$\pi_n(S^n)$ in Homotopy Type Theory

Daniel R. Licata[1] and Guillaume Brunerie[2]

[1] Wesleyan University
dlicata@wesleyan.edu
[2] Université de Nice Sophia Antipolis
brunerie@unice.fr

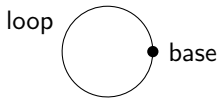
1 Introduction

Homotopy type theory [Awodey and Warren, 2009; Voevodsky, 2011] is an extension of Martin-Löf's intensional type theory [Martin-Löf, 1975; Nordström et al., 1990] with new principles such as Voevodsky's univalence axiom and higher-dimensional inductive types [Lumsdaine and Shulman, 2013]. These extensions are interesting both from a computer science perspective, where they imbue the equality apparatus of type theory with new computational meaning, and from a mathematical perspective, where they allow higher-dimensional mathematics to be expressed cleanly and elegantly in type theory. One example of higher-dimensional mathematics is the subject of homotopy theory, a branch of algebraic topology. In homotopy theory, one studies topological spaces by way of their points, paths (between points), homotopies (paths between paths), homotopies between homotopies (paths between paths between paths), and so on. This infinite tower of concepts—spaces, points, paths, homotopies, and so on—is modeled in type theory by types, elements of types, proofs of equality of elements, proofs of equality of proofs of equality, and so on. A space corresponds to a type A. Points of a space correspond to elements $a,b : A$. Paths in a space are modeled by elements of the identity type (propositional equality), which we notate $p : a =_A b$. Homotopies between paths p and q correspond to elements of the iterated identity type $p =_{a=_A b} q$. The rules for the propositional equality type allow one to define the operations on paths that are considered in homotopy theory. These include identity paths $\mathsf{id} : a = a$ (reflexivity of equality), inverse paths $!\,p : b = a$ when $p : a = b$ (symmetry of equality), and composition of paths $q \circ p : a = c$ when $p : a = b$ and $q : b = c$ (transitivity of equality), as well as homotopies relating these operations (for example, $\mathsf{id} \circ p = p$), and homotopies relating these homotopies, etc. This equips each type with the structure of a (weak) ∞-*groupoid*, as studied in higher category theory [Lumsdaine, 2009; van den Berg and Garner, 2011]. In category theoretic terminology, the elements of a type correspond to objects (or 0-cells), the proofs of equality of elements to morphisms (1-cells), the proofs of equality of proofs of equality to 2-morphisms (2-cells), and so on.

One basic question in algebraic topology is calculating the *homotopy groups* of a space. Given a space X with a distinguished point x_0, the *fundamental group of X at the point x_0* (denoted $\pi_1(X,x_0)$) is the group of loops at x_0 up to homotopy, with composition as the group operation. This fundamental group is the first in a sequence of *homotopy groups*, which provide higher-dimensional information about a space: the

homotopy groups $\pi_n(X,x_0)$ "count" the n-dimensional loops in that space up to homotopy. $\pi_2(X,x_0)$ is the group of homotopies between id_{x_0} and itself, $\pi_3(X,x_0)$ is the group of homotopies between $\mathsf{id}_{\mathsf{id}_{x_0}}$ and itself, and so on. *Calculating a homotopy group* $\pi_n(X,x_0)$ is to construct a group isomorphism between $\pi_n(X,x_0)$ and some explicit description of a group, such as $\mathbb{Z}$ or $\mathbb{Z}_k$ ($\mathbb{Z}$ mod k).

The homotopy groups of a space can be difficult to calculate. This is true even for spaces as simple as the n-dimensional spheres (the circle, the sphere, ...)—some homotopy groups of spheres are currently unknown. A category-theoretic explanation for this fact is that the spheres can be presented as *free* ∞*-groupoids* constructed by certain generators, and it can be difficult to relate a presentation of a space as a free ∞-groupoid to an explicit description of its homotopy groups. For example, the circle is the free ∞-groupoid generated by one point and one loop:



base is a point (object) on the circle, and loop is a path (morphism) from base to itself. That the circle is the free ∞-groupoid on these generators means that all the points, paths, homotopies, etc. on the circle are constructed by applying the ∞-groupoid operations to these generators in a free way. The generator loop represents "going around the circle once counter-clockwise." Using the groupoid operations, one can construct additional paths, such as $!\mathsf{loop}$ (going around the circle once clockwise) and $\mathsf{loop} \circ \mathsf{loop}$ (going around the circle twice counter-clockwise). Moreover, there are homotopies between paths, such as $\mathsf{loop} \circ !\,\mathsf{loop} = \mathsf{id}$ (going clockwise and then counter-clockwise is the same, up to homotopy, as staying still). In this case, one can prove that, up to homotopy, every loop on the circle is either id or $(\mathsf{loop} \circ \mathsf{loop} \ldots \circ \mathsf{loop})$ (n times, for any n) or $(!\,\mathsf{loop} \circ !\,\mathsf{loop} \ldots \circ !\,\mathsf{loop})$ (n times, for any n), and thus that the loops on the circle are in bijective correspondence with the integers. Moreover, under this bijection, concatenation of paths corresponds to addition of integers. Thus, the fundamental group of the circle is $\mathbb{Z}$.

However, in general, it can be quite difficult to relate a presentation of a space as a free ∞-groupoid to an explicit description of its homotopy groups, in part because of *action across levels*. For example, the sphere can be presented as the free ∞-groupoid generated by one point (0-cell) base and one homotopy (2-cell) loop_2 between $\mathsf{id}_{\mathsf{base}}$ (the path that stands still at base) and itself—think of loop_2 as "going around the the surface of the sphere." An ∞-groupoid has group operations at each level, so just as we have identity, inverse, and composition operations on paths (1-cells), we have identity, inverse, and composition operations on homotopies (2-cells). Thus, we can form homotopies such as $\mathsf{loop}_2 \circ \mathsf{loop}_2$ (going around the surface of the sphere twice) and $\mathsf{loop}_2 \circ !\,\mathsf{loop}_2$ (going around the surface of the sphere once in one direction, and then in the opposite direction)—and, analogously to above, there is a homotopy-between-homotopies relating the latter path to the constant homotopy ($\mathsf{loop}_2 \circ !\,\mathsf{loop}_2 = \mathsf{id}_{\mathsf{id}_{\mathsf{base}}}$). Thus, one would expect that the homotopies (2-cells) on the sphere have the same structure as the paths (1-cells) on the circle, and this is indeed the case: $\pi_2(S^2)$ is also $\mathbb{Z}$. However, ∞-groupoids have more structure than just the group operations at each

level—for example, lower-dimensional generators can construct higher-dimensional paths. An example of this is that $\pi_3(S^2)$, the group of homotopies between homotopies (3-cells) on the sphere, is also $\mathbb{Z}$, *despite the fact that there are no generators for 3-cells in the presentation of the sphere!* The paths arise from the applying the algebraic operations of an ∞-groupoid to the 2-cell generator loop_2—and this action across levels is one reason that homotopy groups are so difficult to calculate.

