Identity in Homotopy Type Theory, Part I: The Justification of Path Induction

Friday 24th October, 2014

Abstract

Homotopy type theory (HoTT) is a new branch of mathematics that connects algebraic topology with logic and computer science, and which has been proposed as a new language and conceptual framework for mathematical practice. Much of the power of HoTT lies in the correspondence between the formal type theory and ideas from homotopy theory, in particular the interpretation of types, tokens, and equalities as (respectively) spaces, points, and paths. Fundamental to the use of identity and equality in HoTT is the powerful proof technique of *path induction*. In the 'HoTT Book' [1] this principle is justified through the homotopy interpretation of type theory, by treating identifications as paths and the induction step as a homotopy between paths. This is incompatible with HoTT being an autonomous foundation for mathematics, since any such foundation must be able to justify its principles without recourse to existing areas of mathematics. In this paper it is shown that path induction can be motivated from pre-mathematical considerations, and in particular without recourse to homotopy theory. This makes HoTT a candidate for being an autonomous foundation for mathematics.

Contents

1 Introduction

Homotopy Type Theory (HoTT) [1] is a new branch of mathematics that bridges topology, logic, computer science, and category theory. It tightly connects the hitherto disparate areas of homotopy theory in algebraic topology and type theory in logic and computer science, and provides a way to deal with higher-order equivalences that appear to be intractable in category theory. HoTT has also attracted considerable excitement because it facilitates automatic proof verification [1, p. 10]. HoTT is based on constructive intensional type theory due to Martin-Löf ([2]) and hence is very different in character from an extensional theory such as set theory. We give a brief exposition of some of the essential features of HoTT in Section 2 below.[1]

[1] Note that the subtitle of the HoTT Book is 'Univalent Foundations of Mathematics'. This refers to the 'Univalence Axiom' introduced by Voevodsky, which is one of the primary innovations of the Homotopy Type Theory research programme. While this axiom is an important idea in HoTT, and has major implications for the understanding of identity in HoTT (see [3]), it is beyond the scope of the present paper. In particular, we focus on the basic language of HoTT, which does not include Univalence. We will explore issues relating to Univalence in a subsequent paper.

As well being a new branch of mathematics, HoTT is proposed to be a new foundation for the rest of mathematics. However, the word 'foundation' is often used in two different senses: a weaker sense, commonly used by mathematicians, which we may gloss as a 'framework' or 'language' for mathematical practice; and a stronger sense, more often used by philosophers of mathematics, which takes more seriously the analogy with the foundations of a building [4] and requires that the system be *autonomous* in something like the sense of [5].

The former sense, that of a framework in which to do mathematics, consists of some mathematical entities that are considered elementary and some rules for manipulating them. These resources are then used to define all other mathematical entities and to prove any relevant theorems by application of the rules. Such a framework involves a language in which the entities of the framework can be described and into which, given the rules of the framework, all of mathematics can be translated. A standard example of this is ZFC set theory, in which the basic entities are sets (and perhaps some *urelements*), and the rules are those given by the ZFC axioms. The rest of mathematics can then be built up from this starting point – ordered pairs are defined as certain kinds of sets, relations are defined as sets of ordered pairs, functions as relations satisfying certain conditions, and so on. The language has the symbols for urelments (if any), and the symbols for the membership relation and sets plus the background logical language.

Similarly, HoTT is intended to provide a formalism in which all mathematical entities can be described in the language of tokens and types, and in which the theorems of mathematics can be proved by application of rules that embody a form of constructive logic. In this paper we do not address the question of whether HoTT provides a foundation for mathematics in this sense, and defer to the discussion in the HoTT Book.[2]

For the purposes of giving a foundation in the sense of a framework, the entities and the rules governing them can be anything we like, so long as they do indeed allow us to reconstruct the existing objects and proofs of mathematics. Thus, demonstrating that a proposed system qualifies as a foundation (in this sense) is merely a formal exercise in spelling out how exactly some already-recognised foundational system can be encoded in the proposed language. In particular, there is no requirement to answer ontological questions justifying that the proposed foundational entities exist, nor epistemological questions explaining how we come to know about these entities and their properties, nor to give motivation or justification for the particular rules that have been chosen.

