y : A$ we write $\Pi_1 A(x, y) :\equiv \|x = y\|_0$; this defines the "hom-sets" of a groupoid with A as its type of objects. Note that we have induced groupoid operations

$$\begin{aligned}
(- \cdot -) &: \Pi_1 X(x, y) \to \Pi_1 X(y, z) \to \Pi_1 X(x, z) \\
(-)^{-1} &: \Pi_1 X(x, y) \to \Pi_1 X(y, x) \\
\mathsf{refl}_x &: \Pi_1 X(x, x) \\
\mathsf{ap}_f &: \Pi_1 X(x, y) \to \Pi_1 Y(f(x), f(y))
\end{aligned}$$

for which we use the same notation as the corresponding operations on paths.

The set-truncation also allows us to define quotients of equivalence relations on sets. If A is a set and $R : A \to A \to \mathcal{U}$ is an equivalence relation, then its "homotopy coequalizer" is the HIT generated by

- A quotient map $q : A \to Q$, and
- For each $a, b : A$ such that $R(a, b)$, a path $q(a) = q(b)$.

In general, Q will not be a set; we define the *set-quotient* of R to be its set-truncation $\|Q\|_0$. This has the usual universal property of a quotient with respect to other sets [18, §6.10].

In fact, the definition of the set-quotient makes sense even when A is not a set and when R is not an equivalence relation. That is, for any type A and any type family $R : A \to A \to \mathcal{U}$, we can define its homotopy coequalizer and then set-truncate it; we still call the result the *set-quotient* of A by R. This can be identified with a more usual sort of quotient in the following way:

1. (-1)-truncate the types $R(x, y)$, so that the resulting type family $\|R\|_{-1}$ lands in the type of propositions;
2. Since the type of propositions is a set (by univalence), we can factor $\|R\|_{-1}$ through $\|A\|_0$ to obtain a binary relation on the set $\|A\|_0$;
3. Now consider the quotient of $\|A\|_0$ by the equivalence relation generated by this relation, i.e. its closure under reflexivity, symmetry, and transitivity.

This procedure is fairly complicated, which means on the one hand that it is convenient to have a simpler construction of the set-quotient, but on the other hand that it can be difficult to describe and calculate concretely. This will become relevant in Section 4.

2.1 Encode-decode Proof Style

Many theorems in homotopy type theory, including the Seifert–van Kampen, can be phrased as an equivalence between an abstract, general description X we wish to understand (often a

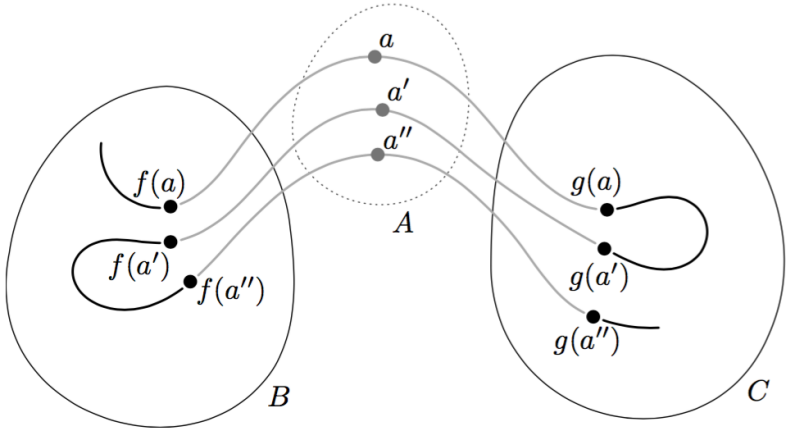

Figure 2 A zigzagging path in the pushout $P :\equiv B \sqcup^A C$.

family of path-spaces or truncated path-spaces), and a concrete, combinatorial description, which we call code. Recall that an equivalence, as mentioned above involves two functions with the proof of their mutual invertibility. We call the function from X to code "encode", and the other "decode". The encode-decode proof style essentially fills in the components of an equivalence one by one:

1. Define the code that will be equivalent to the X we wish to understand. If X is a family of (truncated) path-spaces in some higher inductive type P, then code is usually a type family over P defined using the recursion principle of P.
2. Define a encode function from X to code, and a decode function from code to X. Generally the encode function is immediate from path induction, while decode requires a further induction over the base space P.
3. Show encode and decode are inverse to each other. Again, one direction of this is usually easy, while the other requires an induction.

For the rest of the paper we will follow this recipe.

3 Naive Seifert–van Kampen Theorem

Let $f : A \to B$ and $g : A \to C$ be given functions, and let $P :\equiv B \sqcup^A C$ be their pushout. In Section 3.1 we will define the family $\mathsf{code} : P \to P \to \mathcal{U}$, and in Section 3.2 we will prove that it characterizes the truncated path spaces. In Section 3.3 we will apply this result to calculate some fundamental groups, thereby explaining why it can be regarded as a Seifert–van Kampen theorem.