One enticing idea is to use homotopy type theory to calculate homotopy groups: by doing so, we can give computer-checked proofs of these calculations, and we can potentially exploit constructivity and the type-theoretic perspective ∞-groupoids to attack these difficult problems in algebraic topology. To pose the problem of calculating a homotopy group in homotopy type theory, we use two ingredients.

First, we describe basic spaces using *higher inductive types*, which generalize ordinary inductive types by allowing constructors not only for elements of the type, but for paths (proofs of equality) in the type. For example, the circle is represented by a higher inductive type with two constructors

$$\begin{aligned}&\mathsf{base} : S^1\\&\mathsf{loop} : \mathsf{base} =_{S^1} \mathsf{base}\end{aligned}$$

This says that base is a point on the circle, while loop is a path from base to base. In type theory, we express that S^1 is the *free* type with these generators by an elimination rule: to define a function from S^1 into any other type C, it suffices to give a point in $c : C$, which is the image of base, and a loop $p : c =_C c$, which is the image of loop:

$$\frac{c : C \quad p : c =_C c}{\mathsf{S}^1\text{-}\mathsf{rec}_C(c,p) : S^1 \to C}$$

That is, to define a function $S^1 \to C$, it suffices to find a "circle" in C, which gives the image of the generators.

The computation rules for this elimination rule are as follows:

$$\begin{aligned}&\mathsf{S}^1\text{-}\mathsf{rec}_C(c,p)\mathsf{base} :\equiv c\\&\mathsf{ap}(\mathsf{S}^1\text{-}\mathsf{rec}_C(c,p))\mathsf{loop} := p\end{aligned}$$

ap ("**a**ction on **p**aths") applies a function $f : A \to B$ to a path $p : a =_A a'$ to produce a path $fa =_B fa'$. Note that the first computation rule is a definitional equality, while the second is a propositional equality/path—while future versions of homotopy type theory might take this to be a definitional equality, the known semantics of higher inductive types justifies only a propositional equality [Lumsdaine and Shulman, 2013]. To express freeness, we also need to know that $\mathsf{S}^1\text{-}\mathsf{rec}_C(c,p)$ is the unique such map, up to homotopy. This can be expressed either by generalizing the simple elimination rule to a dependent elimination rule, or by adding an η-equality axiom (see [Awodey et al., 2012] for a discussion of these alternatives for ordinary inductive types).

The second ingredient is to define the homotopy groups of a type. One might think that we could define the homotopy groups by iterating the identity type:

$$\begin{aligned}&\pi_1(X,x_0) := x_0 =_X x_0\\&\pi_2(X,x_0) := \mathsf{id}_{x_0} =_{(x_0=_X x_0)} \mathsf{id}_{x_0}\\&\pi_3(X,x_0) := \mathsf{id}_{\mathsf{id}_{x_0}} =_{(\mathsf{id}_{x_0}=_{x_0=_X x_0}\mathsf{id}_{x_0})} \mathsf{id}_{\mathsf{id}_{x_0}}\end{aligned}$$

and so on. However, these iterated identity types may still have non-trivial higher-dimensional structure. The n^{th} homotopy group considers only the structure up to level n, so we need to "kill" the higher-dimensional structure of these types. Thus, we first define the n^{th} *loop space* $\Omega^n(X,x_0)$ so that

$$\begin{aligned}
\Omega^1(X,x_0) &:= x_0 =_X x_0 \\
\Omega^2(X,x_0) &:= \mathsf{id}_{x_0} =_{(x_0=_X x_0)} \mathsf{id}_{x_0} \\
\Omega^3(X,x_0) &:= \mathsf{id}_{\mathsf{id}_{x_0}} =_{(\mathsf{id}_{x_0}=_{x_0=_X x_0}\mathsf{id}_{x_0})} \mathsf{id}_{\mathsf{id}_{x_0}}
\end{aligned}$$

and so on. We write $\Omega(X,x_0)$ for $\Omega^1(X,x_0)$.

Then we can define

$$\pi_n(X,x_0) := ||\Omega^n(X,x_0)||_0$$

where $||A||_0$, the *0-truncation of A*, is a *set* (a type with no higher structure—any two paths are homotopic) constructed by "killing" the higher-dimensional structure of A—i.e. equating any two paths between the same two points. For a more leisurely introduction to these definitions, we refer the reader to previous work [Licata and Shulman, 2013; The Univalent Foundations Program, 2013].

Using these two ingredients, we can use homotopy type theory to calculate homotopy groups of spaces: define the space X as a higher inductive type, and give a group isomorphism between the type $\pi_n(X,x_0)$ (with path composition as the group structure) and an explicit description of a group like $\mathbb{Z}$. In this note, we give a calculation of the fact that $\pi_n(S^n) = \mathbb{Z}$. That is, we describe a proof in homotopy type theory that the n^{th} homotopy group of the n-dimensional sphere is isomorphic to $\mathbb{Z}$. This proof is interesting for several reasons:

- Calculating $\pi_n(S^n)$ is a fairly easy theorem in algebraic topology (e.g. it would be covered in a first- or second-year graduate algebraic topology course), but it is more complex than many of the results that had previously been proved in homotopy type theory. For example, it was one of the first results about an infinite family of spaces, of variable dimension, to be proved in homotopy type theory.
- When doing homotopy theory in a constructive/synthetic style in homotopy type theory, there is always the possibility that classical results will not be provable—the logical axioms for spaces might not be strong enough to prove certain classical theorems about them. Our proof shows that the characterization of $\pi_n(S^n)$ does follow from a higher-inductive description of the spheres, in the presence of univalence, which provides evidence for the usefulness of these definitions and methods.
  Moreover, while we do not yet have a full computational interpretation of univalence, one can see, in the proof, a computational process that transforms n-dimensional loops on the n-sphere into integers. This is one of the first examples of computation with arbitrary-dimensional structures that has been considered in homotopy type theory.
- The proof is not a transcription of a textbook homotopy theoretic proof, but mixes classical ideas with type-theoretic ones. The type-theoretic techniques used here have been applied in other proofs. For example, the proof described here led to a simpler proof of a more general theorem, the Freudenthal Suspension Theorem (Lumsdaine's proof is described in the HoTT book [The Univalent Foundations