This is not the case with the second sense of 'foundation' of mathematics, which

[2] In particular see [1, Chapter 10] on the reconstruction of set theory in HoTT.

is the one more usually of interest to philosophers. A foundation in this second sense, which Linnebo and Pettigrew [5] call an "autonomous" foundation, provides a conceptual and epistemological basis for mathematics. If we have such a foundation, it must be possible to take any mathematical concept, expressed in the formalisation, and explain it via simpler and simpler concepts, until eventually arriving at concepts that are so simple that they require no pre-existing mathematical comprehension to understand them [4]. Likewise it must be possible to take any mathematical proof and break it down into steps whose justification requires no pre-existing mathematical understanding. ZFC set theory is generally taken to provide a foundation for mathematics in this stronger sense, because the notion of 'a collection of entities' is suitably primitive to require no mathematical explanation.[3]

An autonomous foundation must therefore use only concepts that can be pre-mathematically understood, and rules that can be pre-mathematically motivated. If any aspect of a purported foundation for mathematics relies for its formulation or justification upon some advanced area of mathematics then it cannot be a foundation for mathematics in this second sense. However, it may not be obvious whether a particular collection of concepts and the justifications for the corresponding rules can in fact be taken to be entirely pre-mathematical. A particular presentation of a proposed foundation may fail to be autonomous, but this of course does not mean that the system itself is not autonomous, since it may be possible to give an alternative autonomous presentation of its entities and basic rules.

The presentation of HoTT given in the HoTT Book freely makes use of the analogy between types and spaces (where the latter are understood as in homotopy theory), and between identifications and paths.[4] This manner of thinking is pervasive throughout the HoTT Book from the Introduction onwards, since this is the aspect of the theory that the authors wish to emphasise: they are interested in presenting a framework for mathematics (i.e. a foundation in the first sense) with radically novel connections to the sophisticated and powerful ideas from homotopy theory. However, this manner of presentation obscures the question of whether HoTT can form a foundation for mathematics in the second, stronger sense. To answer this question it is necessary to give a different presentation of the concepts and rules of HoTT, explaining and justifying them in a way that does not depend upon homotopy theory or any other pre-existing mathematics. In Section 2 we give a summary of how this can be done for the majority of the basic notions of HoTT. (This presentation is explained in considerably more detail in Part I of [7].)

[3] However, when we allow the collections to be *infinite* then we are quickly carried away from the primitive understanding we began with, and the intuitive justification for the axioms of ZFC may be called into question. [6]

[4] For more details of this, see Section 5.

A particular sticking point in giving a pre-mathematical account of HoTT is explaining and justifying the principle of *path induction*, which is central to the handling of identity (or equality) in HoTT. In the presentation in the HoTT Book identifications such as $a = b$ are interpreted as paths in (homotopy) spaces, and the explanation and justification of path induction depends upon intuitions arising from homotopy theory – in particular, that paths can be deformed and retracted without changing their essential characteristics. (This is explained in more detail in Section 5.)

If this were the only way of justifying path induction then this would of course undermine any claims for HoTT as an autonomous foundation for mathematics (although, of course, its status as a foundation of the first kind would not be impugned). In this paper we show that the homotopy interpretation is not necessary for the justification of path induction by giving a justification for this principle that depends only upon pre-mathematical ideas. It is important to note that the elimination rule for identity types that we are calling (following the HoTT Book) 'path induction' is part of the original formulation of Martin-Löf type theory [2]. Martin-Löf's motivation for this principle of course owes nothing to the homotopy interpretation which was introduced several decades later by Awodey and Warren [8]. There is a wealth of literature that discusses other aspects of the justification of Martin-Löf's work. However, we have been unable to find the kind of elementary justification of the elimination rule for identity, based on pre-mathematical ideas about identity and the basic features of the type theory, that we provide here.

In Section 2 we give a brief overview of some of the essential ideas of HoTT, up to but not including identity. Section 3 describes identity in HoTT, and states the principle of path induction without giving a justification, while Section 4 considers the general form that a justification for path induction would have to take. Section 5 sketches the basic ideas of homotopy theory, and outlines the argument given in the HoTT Book for path induction using these ideas. The positive contribution of this paper is in Section 6 where we give a justification of path induction that relies only upon pre-mathematical ideas, without reference to homotopy or to any other pre-existing mathematics.