3.1 Definition of code

To explain the underlying idea of code, first note that that a path from $b : B$ to $c : C$ in the pushout P can be obtained by going first from b to $f(a)$ for some $a : A$, which according to the pushout glue can be identified with $g(a)$, then going from $g(a)$ to c. However, this is not the only way to get from b to c; we might do this to get from a b to a $c' : C$, and then go back from c' to a $b' : B$ in the opposite way (using a different $a' : A$), then back from b' to $c : C$ using a third $a'' : A$; and we could have arbitrarily many such zigzags. See Figure 2. Moreover, this even gives us new ways to get from a point $b : B$ to another point $b' : B$, by

going across to C and back any number of times. And when comparing two such zigzags, we have to compare paths in B and C, but we might also relate the intermediate points in A.

To see how this is related to the classical Seifert–van Kampen theorem, consider the case of paths from $f(a_0)$ to $f(a_0)$ for some fixed $a_0 : A$. These will consist of zigzags of paths passing through B and C, connected to each other through A. If A is connected (as it usually is in classical algebraic topology), then the other points $a : A$ occurring in these zigzags can be identified with a_0 along some path, and so the paths in B and C reduce essentially to loops at $f(a_0)$ and $g(a_0)$ respectively. Thus, such a zigzag is a "word" obtained by "freely concatenating" loops in B and C; but it turns out that loops in A at a_0 can be shifted back and forth. This describes essentially the "amalgamated free product" $\pi_1(B) *_{\pi_1(A)} \pi_1(C)$ appearing in the classical Seifert–van Kampen theorem. However, in order to prove this theorem using the encode-decode method, we need to generalize it to characterize *all* the paths, not just the loops at the base points.

Formally, we define the combinatorial description $\mathsf{code} : P \to P \to \mathcal{U}$ by double recursion on P. In other words, we first apply the recursion principle to the first P in the type of code, concluding that it suffices to define maps $\mathsf{code}_B : B \to P \to \mathcal{U}$ and $\mathsf{code}_C : C \to P \to \mathcal{U}$ that agree on A. Then we apply the recursion principle again for each $b : B$ and each $c : C$, so that to define code_B it suffices to define maps $B \to B \to \mathcal{U}$ and $B \to C \to \mathcal{U}$ that agree in $B \to A \to \mathcal{U}$, and similarly for code_C. The definition of these maps is where we actually put in the zigzags. Finally, we apply the induction principle to each $a : A$ to determine what it means for code_B and code_C to agree on A. When this is all reduced out using the theorems of [18, Chapter 2] (which use function extensionality and the univalence axiom), it suffices for us to give the following.[2]

- $\mathsf{code}(i(b), i(b'))$ is a set-quotient of the type of sequences

  $$(b, p_0, x_1, q_1, y_1, p_1, x_2, q_2, y_2, p_2, \dots, y_n, p_n, b')$$

  where
  - $n : \mathbb{N}$
  - $x_k : A$ and $y_k : A$ for $0 < k \leq n$
  - $p_0 : \Pi_1 B(b, f(x_1))$ and $p_n : \Pi_1 B(f(y_n), b')$ for $n > 0$, and $p_0 : \Pi_1 B(b, b')$ for $n = 0$
  - $p_k : \Pi_1 B(f(y_k), f(x_{k+1}))$ for $1 \leq k < n$
  - $q_k : \Pi_1 C(g(x_k), g(y_k))$ for $1 \leq k \leq n$

  The quotient is generated by the following identifications:

  $$(\dots, q_k, y_k, \mathsf{refl}_{f(y_k)}, y_k, q_{k+1}, \dots) = (\dots, q_k \cdot q_{k+1}, \dots)$$
  $$(\dots, p_k, x_k, \mathsf{refl}_{g(x_k)}, x_k, p_{k+1}, \dots) = (\dots, p_k \cdot p_{k+1}, \dots)$$

  (see Remark 1 below). Note that the type of such sequences is a little subtle to define precisely, since the types of p_k and q_k depend on x_k and y_k; the reader may undertake it as an exercise, or refer to the Agda mechanization.
- $\mathsf{code}(j(c), j(c'))$ is identical, with the roles of B and C reversed. We likewise notationally reverse the roles of x and y, and of p and q.
- $\mathsf{code}(i(b), j(c))$ and $\mathsf{code}(j(c), i(b))$ are similar, with the parity changed so that they start in one type and end in the other.

[2] Since code is a "curried" function of two variables and we are using standard mathematical function application notation, we ought technically to write $\mathsf{code}(a)(b)$; but as in [18] we will instead write this as $\mathsf{code}(a, b)$.

- For $a : A$ and $b : B$, we require an equivalence

$$\mathsf{code}(i(b), i(f(a))) \simeq \mathsf{code}(i(b), j(g(a))). \tag{1}$$

  We define this to consist of the two functions defined on sequences by

$$\begin{aligned}(\dots, y_n, p_n, f(a)) &\mapsto (\dots, y_n, p_n, a, \mathsf{refl}_{g(a)}, g(a)),\\ (\dots, x_n, p_n, a, \mathsf{refl}_{f(a)}, f(a)) &\leftarrow\!\shortmid (\dots, x_n, p_n, g(a)).\end{aligned}$$

