Representability of Homotopy Groups in Type Theory

Brandon Shapiro
Constructive Type Theory

The emerging field of homotopy type theory is built around the idea that in 'intensive' type theory types can be meaningfully interpreted as topological spaces up to a kind of deformation called 'homotopy'. This correspondence is valuable both for type theorists, who can benefit from applying intuition from topological ideas to types, and for more traditional mathematicians, who can write formally verifiable proofs of classically complicated results by defining topological spaces as 'higher inductive types'. It is therefore desirable for computer scientists to have a basic understanding of homotopy theory, a topic rather difficult to find in basic math classes (it would not be unusual for this kind of homotopy theory to appear in a third or fourth topology course).

A basic fact in homotopy theory is that the fundamental group of a topological space X, which describes the structure of deformable loops in X, has two equivalent definitions. The first approach considers the 'loop space' $\Omega(X)$ consisting of all loops based at a point in X, and defines the fundamental group $\pi_1(X)$ as the set $\pi_0(\Omega(X))$ of 'connected components' of the loop space. The second approach avoids having to work with loop spaces by instead defining the fundamental group as the set $[S^1, X]$ of based continuous functions from the circle into X considered up to 'homotopy', meaning continuous deformations of functions.

I will develop the background necessary to demonstrate that this equivalence of definitions of homotopy groups is valid for types. We will see that the type theoretic definitions of the circle, higher dimensional spheres, and loop spaces make this equivalence practically trivial, but nonetheless a valuable illustration of how higher inductive types and recursion principles from type theory are valuable tools for reasoning about topological ideas. While these ideas are straightforward to infer from the presentation of homotopy type theory in [1], to my knowledge they do not appear in that text.

This paper is organized as follows: Section 1 establishes notation (all from [1]) for standard type theoretic constructions. Section 2 discusses recursion principles which will be essential to the main result. Section 3 explains how identity types give rise to the homotopy interpretation of type theory. Section 4 establishes some basic definitions regarding pointed types that arise in the statement of the main result. Section 5 introduces the Univalence Axiom and uses it to characterize equality of types and pointed types. Section 6 introduces the higher inductive types corresponding to the circle and higher spheres. Sections 7-9 cover the main result on equivalent definitions of homotopy groups, first for loop spaces, then the set of connected components, and finally for the group structure. As this is an expository paper, in certain places I forgo some technical proof details when they aren't essential to understand the main ideas in play.

1. Type Theory Notation

I follow the notation of [1], writing $\mathbf{0}$, $\mathbf{1}$, $A \times B$, $A + B$, $A \to B$ for the empty type, the unit type, the product type of types A and B, the sum type of A and B, and type of functions from A

to B. $\mathcal{U}$ will refer to the universe type whose elements are types including all types considered here (except $\mathcal{U}$ itself and the type $\tilde{\mathcal{U}}$ of pointed types). In this paper only one universe will be necessary.

Whereas functions from a type A traditionally take values in a single type B, it is often convenient for the type of the value to depend on the input: for any function $B : A \to \mathcal{U}$ associating a type to each element of A, we can define a 'dependent function' as an assignment of an element of $B(a)$ for each $a : A$. The type of such dependent functions is written $\prod_{a:A} B(a)$, a notation justified by the idea that such an assignment is no different from an iterated product over the type A of the types $B(a)$.

In a similar vein, for such a $B : A \to \mathcal{U}$, we have the dependent sum type $\sum_{a:A} B(a)$ with elements of the form (a, b) with $b : B(a)$. Here the notation is similarly justified by the idea of a sum indexed over A of the types $B(a)$. To minimize redundant symbols, I will write $\prod_{a,b:A}$ for $\prod_{a:A} \prod_{b:A}$ and the same for dependent sums.

For a type A with elements $a, b : A$, $a =_A b$ (or simply $a = b$ when the encompassing type A is clear) will denote the 'identity type' of a and b in A. The intuition for this notation is that $a = b$ is inhabited precisely when a and b are equal in A. However, we shall see that this notion of equality in a type is somewhat looser than the analogous notion for sets, as suggested by the possibility of inhabited identity types with multiple elements (this is the 'intensional' approach to type theory). One way of thinking about this is that elements of $a = b$ (sometimes called equalities) correspond to proofs of or 'witnesses to' the equality of a and b. These types are generated in a sense (to be explained in more detail in the next section) by the elements $refl_a : a = a$ for all $a : A$. These elements provide a formal statement of the obvious fact that in any type an element a is equal to itself (though we will see types in which there are also nontrivial elements of this type).