2 An overview of HoTT (without homotopy or identity)

2.1 Main features

As a type theory, the basic elements of HoTT are *tokens* and *types*. In HoTT there is a type associated with each mathematical proposition that can be expressed in the language, and we think of the tokens of a type as 'certificates' to the truth of the corresponding proposition.[5] Each token belongs to exactly one type. A given type may have no tokens (i.e. it may be 'uninhabited') if it corresponds to a false proposition, but an inhabited type may have multiple distinct tokens. We write '`x` : `A`' to denote that `x` is a token of type `A`.

We can also define types that are more usefully thought of as mathematical objects, such as the type $\mathbb{N}$ each of whose tokens corresponds to a natural number. HoTT therefore does not make a sharp distinction between mathematical objects and mathematical propositions: both are treated on an equal footing.

The types in HoTT are treated *intensionally* rather than *extensionally*. Practically this means that we should think of types as being distinguished by the descriptions that define them, rather than by their contents. Thus, for example, the types 'positive integer less than 3' and 'integer exponent n for which $a^n + b^n = c^n$ has a solution in the positive integers' are fundamentally distinct types, even though it can be proved that they are *extensionally* equal. This extends to empty types as well: the type 'even divisor of 9' is a distinct type from 'even divisor of 11', even though both types are uninhabited. This has the advantage that the basic elements of the theory are closer to the descriptions of mathematical entities and the mathematical concepts that are directly used in practice.

The distinction between token and type should *not* be taken to be a 'linguistic vs. semantic' distinction. In some literature on type theory 'token' is used to refer to syntax, but this is not what is done in the HoTT Book, where the word 'token' is used interchangeably with words such as 'object', 'point', and 'element'.[6] Of course we must have *expressions* in a formal language, which serve as the names of tokens and types. The rules for the use of HoTT are most directly expressed as manipulations of expressions, but they can just as well be thought of as ma-

[5] tokens are sometimes called 'witnesses' or 'proofs' to their corresponding propositions, but we prefer to avoid this terminology, for reasons explained in [7].

[6] We avoid this terminology: 'object' begs the question of how tokens are to be interpreted, likewise 'point' implies a spatial interpretation of types, and 'element' is too reminiscent of set theory. Rather than introducing a brand new word, such as 'item', we retain the usual word 'token', but do not assume in advance any particular interpretation of what tokens are.

nipulations of tokens and types directly, and this is generally more convenient. The type formation rules allow us to produce new types from old ones, and correspondingly produce tokens of the new types when given tokens of the old ones. The consistency of the basic theory ([2], [1, Appendix A]) guarantees that if we begin with expressions that name tokens or types then the expressions produced will also name tokens or types – no application of the rules can produce an empty name, unless we begin with empty or contradictory expressions.[7]

A proof in HoTT therefore consists of a sequence of applications of these rules, beginning with tokens of the given premises and ending with a token of the conclusion. Thus the logic of HoTT is *constructive*. However, unlike the situation in ZFC set theory, in which we must first define first-order logic and then use this to set out the axioms of set theory on top of that, in HoTT the logic emerges from the basic notion of types as propositions and tokens of a type as certificates to that proposition, and is incorporated directly into the rules for manipulating tokens and types. (For a more extensive discussion of this see [9].)

2.2 Logic and the rules for manipulating types

In this section we briefly outline the basic language of HoTT and the rules for manipulating tokens and types, with the exception of identity (or equality), which is introduced in Section 3. (For a more detailed exposition see [1, Chapter 1] or [7].)

Functions in HoTT are defined by *lambda abstraction*: given an expression Φ naming a token of type $\mathtt{B}$, possibly containing one or more instances of a variable $\mathtt{x}$ stipulated to be of type $\mathtt{A}$, we get a function $[\mathtt{x} \mapsto \Phi]$ (more traditionally written '$\lambda\mathtt{x}.\Phi$') of type $\mathtt{A} \to \mathtt{B}$. Evaluation of such a function with an argument $\mathtt{y}$ of type $\mathtt{A}$ is given by substituting the expression naming $\mathtt{y}$ for each instance of $\mathtt{x}$ in Φ (with renaming to avoid collisions, as usual), thus producing an expression of type $\mathtt{B}$.[8]

For any types $\mathtt{A}$ and $\mathtt{B}$ there is a *function type* $\mathtt{A} \to \mathtt{B}$ whose tokens are functions defined as above. Given a token of $\mathtt{A} \to \mathtt{B}$ and a token of $\mathtt{A}$ we can combine them

[7] This way of thinking is not explicit in the HoTT Book, but we recommend it for reasons explained in [7]. Note that the consistency proof cited above is for the basic theory, but has not yet been extended to cover the additions made in the HoTT Book such as 'higher inductive types' and the 'Univalence Axiom'. The only proofs of consistency for the expanded system are relative to ZFC and large cardinals. (We thank Steve Awodey for drawing our attention to this point.)