  Both of these functions are easily seen to respect the equivalence relations, and hence to define functions on the types of codes. The left-to-right-to-left composite is

$$(\dots, y_n, p_n, f(a)) \mapsto (\dots, y_n, p_n, a, \mathsf{refl}_{g(a)}, a, \mathsf{refl}_{f(a)}, f(a))$$

  which is equal to the identity by a generating path of the quotient. The other composite is analogous. Thus we have defined an equivalence (1).
- Similarly, we require equivalences

$$\begin{aligned}\mathsf{code}(j(c), i(f(a))) &\simeq \mathsf{code}(j(c), j(g(a)))\\ \mathsf{code}(i(f(a)), i(b)) &\simeq \mathsf{code}(j(g(a)), i(b))\\ \mathsf{code}(i(f(a)), j(c)) &\simeq \mathsf{code}(j(g(a)), j(c))\end{aligned}$$

  all of which are defined in exactly the same way (the second two by adding reflexivity terms on the beginning rather than the end).
- Finally, we need to know that for $a, a' : A$, the following diagram commutes:

$$\begin{array}{ccc}\mathsf{code}(i(f(a)), i(f(a'))) & \longrightarrow & \mathsf{code}(i(f(a)), j(g(a')))\\ \downarrow & & \downarrow\\ \mathsf{code}(j(g(a)), i(f(a'))) & \longrightarrow & \mathsf{code}(j(g(a)), j(g(a')))\end{array} \tag{2}$$

  This amounts to saying that if we add something to the beginning and then something to the end of a sequence, we might as well have done it in the other order.

▶ Remark 1. One might expect to see in the definition of code some additional generating equations for the set-quotient, such as

$$\begin{aligned}(\dots, p_{k-1} \cdot \mathsf{ap}_f(w), x'_k, q_k, \dots) &= (\dots, p_{k-1}, x_k, \mathsf{ap}_g(w) \cdot q_k, \dots) && (\text{for } w : \Pi_1 A(x_k, x'_k))\\ (\dots, q_k \cdot \mathsf{ap}_g(w), y'_k, p_k, \dots) &= (\dots, q_k, y_k, \mathsf{ap}_f(w) \cdot p_k, \dots). && (\text{for } w : \Pi_1 A(y_k, y'_k))\end{aligned}$$

However, these are not necessary! In fact, they follow automatically by path induction on w. This is the main difference between the "naive" Seifert–van Kampen theorem and the more refined one we will consider in Section 4.

3.2 The Encode-decode Proof

Before beginning the encode-decode proof proper, we characterize transports in the fibration code:

- For $p : b =_B b'$ and $u : P$, we have

$$\mathsf{transport}^{b \mapsto \mathsf{code}(u, i(b))}(p, (\dots, y_n, p_n, b)) = (\dots, y_n, p_n \cdot p, b').$$

- For $q : c =_C c'$ and $u : P$, we have

$$\mathsf{transport}^{c \mapsto \mathsf{code}(u, j(c))}(q, (\dots, x_n, q_n, c)) = (\dots, x_n, q_n \cdot q, c').$$

Here we are abusing notation by using the same name for a path in X and its image in $\Pi_1 X$. Note that transport in $\Pi_1 X$ is also given by concatenation with (the image of) a path. From this we can prove the above statements by induction on u. We also have:

- For $a : A$ and $u : P$,

$$\mathsf{transport}^{v \mapsto \mathsf{code}(u, v)}(h(a), (\dots, y_n, p_n, f(a))) = (\dots, y_n, p_n, a, \mathsf{refl}_{g(a)}, g(a)).$$

This follows essentially from the definition of code.

Now, as is often the case, the function encode will be defined by transporting a "reflexivity code" along a path. The reflexivity code $r : \prod_{(u:P)} \mathsf{code}(u, u)$ is defined by induction on u as follows:

$$r(i(b)) :\equiv (b, \mathsf{refl}_b, b)$$
$$r(j(c)) :\equiv (c, \mathsf{refl}_c, c)$$

and for $r(h(a))$ we take the composite path

$$\begin{aligned}(h(a), h(a))_*(f(a), \mathsf{refl}_{f(a)}, f(a)) &= (g(a), \mathsf{refl}_{g(a)}, a, \mathsf{refl}_{f(a)}, a, \mathsf{refl}_{g(a)}, g(a)) \\ &= (g(a), \mathsf{refl}_{g(a)}, g(a))\end{aligned}$$

where the first path is by the observation above about transporting in code, and the second is an instance of the set quotient relation used to define code.

▶ **Theorem 2** (Naive Seifert–van Kampen theorem)**.** *For all $u, v : P$ there is an equivalence*

$$\Pi_1 P(u, v) \simeq \mathsf{code}(u, v).$$

Proof. To define a function $\mathsf{encode} : \Pi_1 P(u, v) \to \mathsf{code}(u, v)$ it suffices to define a function $(u =_P v) \to \mathsf{code}(u, v)$, since $\mathsf{code}(u, v)$ is a set. We do this by transport:

$$\mathsf{encode}(p) :\equiv \mathsf{transport}^{v \mapsto \mathsf{code}(u, v)}(p, r(u)).$$

Now to define $\mathsf{decode} : \mathsf{code}(u, v) \to \Pi_1 P(u, v)$ we proceed as usual by induction on $u, v : P$. In each case for u and v, we apply i or j to all the paths p_k and q_k as appropriate and concatenate the results in P, using h to identify the endpoints. For instance, when $u \equiv i(b)$ and $v \equiv i(b')$, we define

$$\begin{aligned}&\mathsf{decode}(b, p_0, x_1, q_1, y_1, p_1, \dots, y_n, p_n, b') \\ &\qquad :\equiv (p_0) \cdot h(x_1) \cdot \mathsf{ap}_j(q_1) \cdot h(y_1)^{-1} \cdot \mathsf{ap}_i(p_1) \cdot \dots \cdot h(y_n)^{-1} \cdot \mathsf{ap}_i(p_n). \quad (3)\end{aligned}$$

This respects the set-quotient equivalence relation and the equivalences such as (1), by the naturality and functoriality of paths [18, Chapter 2].