Identity types provide the clearest example of types corresponding to logical propositions. "$a =_A b$" makes sense both as a logical statement and a type, and this will generally be the case for propositions regarding types. Accordingly, the type constructors $\mathbf{0}$, $\mathbf{1}$, $\times$, $+$, $\to$, $\prod$, and $\sum$ correspond respectively to the logical constructs $false$, $true$, and, or, $implies$, $forall$, and $exists$. For instance, the statement "for all b in B there exists a a in A such that $f(a)$ is equal to b can be expressed as the type $\prod_{b:B} \sum_{a:A} f(a) = b$, an element of which provides for each $b : B$ an element $a : A$ and witness to the equality of $f(a)$ and b. We will follow the convention of type theory and express our propositions as types, proving them by constructing an element of the type.

As equality in the sense of $a =_A b$ is in the internal language of our type theory, we will avoid ambiguity by writing $Name := thing$ to define the term $Name$ as the $thing$ on the other end of the $:=$, and then $Name \equiv thing$ to indicate that $Name$ and $thing$ are the same by definition outside the language of type theory (so $Name \equiv thing$ is a statement in English about type theory that is not a type). If we have $a \equiv b$, then it is valid to replace b with a in any expression containing b, so that for instance $refl_a : a = b$ as $(a = a) \equiv (a = b)$. We will call this definitial equality 'judgmental equality' and the type theoretic equality described by the type $a =_A b$ 'propositional equality'.

2. Recursion Principles

Typically, the definition of a type A specifies how to construct elements of A. However, in order to 'do anything' with those elements, one must know how to construct functions from A to other types. A recursion principle for a type A characterizes these functions completely and uniquely for the types we will consider ('inductive types').

For instance, the recursion principle for the type $A + B$ states that for a type C, functions $A+B \to C$ correspond to pairs of functions $A \to C$ and $B \to C$. That for $A \times B$ says that functions $A \times B \to C$ correspond to functions $A \to (B \to C)$. As an even simpler example, functions $\mathbf{1} \to C$ correspond to simply elements of C, so to specify a function from $\mathbf{1}$ to C, one just needs to choose an element of C for the unique element of $\mathbf{1}$ to be sent to.

Even more powerful are induction principles, which characterize dependent functions out of types the same way recursion principles do for non-dependent functions. For the purposes of this paper, it will only be necessary to discuss the induction principle for the family of identity types of a type: given a dependent function $C : \prod_{a,b:A} \Big((a =_A b) \to \mathcal{U}\Big)$, to specify an element of $\prod_{a,b:A} \prod_{p:a=b} C(x, y, p)$ it suffices to give an element of $\prod_{a:A} C(a, a, refl_a)$. In other words, interpreting $C(a, b, p)$ as a proposition depending on a, b, and p, to prove something about the elements of all identity types of A it suffices to prove it for those of the form $refl_a$. It is a subtle point that the property (C) in question must range over all identity types in A, in that propositions about all elements of just $a = a$ cannot be proven using only the case of $refl_a$. We will see examples of this sort of proof in the next section.

3. Equality and Paths

Classically, a notion of equality must be reflexive, symmetric, and transitive. While our 'propositional' equality in terms of identity types is not a priori required to conform to these standards, it does so in a way that suggests a rich structure underlying our notion of a type. Identity types as defined by the properties discussed above obey type theoretic translations of each of these properties of equality for any type A:

Lemma 1. *(Reflexivity)* $\prod_{a:A} a = a$

Proof Clearly $\lambda(a : A).refl_a : \prod_{a:A} a = a$. □

Lemma 2. *(Symmetry)* $\prod_{a,b:A} (a = b) \to (b = a)$

Proof Rewriting the statement as a dependent function on $a = b$ (where the function into $\mathcal{U}$ is constant at $b = a$) we want to show $\prod_{a,b:A} \prod_{p:a=b} b = a$. Now using the induction principle for identity types we can assume that a and b are the same (say a) and that p is $refl_a$. It then only remains to construct an element of (a, a), which can be chosen to be $refl_a$. □

For $p : a = b$, we write p^{-1} for its image in $b = a$, called the 'inverse' of p.