Program, 2013]), which gives a shorter calculation of $\pi_n(S^n)$ (also described in the HoTT book). This, in turn, led to a proof of an even more general theorem, the Blakers-Massey theorem [Finster et al., 2013].
- We give a direct higher-inductive definition of the n-dimensional sphere S^n as the free type with a base point base and a loop in $\Omega^n(S^n)$. This definition does not fall into the collection of higher inductive types that has been formally justified by a semantics, because it involves a path constructor at a variable level (i.e. in Ω^n, where n is an internal natural number variable). However, our result shows that it is a useful characterization of the spheres to work with, and it has prompted some work on generalizing schemas for higher inductive types to account for these sorts of definitions.
- The proof we present here includes an investigation of some of the type-theoretic structure of loop spaces. We will characterize $\Omega^n(A)$ for various types A, which explains how concepts such as function extensionality and univalence induce paths at higher levels. This characterization could potentially inform investigations into the computational interpretation of univalence, a major open problem.
- The proof has been formalized in Agda [Norell, 2007], and is available on GitHub in the repository `github.com/dlicata335/hott-agda` (tag `pinsn-cpp-paper`). The proof includes a library of lemmas about iterated loop spaces that is independent of the particular application to n-dimensional spheres.

In the remainder of this paper, we give an informal overview of some of the interesting aspects of the proof, and discuss the Agda formalization. From this point forward, we assume that the reader is familiar with the calculation of $\pi_1(S^1)$ [Licata and Shulman, 2013] and with Part I and Chapter 8 of the HoTT book [The Univalent Foundations Program, 2013].

2 Overview of the Proof

2.1 Definition of the Spheres

We define the n-dimensional sphere S^n (for $n \geq 1$) as the higher inductive type generated by one point base and one point in the n^{th} loop space of S^n at base:

$$\begin{array}{l} \mathsf{base}_n : S^n \\ \mathsf{loop}_n : \Omega^n(S^n, \mathsf{base}) \end{array}$$

The corresponding elimination rule, *sphere recursion*, says that to define a function $S^n \to C$, it suffices to give a point $c : C$ and a loop in $\Omega^n(C,c)$:

$$\frac{C : \mathsf{Type} \quad c : C \quad p : \Omega^n(C,c)}{\mathsf{S^n{-}rec}_C(c,p) : S^n \to C}$$

The computation rules for this elimination rule are as follows:

$$\begin{array}{l} \mathsf{S^n{-}rec}_C(c,p)\mathsf{base}_n :\equiv c \\ \mathsf{ap}^n(\mathsf{S^n{-}rec}_C(c,p))\mathsf{loop}_n := p \end{array}$$

where ap^n applies a function $f : A \to B$ to an n-dimensional loop in $\Omega^n(A,a)$ to get an n-dimensional loop in $\Omega^n(B,fa)$. We discuss the definition of $\Omega^n(X,x_0)$ and ap^n in Section 2.4 below.

We also require a dependent elimination rule, *sphere induction*:

$$\frac{C : S^n \to \mathsf{Type} \quad c : C(\mathsf{base}_n) \quad p : \Omega^n_{\mathsf{loop_n}}(C,c)}{\mathsf{S^n{-}elim}_C(c,p) : \Pi\, x{:}S^n.\, C(x)}$$

Here, the type $\Omega^n_p(C,c)$ represents an "n-dimensional loop-over-a-loop"; it is well-formed when $C : A \to \mathsf{Type}$ and $p : \Omega^n(A,a)$ and $c : C(a)$ (for some A and a). Topologically, it represents an n-dimensional path at c in the total space of C that projects down to p. We discuss the definition of $\Omega^n_p(C,c)$ in Section 2.4 below. The computation rules for sphere induction are similar to those for sphere recursion.

Note that the constructor loop_n is a path whose level depends on n: for $n = 1$, it is a path, for $n = 2$, it is a path between paths, and so on. Because of this, the above definition of S^n does not fall into any of the schemas for higher inductive types that have been formally studied. However, it seems like a sensible notion, because for any fixed n, it expands to a type a higher inductive constructor would be permitted to have: For $n = 1$, it is $\mathsf{base}_1 =_{S^1} \mathsf{base}_1$, for $n = 2$, it is $\mathsf{id} =_{\mathsf{base}_2 =_{S^2} \mathsf{base}_2} \mathsf{id}$, and so on. All of these are iterated identity types in S^n, which is the type being defined. We leave it to future work to justify this kind of definition semantically. Another possible justification would be to take the above rules not as a specification of a higher-inductive type, but as an interface, and implement it by the definition of S^n by iterated suspension [The Univalent Foundations Program, 2013, Section 6.5].

2.2 Calculation of $\pi_n(S^n)$

We now describe the calculation that $\pi_n(S^n) = \mathbb{Z}$ for $n \geq 1$.[3] Formally, this statement means that there is a group isomorphism between the group $\pi_n(S^n)$ (with composition as the group operation) and the additive group $\mathbb{Z}$. In what follows, we will discuss the proof that the *type* $\pi_n(S^n)$ is equivalent (and hence equal, by univalence) to the type $\mathbb{Z}$, and omit the proof that this equivalence sends composition to addition.

The first step is an induction on n. In the base case, we use the homotopy type theory proof of $\pi_1(S^1) = \mathbb{Z}$ described in previous work [Licata and Shulman, 2013]. In the inductive step, the key lemma is that $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$, which, combined with the inductive hypothesis gives the result.

To show that $\pi_{n+1}(S^{n+1}) = \pi_n(S^n)$, we calculate as follows:[4]

---

[3] We write $\pi_n(X)$ for $\pi_n(X,x_0)$ (and similarly for Ω) when the base point is clear from context. For the spheres, if we elide the base point, it is the constructor base_n.