As usual with the encode-decode method, to show that the composite

$$\Pi_1 P(u, v) \xrightarrow{\mathsf{encode}} \mathsf{code}(u, v) \xrightarrow{\mathsf{decode}} \Pi_1 P(u, v)$$

is the identity, we first peel off the 0-truncation (since the codomain is a set) and then apply path induction. The input refl_u goes to $r(u)$, which then goes back to refl_u (applying a further induction on u to decompose $\mathsf{decode}(r(u))$).

Finally, consider the composite

$$\mathsf{code}(u,v) \xrightarrow{\mathsf{decode}} \Pi_1 P(u,v) \xrightarrow{\mathsf{encode}} \mathsf{code}(u,v).$$

We proceed by induction on $u, v : P$. When $u \equiv i(b)$ and $v \equiv i(b')$, this composite is

$$\begin{aligned}
(b, p_0, x_1, q_1, y_1, p_1, \dots, y_n, p_n, b') &\mapsto \Big(\mathsf{ap}_i(p_0) \cdot h(x_1) \cdot \mathsf{ap}_j(q_1) \cdot h(y_1)^{-1} \cdot \mathsf{ap}_i(p_1) \cdot \\
&\qquad \cdots \cdot h(y_n)^{-1} \cdot \mathsf{ap}_i(p_n)\Big)_*(r(i(b))) \\
&= \mathsf{ap}_i(p_n)_* \cdots \mathsf{ap}_j(q_1)_* h(x_1)_* \mathsf{ap}_i(p_0)_*(b, \mathsf{refl}_b, b) \\
&= \mathsf{ap}_i(p_n)_* \cdots \mathsf{ap}_j(q_1)_* h(x_1)_*(b, p_0, i(f(x_1))) \\
&= \mathsf{ap}_i(p_n)_* \cdots \mathsf{ap}_j(q_1)_*(b, p_0, x_1, \mathsf{refl}_{g(x_1)}, j(g(x_1))) \\
&= \mathsf{ap}_i(p_n)_* \cdots (b, p_0, x_1, q_1, j(g(y_1))) \\
&= \quad \vdots \\
&= (b, p_0, x_1, q_1, y_1, p_1, \dots, y_n, p_n, b').
\end{aligned}$$

i.e., the identity function. (To be precise, there is an implicit inductive argument needed here.) The other three point cases are analogous, and the path cases are trivial since all the types are sets. ◀

3.3 Examples

Theorem 2 allows us to calculate the fundamental groups of many types, provided A is a set, for in that case, each $\mathsf{code}(u,v)$ is, by definition, a set-quotient of a *set* by a relation. (This is because all of its ingredients belong to sets: the intermediate points $x_k : A$ and $y_k : A$ and the elements of truncated path spaces $\Pi_1 B$ and $\Pi_1 C$. We did notate b and b' in the definition of $\mathsf{code}(i(b), i(b'))$ in order to "anchor" the list on both sides, and B may not be a set; but for fixed endpoints $u = i(b)$, $v = i(b')$ these points are not allowed to vary, so they do not prevent $\mathsf{code}(u,v)$ from being a set.)

▶ **Example 3.** Let $A :\equiv \mathbf{2}$ be the 2-element type (the booleans), and let $B :\equiv \mathbf{1}$ and $C :\equiv \mathbf{1}$ be the 1-element type. Then P is equivalent to the circle $\mathbb{S}^1$. Inspecting the definition of, say, $\mathsf{code}(i(\star), i(\star))$, we see that the paths all may as well be trivial, so the only information is in the sequence of elements $x_1, y_1, \dots, x_n, y_n : \mathbf{2}$. Moreover, if we have $x_k = y_k$ or $y_k = x_{k+1}$ for any k, then the set-quotient relations allow us to excise both of those elements. Thus, every such sequence is identified with a canonical *reduced* one in which no two adjacent elements are equal. Clearly such a reduced sequence is uniquely determined by its length (a natural number n) together with, if $n > 1$, the information of whether x_1 is $0_{\mathbf{2}}$ or $1_{\mathbf{2}}$, since that determines the rest of the sequence uniquely. And these data can, of course, be identified with an integer, where n is the absolute value and x_1 encodes the sign. Thus we recover $\pi_1(\mathbb{S}^1) \cong \mathbb{Z}$ [15].

Since Theorem 2 asserts only a bijection of families of sets, this isomorphism $\pi_1(\mathbb{S}^1) \cong \mathbb{Z}$ is likewise only a bijection of sets. We could, however, define a concatenation operation on code (by concatenating sequences) and show that encode and decode form an isomorphism respecting this structure (i.e. an equivalence of groupoids, or "pregroupoids"). We leave the details to the reader.