Lemma 3. *(Transitivity)* $\prod_{a,b,c:A} (a = b) \to (b = c) \to (a = c)$

Proof This time more informally, we can assume by the induction principle that $a \equiv b$ and that for $p : a \to b$ and $q : b \to c$, we have $p \equiv refl_a$. Then $(a = c) \equiv (b = c)$, so we have an element of $a = c$ in q. □

We will write $p \cdot q$ for the element of $a = c$ obtained from p and q. Equality in types is now shown to be type theoretically reflexive, symmetric and transitive. But these properties are expressed as functions in such a way that, if a, b, c are all the same element, look a lot like the operations of a group: $refl_a$ could be the identity, p^{-1} the inverse of an element $p : a = a$, and the transitivity function the group multiplication. In fact, we have that $p \cdot p^{-1} = refl_a$ as well as other equalities corresponding to the identity, inverse, and associativity properties of a group, all of which are straightforward to prove using the induction principle.

But these operations are defined on the entire family of identity types for A, which looks less like a group as only compatible (or rather composable as this mirrors the conditions for function composition) elements can be 'multiplied' using transitivity. This gives a type and its identities a structure reminiscent of a groupoid: a collection of objects and arrows between them where each object has an identity arrow, each arrow has an inverse pointing in the opposite direction, and arrows can be composed associatively with the identities and inverses behaving as one would expect in a group. A group is simply a groupoid with a single object, where the elements of the group are the arrows from that object to itself (which by definition include an identity and inverses).

But we don't quite have a groupoid. Statements of equality like $p \cdot p^{-1} = refl_a$ are identity types *of the identity type* $a =_A a$, and while we can prove them to be occupied, establishing the propositional equality of $p \cdot p^{-1}$ and $refl_a$, we have no reason to assume that they are occupied by elements of the form $refl_{refl_a}$. That is, they are not judgmental equalities. $p \cdot p^{-1}$ and $refl_a$ are two different elements of $a = a$ which are equal only under our weaker notion of propositional equality. We say then that a type A and its identity types satisfy the groupoid laws only 'up to higher identities'. And this structure extends through each nested level of identity type: at the top the type A, its identity types like $a = a$, its identity types like $p \cdot p^{-1} = refl_a$, identity types within that type, and so on. Each forms a groupoid up to higher identities, so in total a type has interacting groupoid structures at each nested level of equalities. We say then that types correspond to 'weak ∞-groupoids.

This is where the topological interpretation arises, as the 'homotopy information' of a topological space is naturally described by a weak ∞-groupoid. Consider a space X (for instance a sphere, or Euclidean 3-d space, or any weird higher dimensional shape). Just like a type, X has elements, called 'points'. And between any two points are (potentially) paths connecting them, which we will take to be analogous to elements of the identity types. Paths are transitive in the sense that a path from point x to y and one from y to z concatenate to get a path from x to z, have inverses by reversing the direction of a path, and reflexive in that any point has a path to itself that just stays in the same place. In our repeated example, for a path p from x to y the path $p \cdot p^{-1}$ which goes from x to y then back the way it came to x is clearly not the same as the path which stays at

x, but $p \cdot p^{-1}$ can be shrunk back along itself from y to x until it is just the point at x. This kind of deformation is called a homotopy from $p \cdot p^{-1}$ to the constant path at x, which acts like a path between the two paths. There can also be homotopies between different homotopies, and between those homotopies, and so on. This associates to X a weak ∞-groupoid and suggests a powerful interpretation of equality in types.

This construction of a 'fundamental ∞-groupoid' of a space can go both ways. Given a type A, its identity types, their identity types, and so on, one can imagine a space with points the elements of A, paths between the appropriate points for each element of an identity type of A, a filled in surface between two paths whenever they are equal in the appropriate identity type, a 3-d filling between two of those for an equality in an identity type of an identity type of A, and so on, constructing a space where paths correspond to equalities in A, paths between paths correspond to equalities between equalities, and so on. This is how types can be thought of as spaces.