[4] We use the convention of eliding the base point heavily here. The base point of $\Omega(A)$ is id_a, when a is the base point of A. The base point of $||A||_k$ is $|a|$, where a is the base point of A, and $|-|$ is the constructor for the n-truncation type $||A||_n$ [The Univalent Foundations Program, 2013, Section 7.3].

$$\begin{aligned}
\pi_{n+1}(S^{n+1}) &= ||\Omega^{n+1}(S^{n+1})||_0 && \text{definition}\\
&= ||\Omega^n(\Omega(S^{n+1}))||_0 && \text{unfold } \Omega^{n+1}\\
&= \Omega^n(||\Omega(S^{n+1})||_n) && \text{swap truncation and loop space}\\
&= \Omega^n(||S^n||_n) && \text{main lemma}\\
&= ||\Omega^n(S^n)||_0 && \text{swap truncation and loop space}\\
&= \pi_n(S^n) && \text{definition}
\end{aligned}$$

Several of these steps are relatively easy lemmas. For example, we can unfold $\Omega^{n+1}(X)$ as $\Omega^n(\Omega(X))$—one might take this as the definition of Ω^{n+1}, but for the definition below it is a lemma, which we will call LoopPath. Additionally, there is a rule for swapping a truncation with a loop space, incrementing the index:

$$||\Omega(X)||_n = \Omega(||X||_{n+1})$$

Intuitively, $||\Omega(X)||_n$ is a type built from $\Omega(X)$ by equating all $(n+1)$-cells in $\Omega(X)$. However, $\Omega(X)$ is the space of 1-cells (paths) in X, so the $(n+1)$-cells in $\Omega(X)$ are the $(n+2)$-cells in X. Thus, it is equivalent to equate all $(n+2)$-cells in X, and then take the loop space. Iterating this reasoning gives the equation used above, that

$$||\Omega^n(X)||_0 = \Omega^n(||X||_n)$$

This reasoning reduces the problem to proving the main lemma, that

$$||\Omega(S^{n+1})||_n = ||S^n||_n$$

That is, the loop space on the $(n+1)$-sphere is equivalent to the n-sphere, when appropriately truncated. $\Omega(S^{n+1})$ is, by definition, the type $\mathsf{base}_{n+1} =_{S^{n+1}} \mathsf{base}_{n+1}$, so this lemma is characterizing (the truncation of) a path space of a higher-inductive type. A general template for doing such characterizations is the *encode-decode method* [The Univalent Foundations Program, 2013, Section 8.9], which we apply here.

2.3 The encode-decode argument

The bulk of the proof consists of proving that $||\Omega(S^{n+1})||_n = ||S^n||_n$. To build intuition, consider the case of $||\Omega(S^2)||_1 = ||S^1||_1$. Setting aside the truncations for the moment, this means we are comparing points on S^1 (the circle) with loops on S^2 (the sphere). The idea is to set up a correspondence where the base point of the circle (base_1) corresponds to the constant path at the base point of the sphere ($\mathsf{id}_{\mathsf{base}_2}$), and going n times around the loop of the circle (loop_1^n) corresponds to going n times around the surface of the sphere (loop_2^n). In classical topology, it is clear that this correspondence induces a bijection between the set of points on the circle and the set of loops on the sphere (both considered up to homotopy): the circle has one connected component, and any loop on the sphere can be contracted to the constant loop. Moreover, it induces a bijection between the set of loops on the circle and the set of 2-loops on the sphere (again considered up to homotopy), because *every* loop on the circle is loop_1^n for some n, and *every* 2-loop on the sphere is loop_2^n for some n. While proving these facts is essentially what we are doing in this section, it should at least be intuitively plausible that the points and loops

on the circle are the same as the loops and 2-dimensional loops on the sphere. But the loops and 2-dimensional loops on the sphere are the points and loops of the loop space of the sphere, $\Omega(S^2)$, so the points and loops of S^1 are the same as the points and loops of $\Omega(S^2)$.

However, it is not the case that S^1 and $\Omega(S^2)$ are equivalent types, because S^2 has non-trivial 3-dimensional paths, while S^1 has only trivial 2-dimensional paths. Thus, the situation is that the points and paths of S^1 are in correspondence with the points and paths of $\Omega(S^2)$, but the correspondence does not extend to higher dimensions. The role of the truncation is to account for this difference. Comparing $||S^1||_1$ with $||\Omega(S^2)||_1$ considers only points and paths, not any higher-dimensional cells, and restricted to points and paths the above correspondence is in fact an equivalence.

To prove the lemma, we proceed as follows. First, we define a map $S^n \to \Omega(S^{n+1})$ by sphere recursion, where

$$\begin{aligned}
&\mathsf{promote} : S^n \to \Omega(S^{n+1}) \\
&\mathsf{promote}(\mathsf{base}_n) :\equiv \mathsf{id}_{\mathsf{base}_{n+1}} \\
&\mathsf{ap}^n\, \mathsf{promote}\,(\mathsf{loop}_n) := \mathsf{loop}_{n+1}
\end{aligned}$$

In the third line, we omit an application of LoopPath, which coerces $\Omega^{n+1}(S^{n+1})$, the type of loop_{n+1}, to the required type $\Omega^n(\Omega(S^{n+1}))$—$\mathsf{loop}_{n+1}$ can be seen as an n-dimensional loop in $\Omega(S^{n+1})$. promote is one direction of the correspondence described above; for example, it sends loops on the circle to 2-dimensional loops on the sphere. Because functions are functors, we specify only the action on the generators base_n and loop_n; the function automatically preserves identity, composition, etc., and thus, for example, takes the n-fold composition loop_1^n to the n-fold composition loop_2^n, as desired. Thinking of S^{n+1} as the suspension of S^n, this is the meridian map of the suspension, which embeds X into $\Omega(\Sigma X)$—i.e. it's the unit of the suspension/loop space adjunction.

Because truncation is functorial, promote extends to a map $||S^n||_n \to ||\Omega(S^{n+1})||_n$:

$$\begin{aligned}
&\mathsf{decode}' : ||S^n||_n \to ||\Omega(S^{n+1})||_n \\
&\mathsf{decode}'(|x|) = |\mathsf{promote}(x)|
\end{aligned}$$

(where peeling off the truncation is permitted by truncation-elimination, because $||-||_n$ is an n-type).