▶ **Example 4.** Let $A :\equiv \mathbf{1}$ and B and C be arbitrary, so that f and g simply equip B and C with basepoints b and c, say. Then P is the *wedge* $B \vee C$ of B and C (the coproduct in

the category of based spaces). In this case, it is the elements x_k and y_k which are trivial, so that the only information is a sequence of loops $(p_0, q_1, p_1, \dots, p_n)$ with $p_k : \pi_1(B, b)$ and $q_k : \pi_1(C, c)$. Such sequences, modulo the equivalence relation we have imposed, are easily identified with the usual explicit description of the *free product* of the groups $\pi_1(B, b)$ and $\pi_1(C, c)$. Thus, $\pi_1(B \vee C)$ is isomorphic to this free product $\pi_1(B) * \pi_1(C)$.

Theorem 2 is also applicable to some cases when A is not a set, such as the following generalization of Example 3:

▶ **Example 5.** Let $B :\equiv \mathbf{1}$ and $C :\equiv \mathbf{1}$ but A be arbitrary; then P is, by definition, the *suspension* ΣA of A. Then once again the paths p_k and q_k are trivial, so that the only information in a path code is a sequence of elements $x_1, y_1, \dots, x_n, y_n : A$. The first two generating paths say that adjacent equal elements can be canceled, so it makes sense to think of this sequence as a word of the form $x_1 y_1^{-1} x_2 y_2^{-1} \cdots x_n y_n^{-1}$ in a group. Indeed, it looks similar to the free group on A (or equivalently on $\|A\|_0$), but we are considering only words that start with a non-inverted element, alternate between inverted and non-inverted elements, and end with an inverted one. This effectively reduces the size of the generating set by one. For instance, if A has a point $a : A$, then we can identify $\pi_1(\Sigma A)$ with the group presented by $\|A\|_0$ as generators with the relation $|a|_0 = e$.

In particular, if A is connected (that is, $\|A\|_0$ is contractible), it follows that $\pi_1(\Sigma A)$ is trivial. Since the higher spheres can be defined as $\mathbb{S}^{n+1} :\equiv \Sigma \mathbb{S}^n$, and $\mathbb{S}^1$ is easily seen to be connected, it follows that $\pi_1(\mathbb{S}^n) = 1$ for all $n > 1$.

However, Theorem 2 stops just short of being the full classical Seifert–van Kampen theorem, which we recall states that

$$\pi_1(B \sqcup^A C) \cong \pi_1(B) *_{\pi_1(A)} \pi_1(C) \tag{4}$$

(with base point coming from A). Specifically, when A is not a set, the paths in A should be able to "move back and forth" between the images of B and C in the pushout; but the definition of code, and hence the conclusion of Theorem 2, says nothing at all about $\pi_1(A)$! This "moving back and forth" still happens, of course, but it happens quietly by way of type dependency and transport: the paths such as p_k and q_k in $\mathsf{code}(u, v)$ depend on the intermediate points $x_k, y_k : A$, and hence can be transported forwards and backwards along paths in A. These transported paths then get collapsed in the set-quotient that defines $\mathsf{code}(u, v)$. So the quotienting involved in the "amalgamated free product" (4) still happens, but it happens in an "automatic" type-theoretic way that, while easier to define, makes it hard to extract explicit information. For this reason, we now consider a better version of the Seifert–van Kampen theorem.

4 Improvement with an Indexing Space

The improvement of Seifert–van Kampen we present now is closely analogous to a similar improvement in classical algebraic topology, where A is equipped with a *set S of base points*. In fact, it turns out to be unnecessary for our proof to assume that the "set of basepoints" is a *set* — it might just as well be an arbitrary type. The utility of assuming S is a set arises later, when applying the theorem to obtain computations. What is important is that S contains at least one point in each connected component of A. We state this in type theory by saying that we have a type S and a function $\kappa : S \to A$ which is surjective, i.e. (-1)-connected. If $S \equiv A$ and κ is the identity function, then we will recover Theorem 2.

Another example to keep in mind is when A is pointed and (0-)connected, with $\kappa : \mathbf{1} \to A$ the point: by [18, Lemmas 7.5.2 and 7.5.11] this map is surjective just when A is 0-connected.

Let $A, B, C, f, g, P, i, j, h$ be as in the previous section. We now define, given our surjective map $\kappa : S \to A$, an auxiliary type which improves the connectedness of κ. Let T be the higher inductive type generated by

- A function $\ell : S \to T$, and
- For each $s, s' : S$, a function $m : (\kappa(s) =_A \kappa(s')) \to (\ell(s) =_T \ell(s'))$.

There is an obvious induced function $\overline{\kappa} : T \to A$ such that $\overline{\kappa} \circ \ell = \kappa$, and any $p : \kappa(s) = \kappa(s')$ is identified with the composite $\kappa(s) = \overline{\kappa}(\ell(s)) \overset{\overline{\kappa}(m(p))}{=} \overline{\kappa}(\ell(s')) = \kappa(s')$.

▶ **Lemma 6.** *$\overline{\kappa}$ is 0-connected.*

Proof. We must show that for all $a : A$, the 0-truncation of the type $\sum_{(t:T)}(\overline{\kappa}(t) = a)$ is contractible. Since contractibility is a mere proposition and κ is (-1)-connected, we may assume that $a = \kappa(s)$ for some $s : S$. Now we can take the center of contraction to be $|(\ell(s), q)|_0$ where q is the path $\overline{\kappa}(\ell(s)) = \kappa(s)$.