As a simple example, the type $\mathbf{1}$ has a single element $*$, a single element $refl_* : * =_{\mathbf{1}} *$, and so on. As a space, $\mathbf{1}$ is the space with a single point. Equalities of the form $refl_*$ are constant paths at $*$, and all higher paths take the same form, so the single point can really be taken as the entire space corresponding to $\mathbf{1}$. More interesting examples will arise in section 6.

Lastly, we have a proposition (easily proven by the induction principle for identity types) that guarantees functions between types behave like continuous functions between spaces, in that they send paths to paths:

Proposition 4. *For types A, B, elements $a, b : A$, equality $p : a =_A b$, and function $f : A \to B$, there is an equality $f(p) : f(a) =_B f(b)$. In other words, functions preserve propositional equality.*

4. Pointed Types

It will be convenient to focus on types which come with a specified base element. Given a universe of types $\mathcal{U}$, it is easy to describe a type of 'pointed types', consisting of all combinations of a type and an element of the type.

Definition 5. The universe of pointed types is the type $\tilde{\mathcal{U}} := \sum_{A \in \mathcal{U}} A$, with elements of the form (A, a) where $a : A$

Now that we have pointed types, it is reasonable to define a pointed type of pointed functions between them.

Definition 6. For pointed types (A, a) and (B, b) define

$$(A, a) \to_* (B, b) := \left(\sum_{f : A \to B} f(a) = b, (\lambda(a : A).b) \right)$$

Given a pointed type, we can consider the identity type at the basepoint, consisting of equalities from the basepoint to itself. In the topological interpretation, these equalities take the form of loops in the space, based at the basepoint.

Definition 7. The loop space $\Omega(A, a) := (a =_A a, refl_a)$, and for $n > 1$ $\Omega^n(A, a) := \Omega(\Omega^{n-1}(A, a))$

The loop space has a natural basepoint given by the constant loop at the basepoint of the space, for types given by $refl_a$. Note that for iterated loop spaces the definition unwraps to something looking like $\Omega^3(A, a) = (refl_{refl_a} = refl_{refl_a}, refl_{refl_{refl_a}})$. I will write $refl_a^n$ for $refl_{refl_a^{n-1}}$ where $refl_a^1 = refl_a$.

5. Equivalence and Univalence

Our main result will be that two different (pointed) types describing the loops in a type are the same. But what does that mean in type theory? Typically two mathematical objects are considered the same (or isomorphic) if there is an invertible function from one to the other. But defining inverse functions requires understanding when a function is 'equal' to the identity, and as of yet we have no type theoretic way of comparing two functions.

Definition 8. For types $A, B : \mathcal{U}$ and functions $f, g : A \to B$, we define the type of 'homotopies' from f to g as $(f \sim g) := \prod_{a:A} f(a) = g(a)$

Type theoretically this looks like the statement that f and g act the same on each element of A up to propositional equality. But as we have discussed propositional equality is a weaker notion than we are used to, so the topological interpretation is helpful here: a homotopy from f to g is a continuous family of paths from $f(a)$ to $g(a)$ for all $a : A$.

Now that we have a notion of similarity for functions, we can define equivalences between types as follows, writing id_A for the identity function on the type A:

Definition 9. A function $f : A \to B$ is an equivalence if there exists a function $g : B \to A$ such that $f \circ g \sim id_B$ and $g \circ f \sim id_A$. The type of equivalences from A to B is then $(A \simeq B) := \sum_{f:A\to B} \sum_{g:B\to A} (f \circ g \sim id_B) \times (g \circ f \sim id_A)$

We now have ways of comparing both types and functions between them. But functions form a type and types belong to the type $\mathcal{U}$, there are already notions of equality between functions and types, namely $f =_{(A\to B)} g$ and $A =_{\mathcal{U}} B$. It is natural to ask when are two types or functions propositionally equal, and are those notions of equality related to those defined above using homotopies. The rules of type theory don't give answers to these questions, but we can demand without inviting any contradictions that functions are equal when they are homotopic and types are equal when they are equivalent, as in the following two axioms:

Axiom 10. *(Function Extensionality) For types A, B and $f, g : A \to B$, $(f = g) \simeq (f \sim g)$*

Axiom 11. *(Univalence) For types $A, B : \mathcal{U}$, $(A =_{\mathcal{U}} B) \simeq (A \simeq B)$*

Function extensionality can be interpreted type theoretically as saying that two functions are the same if they take the same value on each input. The topological interpretation is that the

topology of the function space is such that paths from f to g correspond to homotopies from f to g. Univalence gives us the notion that equivalent objects are in fact equal. Both axioms convey the topological idea that in type theory, two constructions can be considered equal if they differ only by homotopy or equivalence (whereas in topology there are constructions like dimension which are not homotopy invariant: a 2-d plane is homotopy equivalent to a 1-d line). Proving these axioms to be consistent with the rules of type theory is far beyond the scope of this paper, but an encouraging result given in [1] is that Function Extensionality can be derived from Univalence.

We will be primarily concerned with equality of pointed types, which is now easy to define. I will abuse notation by writing $f : A \simeq B$ for an equivalence $f : A \to B$ (technically an element of $A \simeq B$ also includes the inverse function and homotopies witnessing that f is an equivalence).

Proposition 12. *For pointed types* $(A, a), (B, b) : \tilde{\mathcal{U}}$, $\Big((A, a) =_{\tilde{\mathcal{U}}} (B, b)\Big) \simeq \sum_{f:A\simeq B} f(a) = b$

Proving this rigorously involves the idea of 'transfer' discussed in [1] regarding equality in dependent sum types, but given Univalence and our general treatment of pointed types, it seems unnecessary to go into such detail. However, this characterization will turn out to be rather unwieldy for loop spaces and pointed function spaces, so we instead use an equivalent definition of pointed equivalence, which is also equivalent to equality in $\tilde{\mathcal{U}}$:

Definition 13. For pointed types $(A, a), (B, b)$, the type of pointed equivalences is $\Big((A, a) \simeq_* (B, b)\Big) := \sum_{f:(A,a)\to_*(B,b)} \sum_{g:(B,b)\to_*(A,a)} (f \circ g \sim id_B) \times (g \circ f \sim id_A)$

6. Higher Inductive Types

We have thus far considered very much in the abstract the idea that types correspond to spaces, without explicitly describing the correspondence for any interesting examples. Higher inductive types are a means of describing relatively simple types corresponding to simple spaces, by allowing type constructors not just for the elements of a type, but for its identity types as well. This fits well into the topological interpretation of types where a type is considered as the entire nested structure of its identity types, not just as elements with equalities that happen to form another type. While a general set of rules for higher inductive types have not yet been completely developed, the uncontroversial simple examples we focus on illustrate the appeal of defining types this way.

Definition 14. We define the circle S^1 to be the type 'freely generated by' an element $base : S^1$ and an equality $loop : base =_{S^1} base$

How is a type freely generated by an element and an equality, you might ask? This idea is a generalization of the concept of 'inductive types', which use the same language for types freely generated by constructors on the level of elements. For instance, the natural numbers $\mathbb{N}$ are the type generated by an element 0 and a function $succ : \mathbb{N} \to \mathbb{N}$. The type has an element 0, and an element $succ(0)$, and $succ(succ(0))$, and so on, with no assumption of relations between them or additional elements beyond what is required by the rules of type theory (the type could be said to

be 'free' of such impositions). For higher inductive types, the 'rules of type theory' are taken rather explicitly to be the rules defining weak ∞-groupoids, including the group-like properties discussed in section 3 and rules regarding interaction between groupoid structures between levels.

For instance, the groupoid properties assert that there are equalities $refl_{base}, loop^{-1} : base =^1_S base$, but because we are assuming S^1 to be freely generated by *base* and *loop*, we cannot assume that $loop = refl_{base}$ or $loop = loop^{-1}$. The laws of type theory require among other things that $loop \cdot loop^{-1} = refl_{base}$ as discussed in section 3, but if freely adding an element to $base = base$ required that the element be equal to $refl_{base}$ then all such equalities would be equal, and our type theory wouldn't be nearly as interesting.