[8] Although this idea derives from lambda calculus, the basic notion of substitution in an expression is a simple pre-mathematical one, familiar to anyone who, for example, is able to use pronouns in natural language. We therefore do not consider this to be an obstruction to the autonomous status of HoTT.

to produce a token of B. Since tokens of types are understood as certificates to the truth of their corresponding propositions, this suggests that the proposition corresponding to the function type $\mathtt{A} \to \mathtt{B}$ is the implication $A \Rightarrow B$. This parallel between functions in type theory and implications in logic is the first step of the Curry-Howard correspondence.

The other basic logical operations – conjunction, disjunction, and negation – can all be interpreted in type theory as well:

- A certificate to the truth of a *conjunction* of two propositions is just a pair of certificates to those two respective propositions. Given $\mathtt{a} : \mathtt{A}$ and $\mathtt{b} : \mathtt{B}$, we write '$(\mathtt{a}, \mathtt{b})$' for the pair of these tokens. The type corresponding to the conjunction, having these pairs as its tokens, is written '$\mathtt{A} \times \mathtt{B}$', and is called the *product* of A and B.

- A certificate to the truth of a *disjunction* of two propositions A and B is a token that is either a certificate to A or a certificate to B. Since every token belongs to exactly one type, the tokens $\mathtt{a} : \mathtt{A}$ and $\mathtt{b} : \mathtt{B}$ cannot also belong to this new type. We therefore formally introduce *counterparts* to these tokens, written as '$\mathtt{inl}(\mathtt{a})$' and '$\mathtt{inr}(\mathtt{b})$'. The type corresponding to the disjunction, to which these tokens belong, is written '$\mathtt{A} + \mathtt{B}$' and is called the *coproduct* of A and B.

- A certificate to the truth of the *negation* of a proposition is something that, if combined with a certificate to the proposition itself, would give a contradiction. Since there cannot be a witness to the truth of a contradiction, the negation of proposition P therefore corresponds to a function $\mathtt{P} \to \mathtt{0}$, where 0 is a type that by definition has no token constructors.[9]

Just as 0 corresponds to a proposition that is by definition false, we also introduce a type 1 corresponding to a proposition that is by definition true. This Unit type is defined to have exactly one token, denoted $*$.[10]

We can also define types that correspond to (bounded) *quantified* propositions, i.e. statements saying that that every token of a given type satisfies some condition, or that there exists a token of a given type that satisfies some condition. Since the logic of HoTT is constructive, these quantifiers must be interpreted constructively. This means that we can only assert that something exists if we have a method of producing it.

[9] Consistency of the system then consists in the claim that 0 has no tokens.
[10] Here we set aside a detail that we examine more carefully in Section 6.1.

- A certificate to the statement that all tokens of type `A` satisfy some condition is something that provides, for each `x : A`, a certificate to the fact that `x` does indeed satisfy that condition, i.e. a function that takes tokens of `A` as inputs and returns these certificates as outputs.

- A certificate to the statement that there exists a token of type `A` that satisfies some condition consists of a pair $(\mathtt{x}, \mathtt{c})$, where `x` is a token of `A` and `c` is a certificate to the fact that this particular `x` satisfies the condition.

The statement that some particular `x : A` satisfies a particular condition is a proposition, and so corresponds to a type in the theory. The statement that some other token `y` satisfies the condition is a different proposition, and so has a different corresponding type. For any condition on `A` we therefore have a *family* of types, indexed by the tokens of `A`, each saying of some token of `A` that it satisfies the condition. We can think of this as a function taking tokens of `A` as input and returning a *type* as output.[11] We call such functions *predicates*.

We can now interpret quantified statements in HoTT. In order to do this we introduce two new ways of forming types, generalisations of the function and product types described above.