It remains to show that for any $\phi : \left\|\sum_{(t:T)}(\overline{\kappa}(t) = \kappa(s))\right\|_0$ we have $\phi = |(\ell(s), q)|_0$. Since the latter is a mere proposition, and in particular a set, we may assume that $\phi = |(t, p)|_0$ for $t : T$ and $p : \overline{\kappa}(t) = \kappa(s)$.

Now we can do induction on $t : T$. If $t \equiv \ell(s')$, then $\kappa(s') = \overline{\kappa}(\ell(s')) \overset{p}{=} \kappa(s)$ yields via m a path $\ell(s) = \ell(s')$. Hence by definition of $\overline{\kappa}$ and of identification in homotopy fibers, we obtain a path $(\kappa(s'), p) = (\kappa(s), q)$, and thus $|(\kappa(s'), p)|_0 = |(\kappa(s), q)|_0$. Next we must show that as t varies along m these paths agree. But they are paths in a set (namely $\left\|\sum_{(t:T)}(\overline{\kappa}(t) = \kappa(s))\right\|_0$), and hence this is automatic. ◀

▶ Remark 7. T can be regarded as the (homotopy) coequalizer of the "kernel pair" of κ. If S and A were sets, then the (-1)-connectivity of κ would imply that A is the 0-truncation of this coequalizer (this is a standard fact about exact categories, proven in our context in [18, Chapter 10]). For general types, higher topos theory suggests that (-1)-connectivity of κ will imply instead that A is the colimit (a.k.a. "geometric realization") of the "simplicial kernel" of κ. The type T is the colimit of the "1-skeleton" of this simplicial kernel, so it makes sense that it improves the connectivity of κ by 1. More generally, we might expect the colimit of the n-skeleton to improve connectivity by n.

4.1 **New** code

Now we define $\mathsf{code} : P \to P \to \mathcal{U}$ by double induction as follows.

- $\mathsf{code}(i(b), i(b'))$ is now a set-quotient of the type of sequences

$$(b, p_0, x_1, q_1, y_1, p_1, x_2, q_2, y_2, p_2, \dots, y_n, p_n, b')$$

where

- $n : \mathbb{N}$,
- $x_k : S$ and $y_k : S$ for $0 < k \leq n$,
- $p_0 : \Pi_1 B(b, f(\kappa(x_1)))$ and $p_n : \Pi_1 B(f(\kappa(y_n)), b')$ for $n > 0$, and $p_0 : \Pi_1 B(b, b')$ for $n = 0$,
- $p_k : \Pi_1 B(f(\kappa(y_k)), f(\kappa(x_{k+1})))$ for $1 \leq k < n$,
- $q_k : \Pi_1 C(g(\kappa(x_k)), g(\kappa(y_k)))$ for $1 \leq k \leq n$.

The quotient is generated by the following paths, as before:

$$(\dots, q_k, y_k, \mathsf{refl}_{f(y_k)}, y_k, q_{k+1}, \dots) = (\dots, q_k \cdot q_{k+1}, \dots)$$
$$(\dots, p_k, x_k, \mathsf{refl}_{g(x_k)}, x_k, p_{k+1}, \dots) = (\dots, p_k \cdot p_{k+1}, \dots)$$

and also the following new paths (see Remark 1):

$$(\dots, p_{k-1} \cdot \mathsf{ap}_f(w), x'_k, q_k, \dots) = (\dots, p_{k-1}, x_k, \mathsf{ap}_g(w) \cdot q_k, \dots) \quad (\text{for } w : \Pi_1 A(\kappa(x_k), \kappa(x'_k)))$$
$$(\dots, q_k \cdot \mathsf{ap}_g(w), y'_k, p_k, \dots) = (\dots, q_k, y_k, \mathsf{ap}_f(w) \cdot p_k, \dots) \quad (\text{for } w : \Pi_1 A(\kappa(y_k), \kappa(y'_k))).$$

We will need below the definition of the case of decode on such a sequence, which as before concatenates all the paths p_k and q_k together with instances of h to give an element of $\Pi_1 P(i(f(b)), i(f(b')))$, cf. (3). As before, the other three point cases are nearly identical.

- For $a : A$ and $b : B$, we require an equivalence

$$\mathsf{code}(i(b), i(f(a))) \simeq \mathsf{code}(i(b), j(g(a))). \tag{5}$$

Since code is set-valued, by Theorem 6 we may assume that $a = \overline{\kappa}(t)$ for some $t : T$. Next, we can do induction on t. If $t \equiv \ell(s)$ for $s : S$, then we define (5) as in Section 3:

$$(\dots, y_n, p_n, f(\kappa(s))) \mapsto (\dots, y_n, p_n, s, \mathsf{refl}_{g(\kappa(s))}, g(\kappa(s)))$$
$$(\dots, x_n, p_n, s, \mathsf{refl}_{f(\kappa(s))}, f(\kappa(s))) \mapsfrom (\dots, x_n, p_n, g(\kappa(s))).$$