Intertwined with the notion of S^1 being freely generated by *base* and *loop* is its recursion principle, which asserts that for any type C, a function $f : S^1 \to C$ corresponds precisely to a choice of an element $f(base) : C$ and an equality $f(loop) : f(base) =_C f(base)$. The 'free' aspect of S^1 is expressed by the ability to choose any point in C and any loop at that point as the image of the function.

As the notation suggests, this type is meant to correspond to the circle, with a specified basepoint. The circle can be defined topologically as a point with a single nondegenerate loop, and continuous functions from this circle to other spaces likewise correspond to a choice of a point in the space and a loop at that point. In topology, it is also valuable to consider the higher dimensional loops (resembling higher dimensional spheres) based at a point in the space, which motivates the following types.

Definition 15. For $n > 1$, define the n-sphere type S^n to be freely generated by an element $base : S^n$ and $loop_n : refl^{n-1}_{base} = refl^{n-1}_{base}$

Note here that $loop_n$ lives in the nth level nested identity type of S^n at *base* and its successive deeper $refl$ elements. We can think of constructing this type as a space by starting with the point *base*, observing that there must be an equality $refl_{base}$ corresponding to the constant path at *base*, and thus an equality $refl_{refl_{base}}$ corresponding to the constant 2-d path at *base*, and so on up to level $n-1$, where we add a non-constant n-dimensional path $loop_n$ which can be thought of as an n-dimensional sphere. For those less comfortable with higher dimensional topology, recall that the traditional sphere S^2 is the set of points unit distance from the origin in Euclidean 3-dimensional space. The higher dimensional spheres S^n are defined as the points unit distance from the origin in $(n+1)$-dimensional Euclidean space, here considered up to potential deformations (homotopy).

The recursion principle for S^n is analogous to that of the circle: for any type C a function $f : S^n \to C$ is given by any element $f(base) : C$ and any n-path $f(loop_n) : refl^{n-1}_{f(base)} = refl^{n-1}_{f(base)}$.

While the spheres are valuable type theoretically, as they identify loops in other types by their recursion principles (discussed more below), they also suggest how to define types corresponding to any topological space which can be built from paths and spheres. Just freely add whatever basepoints are needed, as well as the appropriate paths between them, paths between those, and so on. There are sometimes reasons to use clever tricks for defining spaces using only elements and paths (as opposed to higher dimensional paths), but this serves as an effective proof of concept for modeling an important class of topological spaces as types.

7. Equivalence for Loop Spaces

The essence of our main result, stated below, now reduces to a simple argument from merely unwinding previous definitions and axioms.

Theorem 16. $\prod_{A:U} \prod_{a:A} \Omega^n(A,a) = ((S^n, base) \to_* (A,a))$

Proof We will fix an arbitrary pointed type (A,a), so that the proof reduces to constructing an element of $\Omega^n(A,a) = ((S^n, base) \to_* (A,a))$. By our characterization of equality in $\tilde{\mathcal{U}}$, this is given by pointed function

$$\phi : \Omega^n(A,a) \to_* ((S^n, base) \to_* (A,a))$$

a pointed function

$$\psi : ((S^n, base) \to_* (A,a)) \to_* \Omega^n(A,a)$$

and witnesses to $\phi \circ \psi$ and $\psi \circ \phi$ being homotopic to the appropriate identity functions.

First we note that the basepoint of $((S^n, base) \to_* (A,a))$ is the function sending $base$ to a and $loop_n$ to $refl_a^n$ (this is analogous to the constant function on the sphere in topology), and the basepoint of $\Omega^n(A,a)$ is $refl_a^n$.

Defining ϕ is simple by the recursion principle for S^n. $\Omega^n(A,a) = (refl_a^{n-1} = refl_a^{n-1}, refl_a^n)$, so an element $\omega : \Omega^n(A,a)$ along with the basepoint a give a function $f : S^n \to A$ with $f(base) = a$ (so the function is pointed) and $f(loop_n) = \omega$. ϕ then sends ω to f, and is pointed as if ω is $refl_a^n$, f is the constant function by definition.