- Given a predicate `P` on `A`, a *dependent function* is a function whose output type is not fixed in advance, but depends upon the input token: when given `x : A` it returns a token of $\mathtt{P}(\mathtt{x})$, when given `y : A` it returns a token of $\mathtt{P}(\mathtt{y})$, and so on. We write the type to which these dependent functions belong as '$\prod_{\mathtt{a:A}} \mathtt{P}(\mathtt{a})$' to emphasise its correspondence with the universally quantified statement that all tokens of `A` satisfy predicate `P`. Alternatively, we may write the type as '$\langle \mathtt{a} : \mathtt{A} \rangle \to \mathtt{P}(\mathtt{a})$' to emphasise its similarity to the (non-dependent) function type.

- Given a predicate `P` on `A`, a *dependent pair* is a pair the type of whose second component is not fixed in advance, but depends upon the type of its first component: if the first component is `x : A` then the second component is a token of type $\mathtt{P}(\mathtt{x})$, if the first component is `y : A` then the second component is a token of type $\mathtt{P}(\mathtt{y})$, and so on. We write the type to which these dependent pairs belong as '$\sum_{\mathtt{a:A}} \mathtt{P}(\mathtt{a})$' to emphasise its correspondence with the existentially quantified statement that there exists a token of `A` that

[11] Compare with the definition of functions above, where we said that a function returns a *token* of its output type. This way of thinking therefore requires the introduction of a higher-order type, called `TYPE`, that has other types as its tokens. This must be done with care to avoid paradox – in particular we must disallow `TYPE` from being a token of itself. For more details see [1, Section 1.3] or [7].

satisfies predicate $\mathtt{P}$. Alternatively, we may write the type as '$\langle \mathtt{a} : \mathtt{A} \rangle \times \mathtt{P}(\mathtt{a})$' to emphasise its similarity to the (non-dependent) product type.

Dependent pair types are useful in two different ways. The first is the reading as an existentially quantified proposition as described above. We can also think of a dependent pair type as a way of forming a *subtype* by imposing a predicate: the type $\sum_{\mathtt{a}:\mathtt{A}} \mathtt{P}(\mathtt{a})$ consists of those tokens of $\mathtt{A}$ that satisfy the predicate $\mathtt{P}$, each accompanied by a certificate to that fact.

These are the type formers and token constructors for the basic vocabulary of HoTT (aside from identity). But to give the definition of a type it is not sufficient merely to say how the type and its tokens are produced: we must also specify how they can be used. Functions themselves (i.e. tokens of function types $\mathtt{A} \to \mathtt{B}$) are used by applying them to arguments of their input types, or by composing them with other functions to give new functions. For the other types above we specify how they are used by giving an *elimination rule* for the type, i.e. the method for defining functions from that type to an arbitrary other type.[12]

The language described so far is sufficient to allow proofs in (constructive) predicate logic to be carried out, and enables us to define and introduce new types as needed and to prove theorems about them. In summary, to define a new type we must specify:

- a type former that gives the name of the new type, given the names of the input types and tokens (if any);
- one or more token constructors: functions that output tokens of the new type;
- an elimination rule: a method for using tokens of the type, such as rules for constructing functions from the new type to an arbitrary other type.

The type theory described above is not the whole of HoTT since it is missing an essential component: it has no way of asserting that two things are equal or identical.[13] (We discuss the definition of this remaining component, called *identity types*, in the next section.) However, the theory described so far provides a language that is sufficiently powerful for us to define many important types used

[12] The elimination rules for the types described above are fairly obvious, and won't be needed in what follows here. For details see [1] or [7].

[13] There are other features of the full theory of HoTT missing as well, such as Univalence and function extensionality. As noted in Section 1, we will not examine these in this paper as they are not basic features of the language of the theory.

in ordinary mathematics. For example, we can introduce a type $\mathbb{N}$ whose tokens correspond to natural numbers.

Call the theory defined so far HoTT$^-$. Note that none of the definitions or rules in HoTT$^-$ depend upon any sophisticated domain of mathematics such as homotopy theory: they were all derived from considerations of elementary pre-mathematical notions, such as substitution and conjunction. Everything in HoTT$^-$ is therefore eligible to form part of an autonomous foundation for mathematics.[14]

3 Identity types and path induction

HoTT$^-$ is quite powerful, but without identity types many elementary mathematical truths cannot be expressed, because identity types are required to represent equality in mathematical theories. Without a notion of equality we cannot do even the most basic number theory because there are no equations, and we can't even state (let alone prove) that, for example, the sum of the natural numbers up to n is $n(n-1)/2$. Furthermore, we cannot prove (or even state) elementary properties about the things we can define in HoTT$^-$, for example that two alternative definitions of a function give the same output for all inputs. Equality is obviously an essential component of a language that claims to serve as a foundation for mathematics.