We would like to show that this map is an equivalence. To define the inverse map $||\Omega(S^{n+1})||_n \to ||S^n||_n$, we need to define a map out of (the truncation of) a path space. One central tool for doing this is univalence: we define a map from S^{n+1} into the universe, so that the base point of S^{n+1} is sent to $||S^n||_n$ and paths in S^{n+1} are sent to equivalences, and then we apply the equivalence determined by a loop to the base point of $||S^n||_n$. In this case, we define a fibration Codes by sphere recursion

$$\begin{aligned}
&\mathsf{Codes} : S^{n+1} \to \mathsf{Type} \\
&\mathsf{Codes}(\mathsf{base}_{n+1}) :\equiv ||S^n||_n \\
&\mathsf{ap}^n\, \mathsf{Codes}\,(\mathsf{loop}_{n+1}) := (\ldots : \Omega^{n+1}(\mathsf{Type}, ||S^n||_n))
\end{aligned}$$

The fiber over the base point is $||S^n||_n$, so Codes will send an element of $\Omega(S^{n+1})$ (which, recall, is notation for $\mathsf{base}_{n+1} =_{S^{n+1}} \mathsf{base}_{n+1}$) to an equivalence $||S^n||_n \simeq ||S^n||_n$.

Thus, we can define a function $\mathsf{encode}' : ||\Omega(S^{n+1})||_n \to ||S^n||_n$ by applying Codes to the given path (after peeling off the truncation brackets, which is allowed because the result is an n-type), and then applying the resulting equivalence to $|\mathsf{base}_n|$:

$$\begin{aligned}&\mathsf{encode}' : ||\Omega(S^{n+1})||_n \to ||S^n||_n\\ &\mathsf{encode}'(|p|) = (\mathsf{ap}(\mathsf{Codes})p)|\mathsf{base}_n|\end{aligned}$$

Eliding truncations for a moment, the term $\mathsf{ap}^n(\mathsf{encode}')$ is a function from $\Omega^n(\Omega(S^{n+1}))$ to $\Omega^n(S^n)$, so it determines (by LoopPath) a function from $\Omega^{n+1}(S^{n+1})$ to $\Omega^n(S^n)$. Because we would like encode' to be inverse to decode', we need to fill in the $\ldots$ in the definition of Codes so that $\mathsf{ap}^n\,\mathsf{encode}'$ sends loop_{n+1} to loop_n (modulo truncations and LoopPath). Thus, we need an element of $\Omega^{n+1}(\mathsf{Type}, ||S^n||_n)$ that is somehow determined by loop_n, so that we get loop_n back out when we apply encode'.

The key maneuver is to apply an equivalence between $\Omega^{n+1}(\mathsf{Type}, A)$ and $\Pi\, x{:}A.\, \Omega^n(A,x)$ (discussed below). That is, an $n+1$-dimensional loop in the space of types with base point A is the same as a family of n-dimensional loops in A, given for each point in A. This reduces the problem to giving an element of type

$$\Pi\, x{:}||S^n||_n.\, \Omega^n(||S^n||_n, x)$$

which (modulo some truncation manipulation) is defined by sphere-elimination, sending base_n to loop_n, and proving that this choice respects loop_n. For $n = 1$, a small calculation is needed to prove this final condition, and for any greater n it is trivial by truncation reasons. This "packages up" loop_n in the Codes fibration in such a way that encode' extracts it.

This defines a fibration Codes over S^{n+1}, where the fiber over the base point is $||S^n||_n$, and the lifting of the $n+1$-loop is an n-dimensional homotopy given by "going around loop_n once". For $n = 1$, this is a fibration over S^2, where the fiber over the base is S^1 (S_1 is already a 1-type, so the truncation cancels), and the lifting of loop_2 goes around the circle—i.e., it is the Hopf fibration (though we do not need to calculate the total space to prove our theorem).

Now that we have defined encode' and decode', the task is to show that they are mutually inverse. The remaining steps required to do so are as follows:

- First, we do a calculation to show that $\mathsf{encode}'(\mathsf{decode}'c) = c$. The proof uses sphere-elimination, and calculations with the loop space library—this is where we prove that encode' takes loop_{n+1} to loop_n.
- The definition of encode' given above in fact has a more general type: it works not only for loops, but for paths to any endpoint x:

  $$\begin{aligned}&\mathsf{encode} : \Pi\, x{:}S^{n+1}.\, (\mathsf{base}_{n+1} =_{S^{n+1}} x) \to \mathsf{Codes}(x)\\ &\mathsf{encode}(|p|) = (\mathsf{ap}(\mathsf{Codes})p)|\mathsf{base}_n|\end{aligned}$$

- Through a somewhat involved calculation with the loop space library, we can show that decode' extends to a function

  $$\mathsf{decode} : \Pi\, x{:}S^{n+1}.\, \mathsf{Codes}(x) \to (\mathsf{base}_{n+1} =_{S^{n+1}} x)$$

  This function is defined by sphere elimination; when x is base_{n+1}, the function is decode'; then we have to prove that this choice respects the loop.

– Now that we have generalized to encode and decode, it is easy to show that

$$\Pi x : S^{n+1}, p : ||\mathsf{base}_{n+1} = x||.\mathsf{decode}_x(\mathsf{encode}_x(p)) = p$$

by path induction, because on the identity path, it is true by definition.

The details of these steps are somewhat intricate, so we refer the reader to the Agda proof.

2.4 Loop space library

Next, we give a brief overview of some of the key lemmas in the loop space library.