These respect the equivalence relations, and define quasi-inverses just as before. Now suppose t varies along $\mathsf{ap}_{m_{s,s'}}(w)$ for some $w : \kappa(s) = \kappa(s')$; we must show that (5) respects transporting along $\mathsf{ap}_{\overline{\kappa}}(\mathsf{ap}_m(w))$. By definition of $\overline{\kappa}$, this essentially boils down to transporting along w itself. By the characterization of transport in path types, what we need to show is that

$$w_*(\dots, y_n, p_n, f(\kappa(s))) = (\dots, y_n, p_n \cdot \mathsf{ap}_f(w), f(\kappa(s')))$$

is mapped by (5) to

$$w_*(\dots, y_n, p_n, s, \mathsf{refl}_{g(\kappa(s))}, g(\kappa(s))) = (\dots, y_n, p_n, s, \mathsf{refl}_{g(\kappa(s))} \cdot \mathsf{ap}_g(w), g(\kappa(s'))).$$

But this follows directly from the new generators we have imposed on the set-quotient relation defining code.

- The other three requisite equivalences are defined similarly.
- Finally, since the commutativity (2) is a mere proposition, by (-1)-connectedness of κ we may assume that $a = \kappa(s)$ and $a' = \kappa(s')$, in which case it follows exactly as before.

4.2 Improved Theorem

▶ **Theorem 8** (Seifert–van Kampen with a set of basepoints). *For all $u, v : P$ there is an equivalence $\Pi_1 P(u, v) \simeq \mathsf{code}(u, v)$ with code defined as in Section 4.1.*

Proof. Basically just like before. To show that decode respects the new generators of the quotient relation, we use the naturality of h. And to show that decode respects the equivalences such as (5), we need to induct on $\overline{\kappa}$ and on T in order to decompose those equivalences into their definitions, but then it becomes again simply functoriality of f and g. The rest is easy. In particular, no additional argument is required for $\mathsf{encode} \circ \mathsf{decode}$, since the goal is to prove an equality in a set, and so the case of h is trivial. ◀

4.3 Examples

Theorem 8 allows us to calculate the fundamental group of a pushout $B \sqcup^A C$ even when A is not a set, provided S is a set, for in that case, each $\mathsf{code}(u, v)$ is, by definition, a set-quotient of a *set* by a relation. In that respect, it is an improvement over Theorem 2.

▶ **Example 9.** Suppose $S :\equiv \mathbf{1}$, so that A has a basepoint $a :\equiv \kappa(\star)$ and is connected. Then code for loops in the pushout can be identified with alternating sequences of loops in $\pi_1(B, f(a))$ and $\pi_1(C, g(a))$, modulo an equivalence relation which allows us to slide elements of $\pi_1(A, a)$ between them (after applying f and g respectively). Thus, $\pi_1(P)$ can be identified with the *amalgamated free product* $\pi_1(B) *_{\pi_1(A)} \pi_1(C)$ (the pushout in the category of groups). This (in the case when B and C are open subspaces of the pushout P, and A is their intersection) is probably the most classical version of the Seifert–van Kampen theorem.

▶ **Example 10.** As a special case of Example 9, suppose additionally that $C :\equiv \mathbf{1}$, so that P is the cofiber B/A. Then every loop in C is identified with reflexivity, so the relations on path codes allow us to collapse all sequences to a single loop in B. The additional relations require that multiplying on the left, right, or in the middle by an element in the image of $\pi_1(A)$ is the identity. We can thus identify $\pi_1(B/A)$ with the quotient of the group $\pi_1(B)$ by the normal subgroup generated by the image of $\pi_1(A)$.

▶ **Example 11.** As a further special case of Example 10, let $B :\equiv \mathbb{S}^1 \vee \mathbb{S}^1$, let $A :\equiv \mathbb{S}^1$, and let $f : A \to B$ pick out the composite loop $p \cdot q \cdot p^{-1} \cdot q^{-1}$, where p and q are the generating loops in the two copies of $\mathbb{S}^1$ comprising B. Then P is a presentation of the torus T^2 [13]. Thus, $\pi_1(T^2)$ is the quotient of the free group on two generators (i.e., $\pi_1(B)$) by the relation $p \cdot q \cdot p^{-1} \cdot q^{-1} = 1$. This clearly yields the free *abelian* group on two generators, which is $\mathbb{Z} \times \mathbb{Z}$.

▶ **Example 12.** More generally, any CW complex can be obtained by repeatedly "coning off" spheres. That is, we start with a set X_0 of points ("0-cells"), which is the "0-skeleton" of the CW complex. We take the pushout

$$\begin{array}{ccc} S_1 \times \mathbb{S}^0 & \xrightarrow{f_1} & X_0 \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & X_1 \end{array}$$

for some set S_1 of 1-cells and some family f_1 of "attaching maps", obtaining the "1-skeleton" X_1. Then we take the pushout

$$\begin{array}{ccc} S_2 \times \mathbb{S}^1 & \xrightarrow{f_2} & X_1 \\ \downarrow & & \downarrow \\ \mathbf{1} & \longrightarrow & X_2 \end{array}$$

for some set S_2 of 2-cells and some family f_2 of attaching maps, obtaining the 2-skeleton X_2, and so on. The fundamental group of each pushout can be calculated from the Seifert–van Kampen theorem: we obtain the group presented by generators derived from the 1-skeleton, and relations derived from S_2 and f_2. The pushouts after this stage do not alter the fundamental group, since (as noted in Example 5) $\pi_1(\mathbb{S}^n)$ is trivial for $n > 1$.