We define ψ essentially as an evaluation function. A pointed function f from $(S^n, base)$ to (A,a) has $f(base) =_A a$ and $f(loop_n) : refl_{f(base)}^{n-1} = refl_{f(base)}^{n-1} \equiv \Omega^n(A, f(base))$. This is encouraging, but not quite what we want, as $\Omega^n(A, f(base))$ is not the same as $\Omega^n(A,a)$ as a and $f(base)$ are only propositionally equal. However, this corresponds to 'change of base' for higher dimensional loops in topology, or equivalently a weak ∞-groupoid operation caled 'whiskering', which allows one to use the equality $f(base) =_A a$ to get an equivalence between $\Omega^n(A, f(base))$ and $\Omega^n(A,a)$. The details of this construction rely on more intuition from topology or category theory than we wish to develop here. The key point is that a pointed function from S^n identifies, at least approximately, an n-dimensional loop at the basepoint $a : A$, so psi sends the pointed function to that loop in $\Omega^n(A,a)$. ψ is trivially pointed as, by the description above, the constant function sending all of S^n to a identifies the n-loop $refl_a^n$.

It only remains to show that ϕ and ψ are inverses up to homotopy, which I will justify informally. $\phi \circ \psi$ takes a pointed function from S^n to A, identifies in its image an n-loop at a, then takes the function sending $loop_n$ to that n-loop. By the uniqueness aspect of the recursion principle these functions are equal, so $\phi \circ \psi$ is equal to the identity on pointed functions. $\psi \circ \phi$ takes an n-loop at a, constructs by the recursion principle a function from S^n sending $loop_n$ to that n-loop, then takes the image of $loop_n$, which is of course the same n-loop at a, so $\psi \circ \phi$ is equal to the identity on n-loops at a. This completes the proof. □

Aside from hand waving over a few technical points, this proof offered nothing particularly new: this equivalence follows essentially from our definitions. In topology this equivalence is slightly

less obvious, , but the points of topological interest are more evident after passing to the connected components of these spaces, which we now describe.

8. Truncation to Set Structures

As discussed in the introduction, this equivalence of definitions is meant to apply to homotopy groups, which are meant to be sets. Type theory is often touted as being a supertheory of set theory in that it can describe sets, which in short goes as follows:

Definition 17. A 'set' is a type A such that all equalities between any pair of elements are equal. As a proposition, $isSet(A) := \prod_{a,b:A} \prod_{p,q:a=b} p = q$

This condition can be phrased topologically as the statement that the equality types are 'contractible spaces' (this analogy is subtle though and relies on function extensionality) in which all elements are uniformly equal to each other and the type is equivalent to $\mathbf{1}$ if nonempty. However, this notion of a set differs from the classical notion in that there can be explicitly different elements which are still equal to each other. The classical elements of the set are then more like the 'connected components' of the type, or in other words the equivalence classes of elements under propositional equality, and the condition for a type to be a set is that each such equivalence class is equivalent to $\mathbf{1}$ (so up to equivalence it looks like a set or 'discrete space').

Given a type A, we can form a set from it by freely adding equalities of equalities to make each connected component contractible. The resulting set is then analogous in topology to the set of connected components of the corresponding space.

Definition 18. For a type A, the 'set truncation' $||A||_0$ of A is the type specified (inductively generated) by a function $A \to ||A||_0$ and a witness of $isSet(||A||_0) \equiv \prod_{a,b:||A||_0} \prod_{p,q:a=b} p = q$

Now to show our result for homotopy groups, we have to take the set truncations of both definitions. But as they have already been proven equal as types, it is reasonable to think that they will remain equal after this construction.

Lemma 19. *For types A and B, if $A = B$ then $||A||_0 = ||B||_0$*

Proof Informally, the equality $A = B$ gives an equivalence between A and B, and as functions respect equality that equivalence extends to one from $||A||_0$ to $||B||_0$. □

We have thus proven the following:

Theorem 20. $\prod_{A:U} \prod_{a:A} ||\Omega^n(A, a)||_0 = ||((S^n, base) \to_* (A, a))||_0$

This result is an example of the power of axioms like Function Extensionality in the homotopy interpretation. In classical homotopy theory, the nth fold loop space and the space of functions from the n-sphere are (nearly) the same from the definitions, as they are here, but passing to the

homotopy groups in each case is done differently. The nth homotopy group $\pi_n(X, x)$ of a pointed space (X, x) is defined as the set of connected components of the loop space $\Omega^n(X, x)$, as we have done here. But in the alternate definition the group is defined as the set of homotopy classes of pointed maps from the n-sphere into X, and homotopies need not correspond to paths in the function space. Thus Function Extensionality gives us a powerful tool in type theory with nontrivial topological significance.