Definitions, groupoid and functor structure First, we define

$$\begin{aligned}
&\Omega(X : \mathsf{Type}, x_0 : X) : \mathsf{Type}\\
&\Omega(X, x_0) :\equiv (x_0 =_X x_0)\\
\\
&\Omega^{n\geq 1}(X : \mathsf{Type}, x_0 : X) : \mathsf{Type}\\
&\Omega^1(X, x_0) :\equiv \Omega(X, x_0)\\
&\Omega^{1+n}(X, x_0) :\equiv \Omega(\Omega^n(X, x_0), \mathsf{id}^n)\\
\\
&\mathsf{id}^n : \Omega(X, x_0)\\
&\mathsf{id}^1 :\equiv \mathsf{id}_{x_0}\\
&\mathsf{id}^{1+n} :\equiv \mathsf{id}_{\mathsf{id}^n}
\end{aligned}$$

That is, we define Ω^n mutually with a point id^n of it. This definition unfolds as $\Omega^{1+n}(X) = \Omega(\Omega^n(X))$. An alternative is to define

$$\begin{aligned}
&\Omega_0^1(X, x_0) :\equiv \Omega(X, x_0)\\
&\Omega_0^{1+n}(X, x_0) :\equiv \Omega_0^n(\Omega(X), \mathsf{id}_{x_0})
\end{aligned}$$

i.e. $\Omega_0^{1+n}(X) = \Omega_0^n(\Omega(X))$. By the LoopPath lemma, these two definitions are equivalent, so we can unfold Ω^{1+n} in both ways. The current loop space library takes Ω as the main definition and uses Ω_0 as an auxiliary notion. However, since the different definitions have different definitional behaviors, it would be interesting to try revising the library based on taking Ω_0 as the main notion, to see if it is simpler or not.

In addition to id^n, there are also inverse and composition operations on each loop space:

$$\begin{aligned}
&!^n(l : \Omega^n(X, x_0)) : \Omega^n(X, x_0)\\
&(l_1 : \Omega^n(X, x_0)) \circ^n (l_2 : \Omega^n(X, x_0)) : \Omega^n(X, x_0)
\end{aligned}$$

We prove various groupoid laws for these operations (unit, involution).

Many loop space operations are defined by induction on n. For example, consider applying a function to a loop. Intuitively, the idea is that $\mathsf{ap}^n\, f\, l$ iterates ap to apply f

at the appropriate level. For example, $\mathsf{ap}^2\, f$ should be $\mathsf{ap}(\mathsf{ap}\, f)$, while $\mathsf{ap}^3\, f$ should be $\mathsf{ap}(\mathsf{ap}\,(\mathsf{ap}\, f))$. In general, it is defined as follows:

$$\begin{aligned}&\mathsf{ap}^n(f : X \to Y)(l : \Omega^n(X, x_0)) : \Omega^n(Y, fx_0)\\&\mathsf{ap}^1\, f\, l :\equiv \mathsf{ap} f l\\&\mathsf{ap}^{1+n}\, f\, l :\equiv \mathsf{ap}^{\mathsf{n}}\text{-}\mathsf{id} \circ \mathsf{ap}(\mathsf{ap}^n f) l \circ (!\mathsf{ap}^{\mathsf{n}}\text{-}\mathsf{id})\end{aligned}$$

In the $1+n$ case, $l : \Omega(\Omega^n(X))$. The recursive call $\mathsf{ap}^n f$ has type $\Omega^n(X, x_0) \to \Omega^n(Y, f(x_0))$. Thus, using ap to apply this to the path l gives an element of

$$\mathsf{ap}^n\, f\, \mathsf{id}^n = \mathsf{ap}^n\, f\, \mathsf{id}^n$$

We require an element of $\mathsf{id}^n = \mathsf{id}^n$, so we compose on both sides with a proof $\mathsf{ap}^{\mathsf{n}}\text{-}\mathsf{id}$ that ap^n preserves identities. Many definitions on Ω^n follow this template: an induction on n, defined mutually with a lemma stating preservation of identities. ap^n also preserves inverses and composition, and is functorial in the function position:

$$\begin{aligned}&\mathsf{ap}^n(\lambda x.x) l = l\\&\mathsf{ap}^n(g \circ f) l = \mathsf{ap}^n\, g(\mathsf{ap}^n\, f\, l)\end{aligned}$$

Another key lemma is that ap^{1+n} can be unfolded in the other assocativity: above, we essentially defined $\mathsf{ap}^{1+n} = \mathsf{ap}(\mathsf{ap}^n f)$. We can prove that it is also equal to $\mathsf{ap}^n(\mathsf{ap} f)$ (with the appropriate $\mathsf{LoopPath}$ coercions inserted).

Some properties hold only for $n \geq 2$. For example,

$$\mathsf{ap}^n\, !\, l = !^n l$$

This follows from the Eckmann-Hilton argument, which shows that the higher homotopy groups are abelian, and that the two different ways of composing higher-dimensional loops are homotopic.

Loops in Types For many types A, one can give a straightforward characterization of the paths in A. For example:

- Paths $f =_{A \to B} g$ are equivalent to paths $\Pi x{:}A. fx =_B gx$, by function extensionality.
- Paths $A =_{\mathsf{Type}} B$ are equivalent to equivalences between A and B, by univalence.
- Paths $e_1 =_{A \simeq B} e_2$ between equivalences are equivalent to paths $e_1 =_{A \to B} e_2$ (where we implicitly cast e_i from an equivalence to a function), because being an equivalence is an hprop.

One important piece of the loop space library is an investigation of how these characterizations extend to higher-dimensional loop spaces.

Functions. First, we characterize $\Omega^n(\Pi x{:}A. B, f)$. For $n = 1$, it is equal to $\Pi x{:}A. \Omega^n(B, fx)$ by function extensionality. For $n = 2$, the question is to characterize the type

$$\mathsf{id}_f =_{f =_{\Pi x:A.B} f} \mathsf{id}_f$$

But by applying function extensionality (and using its action on id_f), this type is equivalent to

$$(\lambda x.\mathsf{id}_{fx}) =_{\Pi x{:}A.fx=_{B(x)}fx} (\lambda x.\mathsf{id}_{fx})$$

Using function extensionality again, this type is equivalent to

$$\Pi x{:}A.\, id_{fx} =_{fx=_{B(x)}fx} \mathsf{id}_{fx}$$

which is $\Pi x{:}A.\, \Omega^2(B(x), fx)$.

Indeed, in general, we prove

$$\Omega^n(\Pi x{:}A.B, f) = \Pi x{:}A.\, \Omega^n(B(x), fx)$$

That is, a loop in a function space is a family of loops.