Since a statement of equality between two tokens is a proposition, it corresponds to a type. HoTT$^-$ does not enable us to form a type corresponding to this proposition, so we must supplement the language by introducing *identity types*. In the remainder of this section we introduce and explain the type former and token constructor for the identity type, and then state the elimination rule without providing motivation or justification for it. In the following sections we give two different justifications for this rule: the first from the HoTT Book, using homotopy theory, and then our own justification avoiding the using of sophisticated mathematical ideas.[15]

For any two tokens `a` : `A` and `b` : `A` of the same type we can state the proposition that `a` and `b` are equal. tokens that are not of the same type cannot be equal. Thus the type former for the identity type must take as inputs a type `A` and two tokens `a` : `A` and `b` : `A`. The type corresponding to the proposition that `a` : `A` and `b` : `A`

[14] We examine the autonomy of HoTT$^-$ and related issues in a companion paper [10].

[15] An alternative justification for path induction via the Yoneda lemma from category theory is hinted at in the HoTT Book [1, p. 216]. But such a justification would be of no help in establishing HoTT as an autonomous foundation since it requires the machinery of category theory, so we will not examine it further.

are equal is written as '$\mathtt{Id_A(a,b)}$', or sometimes as '$\mathtt{a} =_{\mathtt{A}} \mathtt{b}$'. A token of $\mathtt{Id_A(a,b)}$ is a certificate to the proposition that the tokens $\mathtt{a}$ and $\mathtt{b}$ are equal. We call such a token an *identification* of $\mathtt{a}$ and $\mathtt{b}$. As with any type in the theory, in general there may be multiple tokens of $\mathtt{Id_A(a,b)}$, or just one, or none at all. That is, we do not consider the options 'equal' or 'not equal' to exhaust the possibilities, but rather there may be multiple different identifications of two tokens.[16]

The token constructor for the identity type provides the identifications that should be included as part of the *definition* of identity. It's clear that we shouldn't just be able to freely create tokens of $\mathtt{Id_A(a,b)}$ for arbitrary $\mathtt{a}$ and $\mathtt{b}$, since this corresponds to proving that any two tokens of a type are equal (for example, that two arbitrary natural numbers are equal). The only identifications that are guaranteed to exist (with no further assumptions or premises) are the *trivial self-identifications*. That is, for any type $\mathtt{A}$ we must have a certificate to the proposition that each token of $\mathtt{A}$ is self identical – i.e. that identity is *reflexive*. The token constructor for the identity type is therefore a function that gives us, for any type $\mathtt{A}$ and any token $\mathtt{x} : \mathtt{A}$, a token of $\mathtt{Id_A(x,x)}$, which we write as $\mathtt{refl_x}$.

Since the token constructor for the identity type allows us to construct one certificate to $\mathtt{Id_A(x,x)}$ for any $\mathtt{x} : \mathtt{A}$, and nothing else, the identity type may appear to be of little use. After all, what good is an equality sign if we can only assert things of the form $a = a$ and $x = x$, etc.? However, the situation is more subtle than this.

3.1 Path induction and based path induction

The elimination rule for the identity type is called *path induction*.[17] The essential idea of path induction is as follows: to prove that a property holds for *all* identifications between tokens in some $\mathtt{A}$ it suffices to show that the property holds for all trivial self-identities $\mathtt{refl_a}$.

For comparison, consider the general form of inductive proofs on the natural numbers. Here, to prove that a property holds of all numbers we must prove that it holds for the base case, 0, and then prove that if it holds for a given number n then it also holds for the successor $n + 1$.

The situation with identity types is slightly different. Here, the principle of path induction takes the place of the inductive step. That is, we establish just once, for

[16] This is the essence of HoTT's being an *intensional* type theory. If we added a rule eliminating the possibility of multiple identifications we would reduce HoTT to an *extensional* type theory. See [1, p. 71, p. 128] for further details.

[17] While the elimination rule was part of Martin-Löf's intensional type theory [2], this name for it comes from the homotopy interpretation, see Section 5.

all properties, that if the base case satisfies that property then all identifications do.[18] Thus we are left to prove just the base case (i.e. that the property holds for all trivial self-identifications $\mathtt{refl}_{\mathtt{x}}$).