▶ **Example 13.** In particular, suppose given any presentation of a group $G = \langle X \mid R \rangle$, with X a set of generators and R a set of words in these generators. Let $B :\equiv \bigvee_X \mathbb{S}^1$ and $A :\equiv \bigvee_R \mathbb{S}^1$, with $f : A \to B$ sending each copy of $\mathbb{S}^1$ to the corresponding word in the generating loops of B. It follows that $\pi_1(P) \cong G$; thus we have constructed a connected type whose fundamental group is G. Since any group G has a presentation (e.g. let $X = G$ and R the set of all products that equal the identity), any group is the fundamental group of some type. If we 1-truncate such a type, we obtain a type whose only nontrivial homotopy group is G; this is called an **Eilenberg–Mac Lane space** $K(G, 1)$. (Eilenberg–Mac Lane spaces in homotopy type theory were constructed more explicitly by [14]).

5 Mechanization

Overall, the structure of the Agda proof follows closely the informal argument, giving evidence that homotopy type theory is suitable for mechanization. The major difference from the informal proof is that homotopy pushouts and set quotients are simulated due to the lack of native support. Although there is a trick due to [11], which we use, that enables some of the computation rules to be definitional (those involving point constructors), many computations still need to be manually carried out, resulting in 900 lines of Agda code merely for the well-definedness of the type family code. (The entire mechanization is of roughly 1,800 lines.)

Homotopy pushouts and set quotients can be easily defined as higher inductive types. Unfortunately, Agda like other proof assistants such as Coq [5] was designed for a more traditional variant of Martin-Löf type theory without the univalence axiom and higher inductive types. Lean [6] has built-in quotients and truncations but still only computes on point constructors. The proof assistant Cubical [3] aims to restore full computation rules but was not mature enough at the time of this mechanization.

Finally, the cubical approach [13] is shown to be useful in handling coherence conditions in many theorems, for example the homotopy groups of torus [13], the 3×3 lemma [13], and the Mayer-Vietoris sequences [4]. This might also simplify the proof of Eq. (2) while constructing the family code.

References

1. Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. *Mathematical Structures in Computer Science*, 25:1010–1039, 6 2015. doi:10.1017/S0960129514000486.
2. Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, jan 2009. doi:10.1017/S0305004108001783.
3. Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs*, volume 26 of *Leibniz International Proceedings in Informatics*, pages 107–128. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.TYPES.2013.107.
4. Evan Cavallo. The Mayer-Vietoris sequence in HoTT. In *Oxford Quantum Video*. The QMAC and Clay Mathematics Institute and The EPSRC, 2014. URL: https://youtu.be/6QCFV4op1Uo.
5. The Coq proof assistant. URL: https://coq.inria.fr/.
6. Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *Automated Deduction – CADE-*

25, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015. doi:10.1007/978-3-319-21401-6_26.

7 Nicola Gambino and Richard Garner. The identity type weak factorisation system. *Theoretical Computer Science*, 409(1):94–109, 2008. doi:10.1016/j.tcs.2008.08.030.

8 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford University Press, 1998. arXiv:pLnKggT_In4C.

9 Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine. A mechanization of the Blakers-Massey connectivity theorem in homotopy type theory, May 2016. arXiv:1605.03227.

10 Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky), 2012. arXiv:1211.2851.

11 Daniel R. Licata. Running circles around (in) your proof assistant; or, quotients that compute, 2011. URL: http://homotopytypetheory.org/2011/04/23/running-circles-around-in-your-proof-assistant/.

12 Daniel R. Licata and Guillaume Brunerie. $\pi_n(S^n)$ in homotopy type theory. *CPP*, 2013. URL: http://dlicata.web.wesleyan.edu/pubs/lb13cpp/lb13cpp.pdf.

13 Daniel R. Licata and Guillaume Brunerie. A cubical approach to synthetic homotopy theory. *LICS*, 2015. URL: http://dlicata.web.wesleyan.edu/pubs/lb15cubicalsynth/lb15cubicalsynth.pdf.

14 Daniel R. Licata and Eric Finster. Eilenberg–MacLane spaces in homotopy type theory. *LICS*, 2014. URL: http://dlicata.web.wesleyan.edu/pubs/lf14em/lf14em.pdf.

15 Daniel R. Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *LICS'13*, 2013. arXiv:1301.3443.

16 Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007. URL: http://www.cse.chalmers.se/~ulfn/papers/thesis.html.

17 Charles Rezk. Proof of the Blakers–Massey theorem, 2014. URL: http://www.math.uiuc.edu/~rezk/freudenthal-and-blakers-massey.pdf.

18 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations for Mathematics*. Institute for Advanced Study, git commit hash g662cdd8 edition, 2013. URL: http://homotopytypetheory.org/book.

19 Benno van den Berg and Richard Garner. Topological and simplicial models of identity types. *ACM Transactions on Computational Logic*, 13(1):3:1–3:44, 2012. doi:10.1145/2071368.2071371.

20 Michael A. Warren. *Homotopy theoretic aspects of constructive type theory*. PhD thesis, Carnegie Mellon University, 2008. URL: http://mawarren.net/papers/phd.pdf.