Paths between types. Second, we characterize $\Omega^n(A =_{\mathsf{Type}} A, p)$, where p is path in the universe from A to A. Consider $n = 1$: by univalence, we know that $\Omega(A = A, p)$ is equivalent to $\Omega(A \simeq A, p^*)$, where p^* is the equivalence induced by the path p. But a path between equivalences is equivalent to a path between the underlying functions: an equivalence between A and B is a pair (f, i) where $f : A \to B$ and $i : \mathsf{IsEquiv}(f)$, and being an equivalence is an hprop, so the second components of such pairs are always equal. Thus,

$$\Omega(A = A, p) = \Omega(A \to A, \mathsf{coe}(p))$$

where $\mathsf{coe}(p : A = B) : A \to B$—i.e. coe ("coerce") can be thought of as turning the path into an equivalence, and then selecting the "forward" direction. We can prove by induction that this rule extends to higher dimensions, so that

$$\Omega^n(A = A, p) = \Omega^n(A \to A, \mathsf{coe}(p))$$

Loop spaces. The LoopPath lemma mentioned above also fits this pattern: it characterizes an n-dimensional loop in a loop space as a $(1+n)$-dimensional loop in the underlying space:

$$\Omega^n(\Omega(A), \mathsf{id}_a) = \Omega^{1+n}(A, a)$$

Putting it all together. Combining the previous three lemmas, we have that

$$\begin{aligned}
\Omega^{1+n}(\mathsf{Type}, A) &= \Omega^n(\Omega(\mathsf{Type}, A), \mathsf{id}_A) && \text{loop in loop space} \\
&= \Omega^n(A =_{\mathsf{Type}} A, \mathsf{id}_A) && \text{definition} \\
&= \Omega^n(A \to A, \lambda x.x) && \text{loop in path between types} \\
&= \Pi x{:}A.\, \Omega^n(A, x) && \text{loop in function type}
\end{aligned}$$

This equivalence is the "key maneuver" that we used to define the Codes fibration in Section 2.3 above.

Loops over a Loop The notion of a "path over a path" is a key ingredient of the dependent elimination rule for higher inductive types. Given $p : a_1 =_A a_2$, and a fibration $B : A \to \mathsf{Type}$, a path over p relates $b_1 : B(a_1)$ to $b_2 : B(a_2)$. This is a kind of heterogeneous equality [McBride, 2000] between elements of different instances of B. However, the notion of a path over a path need not be taken as primitive, since it can be represented by a homogeneous path $\mathsf{transport}_B\, p\, b_1 =_{B(a_2)} b_2$.

In the library, we define an n-dimensional loop over a loop $\Omega^n_p(B, b_0)$ (where $p : \Omega^n(A, a_0)$ and $b_0 : B(a_0)$) by induction on n, using $\mathsf{transport}$ at each level to account for the heterogeneity. For small n, this definition gives the expected dependent elimination rule for, e.g., S^1 and S^2 and S^3, the specific spheres whose higher inductive elimination rule had previously been written out. For example, for S^2, sphere elimination with $B : S^2 \to \mathsf{Type}$ and $b : B(\mathsf{base})$ requires a proof of

$$(\mathsf{transport}(\lambda x.(\mathsf{transport}\, B\, x\, b) = b)\, \mathsf{loop}_2\, \mathsf{id}) = \mathsf{id}$$

A key lemma relates this definition of loop-over-a-loop to an alternate characterization, which coalesces all of the $\mathsf{transport}$s into a single use of ap^n combined with the equivalence defined above between $\Omega^{1+n}(\mathsf{Type}, A)$ and $\Pi x{:}A.\, \Omega^n(A, x)$. The reason this lemma is important is that we then give rules for $\mathsf{ap}^n A$ driven by the structure of A, such as when A is $\lambda x.A_1(x) \to A_2(x)$ or when A is $\lambda x.f(x) = g(x)$ or when A is $\lambda x.||A_1(x)||_k$. These rules are higher-dimensional analogues of the computation rules for $\mathsf{transport}_A$ that are driven by the structure of A.

3 Formalization

The calculation of $\pi_n(S^n)$ described in Sections 2.2 and 2.3 is in `homotopy/PiNSN.agda`; it is about 250 lines of code. The loop space library described in Section 2.4 is in `lib/loopspace/`; it is about 1500 lines of code. The proof of $\pi_n(S^n)$, including specifying the lemmas it uses from the loop space library, took a few days. The loop space library then took a couple of weeks to complete, working for perhaps 4 hours per day.

The formalization has one cheat, which is that it takes place in an Agda homotopy type theory library that uses type:type as a terser form of universe polymorphism than Agda's. More recent homotopy type theory libraries use Agda's universe polymorphism instead of type:type, and we believe that the proof could be ported to such a library.

Agda does not provide very much proof automation for this proof: the bulk of the proof is manual equational calculations with the loop space operations. However, with an improved computational understanding of homotopy type theory, some of these calculation steps might be definitional equalities.

That said, Agda was quite useful as a proof checker, and for telling us what we needed to prove next. The terms involved in the calculations get quite long, so it would be difficult to do these calculations, or to have confidence that they were done correctly, without the use of a proof checker.

It is worth describing one new device that we developed for this proof, which is a combination of a mathematical and a engineering insight. Often in this proof, we are manipulating paths that have the form $\mathsf{p} \circ \mathsf{q} \circ !\ \mathsf{p}$ (q conjugated by p), where q is thought of as "the actual path of interest" and p is some "coercion" or "type cast" that shows that

it has some desired type. Early in the development of the proof, we got stuck, because manipulating these coercions explicitly gets quite cumbersome.

The engineering insight is that, if we define a function adj p q that returns p ∘ q ∘ !p, but make it *abstract* (i.e. hide its definition), then Agda can fill in in terms of the form adj _ q in the middle of an equational calculation by unification. By stating the coercions at the beginning and end of the proof, and using lemmas that propagate this information without explicitly mentioning it, we need not state the coercions at each step of the proof. Though adj p q is abstract, we export a propositional equality equating it to p ∘ q ∘ !p, so that we can use this technique in the intermediate steps of a calculation; the overall theorem is the same.

The mathematical insight is that, for an element p of a doubly-iterated identity type (i.e. when p is at least a path between paths), for any coercions q and q' of the same type, $(q \circ p \circ !\, q) = (q' \circ p \circ !q')$. This is a consequence of the higher homotopy groups being abelian.

Combining these two insights, we can let Agda infer the coercions as we proceed through the steps of the proof, and then, at the end, when we need the inferred coercion to turn out to be a specific one, we simply apply the lemma. As a practical matter, this technique for managing these coercions was essential to our being able to complete this proof.