To understand this better we now give a more formal statement of the principle. We begin by setting out the definition of a variant called *based path induction*, which is equivalent to path induction ([11], [1, Section 1.12.2]) but a little easier to explain.[19]

3.1.1 Based path induction

We can form an identity type $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$ for any pair of tokens $\mathtt{a}$ and $\mathtt{x}$ of a given type $\mathtt{A}$.[20] Thus for any $\mathtt{a} : \mathtt{A}$ we can define a predicate $\mathtt{Id}_{\mathtt{A}}[\mathtt{a}]$ that, when given a token $\mathtt{b}$, returns the identity type $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$. That is, $\mathtt{Id}_{\mathtt{A}}[\mathtt{a}]$ is the 'equal to $\mathtt{a}$' predicate.

Fixing a particular $\mathtt{a} : \mathtt{A}$, we can use the predicate $\mathtt{Id}_{\mathtt{A}}[\mathtt{a}]$ to form the dependent pair type $\sum_{\mathtt{x}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$, which we call the *based identity type*. As a dependent pair type, the tokens of this type are pairs $(\mathtt{b},\mathtt{p})$, where $\mathtt{b}$ is a token of $\mathtt{A}$ and $\mathtt{p}$ is a token of $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$ (i.e. an identification of $\mathtt{a}$ with that particular $\mathtt{b}$). This type corresponds to the proposition 'there exists a token of $\mathtt{A}$ that is equal to $\mathtt{a}$'. This proposition is of course true, since identity is reflexive, and so the token $(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$ is guaranteed to exist. But there may in general be many other tokens as well.

Now consider a predicate $\mathtt{Y}$ on the based identity type, which is a function that takes a pair $(\mathtt{b},\mathtt{p})$ and returns a type involving $\mathtt{b}$ and $\mathtt{p}$ (corresponding to a proposition about $\mathtt{p}$). Such a predicate therefore corresponds to a property that identifications involving $\mathtt{a}$ may or may not satisfy.

Based path induction says that if we have a token $\mathtt{y}$ of $\mathtt{Y}(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$ – i.e. a certificate to the fact that $\mathtt{refl}_{\mathtt{a}}$ satisfies the property $\mathtt{Y}$ – then we can produce a certificate to $\mathtt{Y}(\mathtt{b},\mathtt{p})$ for every token $(\mathtt{b},\mathtt{p})$ of $\sum_{\mathtt{x}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$. More specifically, it says that given $\mathtt{y} : \mathtt{Y}(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$ we have a function that takes a pair $(\mathtt{b},\mathtt{p})$ as input and returns a token of $\mathtt{Y}(\mathtt{b},\mathtt{p})$ as output, in particular giving $\mathtt{y}$ itself when given $(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$ as input.

[18] In the homotopy interpretation the inductive nature of path induction is more apparent (see Section 5).

[19] Henceforth we will sometimes omit 'based' where the distinction is not important.

[20] Recall the distinction between forming the *type* $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$, which corresponds to forming the proposition that $\mathtt{a}$ and $\mathtt{x}$ are equal (which we can do for any pair of tokens), and forming a *token* of $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$, which corresponds to proving this proposition.

3.1.2 Path induction

The definition of path induction is similar. We begin by defining the *total identity type*, $\sum_{\mathtt{s},\mathtt{t}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{s},\mathtt{t})$. Whereas the based identity type fixed one token of $\mathtt{A}$ and let the other range over all values in $\mathtt{A}$, the total identity type lets both variables vary across $\mathtt{A}$. Its tokens are therefore triples $(\mathtt{x},\mathtt{y},\mathtt{p})$, where $\mathtt{p}$ is an identification between $\mathtt{x}$ and $\mathtt{y}$. In the based identity type the reflexivity of identity provided the single distinguished token $(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$. In the total identity type reflexivity gives a token $(\mathtt{x},\mathtt{x},\mathtt{refl}_{\mathtt{x}})$ for each $\mathtt{x} : \mathtt{A}$.