9. Group Structures

We have now established that two competing definitions of the nth homotopy group of a type give equivalent sets. However, we have done nothing to suggest that they are isomorphic as groups, or given any attention to what their group structures are. I will describe informally the group-like operations on the un-truncated types, which are group operations only up to propositional equality, precisely the condition for being a group in the set truncated type on the nose (at least in the sense of the classical set corresponding to the truncated type).

The group-like structure on $\Omega^n(A, a)$ comes from the concatenation of equalities arising from transitivity in section 3. Any two loops, as they witness the equality of a or some $refl_a^{n-1}$ with itself, can be composed by transitivity, and as we discussed in section 3 this operation forms a group up to propositional equality.

The group multiplication on the function type $S^n \to_* A$ (with basepoints omitted for brevity) ought to have type signature $(S^n \to_* A) \times (S^n \to_* A) \to (S^n \to_* A)$. But the type $(S^n \to_* A) \times (S^n \to_* A)$ is equivalent to the type $S^n \vee S^n \to_* A$, where $S^n \vee S^n$ is the type generated by the element $base$ and two n-loops $loop1_n$ and $loop2_n$. It is easy to see that these two types are equivalent, as a pair of pointed maps from the n-sphere into A both send the basepoint $base$ to the basepoint of A, and each identify an n-loop at the basepoint in A. A map from $S^n \vee S^n$ to A also does just that, sending $base$ to the basepoint of A and $loop1_n$ and $loop2_n$ each to an n-loop.

To get the multiplication on $S^n \to_* A$, we first consider the map $\gamma : S^n \to_* S^n \vee S^n$ with $\gamma(base) = base$ and $\gamma(loop_n) = loop1_n \cdot loop2_n$. We now have, for any map $f : S^n \vee S^n \to_* A$, the map $f \circ \gamma : S^n \to_* A$. This gives us the desired multiplication $(S^n \to_* A) \times (S^n \to_* A) \to (S^n \to_* A)$ by considering a pair of maps $S^n \to_* A$ as a map $S^n \vee S^n \to A$ and precomposing with γ. For the rest of the group definition, take the identity to be the constant map $S^n \to_* A$, and inverses by precomposing with the map $S^n \to_* S^n$ sending $base$ to $base$ and $loop_n$ to $loop_n^{-1}$. That this forms a group up to propositional equality follows from the 'cogroup-like-ness' of S^n in that it satisfies properties dual to that of a group.

Now to show that these two group structures are the same, let ω_1 and ω_2 be n-loops in A and f_1 and f_2 be the corresponding pointed functions from S^n into A. The function $f_1 \vee f_2 : S^n \vee S^n \to_* A$ sends $loop1_n$ and $loop2_n$ to ω_1 and ω_2 respectively. The composition $(f_1 \vee f_2) \circ \gamma : S^n \to_* A$ sends $loop_n$ by γ to $loop1_n \cdot loop2_n$ which is sent by $(f_1 \vee f_2)$ to $\omega_1 \cdot \omega_2$ as continuous functions preserve concatenation of paths. But $\omega_1 \cdot \omega_2$ is the product of ω_1 and ω_2 in the group structure on Ω^n. Therefore our equivalence preserves the group structure!

10. Conclusion

I have shown, somewhat informally, that a classical nontrivial equivalence of different definitions for the homotopy groups of topological spaces holds as well for the analogous definitions of the homotopy groups of types. Furthermore, the axioms generally used in homotopy type theory make this equivalence hold more directly by definition than in topology. Representing a type (or more general mathematical object) as a function space is a powerful idea in math (coming from category theory) called representability, and homotopy type theory makes these representations clear in its definitions of higher inductive types. In addition to introducing an interesting idea from topology in the language of type theory, I hope that this paper has been a concise but instructive summary of many key ideas from homotopy type theory.

References

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