For example, here is a snippet of an equational deduction without applying the technique:

```
adjust (ap^-id n (λ f → f x) {f}) (ap (λ f → apl n f x) (adjust (λl-id n) (ap (λl n) (λ≃ a)))) ≃⟨ ... ⟩
adjust (ap^-id n (λ f → f x) {f}) (ap (λ f → apl n f x) (adj _ (ap (λl n) (λ≃ a)))) ≃⟨ .. ⟩
adj (ap^-id n (λ f₁ → f₁ x)) (ap (λ f → apl n f x) (adj _ (ap (λl n) (λ≃ a)))) ≃⟨ ... ⟩
adj (ap^-id n (λ f₁ → f₁ x)) ∘ ap (λ f' → apl n f' x) (λl-id n)) (ap (λ f → apl n f x) (ap (λl n) (λ≃ a))) ≃⟨ ... ⟩
adj (ap^-id n (λ f₁ → f₁ x)) ∘ ap (λ f' → apl n f' x) (λl-id n)) (ap (λ f → apl n (λl n f) x) (λ≃ a)) ≃⟨ ... ⟩
adj ((ap^-id n (λ f₁ → f₁ x)) ∘ ap (λ f' → apl n f' x) (λl-id n)) ∘ ap≃ (! (β n (λ x₁ → id^ n)))) (ap (λ f → f x) (λ≃ a)) ≃⟨ ... ⟩
adj ((ap^-id n (λ f₁ → f₁ x)) ∘ ap (λ f' → apl n f' x) (λl-id n)) ∘ ap≃ (! (β n (λ x₁ → id^ n)))) (a x) ≃⟨ ... ⟩
adj id (a x) ≃⟨ ! (adj-id _) ⟩
a x ■
```

Here is the same snippet, where we use the technique and replace the first argument to adj with _:

```
adjust (ap^-id n (λ f → f x) {f}) (ap (λ f → apl n f x) (adjust (λl-id n) (ap (λl n) (λ≃ a)))) ≃⟨ ... ⟩
adjust (ap^-id n (λ f → f x) {f}) (ap (λ f → apl n f x) (adj _ (ap (λl n) (λ≃ a)))) ≃⟨ ... ⟩
adj _ (ap (λ f → apl n f x) (adj _ (ap (λl n) (λ≃ a)))) ≃⟨ ... ⟩
adj _ (ap (λ f → apl n f x) (ap (λl n) (λ≃ a))) ≃⟨ ... ⟩
adj _ (ap (λ f → apl n (λl n f) x) (λ≃ a)) ≃⟨ ... ⟩
adj _ (ap (λ f → f x) (λ≃ a)) ≃⟨ ... ⟩
adj _ (a x) ≃⟨ ... ⟩
adj id (a x) ≃⟨ ... ⟩
a x ■
```

Moreover, if we did not appeal to the fact that any two coercions give equal results, it is unclear how we would even prove, between the second-to-last and third-to-last lines, that

$$((\mathsf{ap\hat{}\text{-}id}\ \mathsf{n}\ (\lambda\ \mathsf{f}_1 \to \mathsf{f}_1\ \mathsf{x}) \circ \mathsf{ap}\ (\lambda\ \mathsf{f'} \to \mathsf{apl}\ \mathsf{n}\ \mathsf{f'}\ \mathsf{x})\ (\lambda\mathsf{l}\text{-}\mathsf{id}\ \mathsf{n})) \circ \mathsf{ap}{\simeq}\ (!\ (\beta\ \mathsf{n}\ (\lambda\ \mathsf{x}_1 \to \mathsf{id\hat{}}\ \mathsf{n})))) = \mathsf{id}$$

The inferred coercion (the left-hand-side of this equation) uses several loop space lemmas that are defined by induction on n, and it is unclear how to prove that they cancel each other.

4 Conclusion

In this paper, we have described a computer-checked calculation of $\pi_n(S^n)$ in homotopy type theory. One important direction for future work is to develop a computational interpretation of homotopy type theory; our proof would be a good test case for such an interpretation. Given a number k, how does the proof compute the path loop_n^k? Or, more interestingly, given a path on S^n, how does the proof compute a number? Another direction would be to investigate the relationship between this proof and proofs of $\pi_n(S^n)$ in classical homotopy theory. The proof we have described here has since been generalized to a proof of the Freudenthal Suspension Theorem [The Univalent Foundations Program, 2013], which is one way that $\pi_n(S^n)$ is proved in classical homotopy theory. However, it would be interesting to see whether the more specific proof presented here has been (or can be) phrased in classical terms.

Acknowledgments. We thank the participants of the Institute for Advanced Study's special year on homotopy type theory for uncountably many helpful conversations.

This material is based in part upon work supported by the National Science Foundation under grants CCF-1116703 and DMS-1128155 and by the Institute for Advanced Study's Oswald Veblen fund. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or any other sponsoring entity.

Bibliography

S. Awodey and M. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 2009.

S. Awodey, N. Gambino, and K. Sojakova. Inductive types in homotopy type theory. In *IEEE Symposium on Logic in Computer Science*, 2012.

E. Finster, D. R. Licata, and P. L. Lumsdaine. The Blakers-Massey theorem in ∞-topoi and homotopy type theory. In preparation, 2013.

D. R. Licata and M. Shulman. Calculating the fundamental group of the cirlce in homotopy type theory. In *IEEE Symposium on Logic in Computer Science*, 2013.

P. L. Lumsdaine. Weak ω-categories from intensional type theory. In *International Conference on Typed Lambda Calculi and Applications*, 2009.

P. L. Lumsdaine and M. Shulman. Higher inductive types. In preparation, 2013.

P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. Rose and J. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73 – 118. Elsevier, 1975.

C. McBride. *Dependently Typed Functional Programs and Their Proofs*. PhD thesis, University of Edinburgh, 2000.

B. Nordström, K. Peterson, and J. Smith. *Programming in Martin-Löf's Type Theory, an Introduction*. Clarendon Press, 1990.

U. Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations Of Mathematics*. Institute for Advanced Study, 2013. Available from `homotopytypetheory.org/book`.

B. van den Berg and R. Garner. Types are weak ω-groupoids. *Proceedings of the London Mathematical Society*, 102(2):370–394, 2011.

V. Voevodsky. Univalent foundations of mathematics. Invited talk at WoLLIC 2011 18th Workshop on Logic, Language, Information and Computation, 2011.