Now consider a predicate $\mathtt{Z}$ on the total identity type, which corresponds to a property that arbitrary identifications between tokens of $\mathtt{A}$ may or may not satisfy. Path induction says that if we have a certificate to the fact that every trivial self-identification $\mathtt{refl}_{\mathtt{x}}$ satisfies the property $\mathtt{Z}$, which is given by a token $\mathtt{z}$ of

$$\prod_{\mathtt{x}:\mathtt{A}} \mathtt{Z}(\mathtt{x},\mathtt{x},\mathtt{refl}_{\mathtt{x}})$$

then we can produce a certificate to $\mathtt{Z}(\mathtt{a},\mathtt{b},\mathtt{p})$ for every $(\mathtt{a},\mathtt{b},\mathtt{p})$. Specifically, it says that given $\mathtt{z}$ we have a function that takes a triple $(\mathtt{a},\mathtt{b},\mathtt{p})$ as input and returns a token of $\mathtt{Z}(\mathtt{a},\mathtt{b},\mathtt{p})$ as output, in particular giving $\mathtt{z}$ itself when given $(\mathtt{a},\mathtt{a},\mathtt{refl}_{\mathtt{a}})$ as input.

4 What would a justification of path induction look like?

How is the principle of (based) path induction to be justified? In more familiar inductive proofs, such as induction over the natural numbers (i.e. proving $P(n)$ for all natural numbers n, for some property P) we must prove both a base case $P(0)$ and an inductive step $P(n) \Rightarrow P(n+1)$. However, the principle of path induction says that to prove that a property holds of all identifications it suffices to prove it only for the base case $\mathtt{refl}_{\mathtt{a}}$. That is, no inductive step is required – or, put another way, the principle of path induction proves the inductive step for us in advance, for all predicates on identifications.

A natural way to justify path induction would follow from a natural interpretation of the token former for identity types. Since the token former provides the trivial self-identifications $\mathtt{refl}_{\mathtt{a}}$ for each $\mathtt{a} : \mathtt{A}$ and no other identifications, then it would be natural to assume that these are the *only* identifications that exist. The principle of path induction would then be trivially true: to prove something

for all identifications it suffices to prove it for all identifications of the form $\mathtt{refl}_{\mathtt{a}}$, because there are no other identifications.

However, the HoTT Book [1, Remark 1.12.1] explicitly rules out this interpretation of path induction. Indeed, if this interpretation were correct then the project of Homotopy Type Theory would be of considerably less interest, since it is the non-trivial behaviour of identity types and the possibility of higher-order structure in them that allows the powerful connection with homotopy theory [1, Chapter 2]. Moreover, even with path induction it is not possible to prove a statement expressing 'all identifications trivial self-identifications' in the language of HoTT – indeed, we cannot even prove that all self-identifications are *trivial* self-identifications. So while it may seem natural to interpret the token constructor and path induction in this way, this is not an assertion that HoTT itself validates, nor is it a desirable additional assumption to add to HoTT.

Another way to justify path induction is to argue that there is a structure amongst identity types that ensures that properties held by trivial self-identifications must also be shared by other identifications. We could draw an analogy with vector spaces or groups. Given a basis for a vector space, we can define a (linear) function on the vector space by specifying its output just on the basis vectors. Similarly, given a presentation of a group in terms of generators and relations, we can define a (homomorphic) function on the group by specifying its output on the generators. In each case it is sufficient to specify the function's behaviour at a distinguished subset of elements, and then the linear or group structure ensures that the function is well-defined for all other elements. But of course in neither case would it make sense to argue that this indicates that the basis vectors or generators are the *only* elements of the vector space or group.

So in the present case what is the structure of the identity types, corresponding to the linear or group structure in the above examples? The HoTT Book [1, p. 67] says that "the family of types $(\mathtt{x} =_{\mathtt{A}} \mathtt{y})$, as $\mathtt{x}$, $\mathtt{y}$ vary over all elements of $\mathtt{A}$, is inductively defined by the elements of the form $\mathtt{refl}_{\mathtt{x}}$". This is supported by appeal to the homotopy interpretation of the type theory, to which we turn in the next section.

5 Homotopy theory and the homotopy interpretation

Homotopy theory, briefly, is the study of spaces up to continuous distortions. That is, the properties of spaces studied in homotopy theory are those that are preserved

by continuous deformations, and any property that is not so preserved 'cannot be seen' by homotopy.

Homotopy is generally defined by first considering *topological spaces* and *continuous functions* between them. Given any two continuous functions $f, g : X \to Y$ we define a *homotopy* between f and g as a continuous function $h : [0,1] \times X \to Y$ such that $\forall x \in X$, $h(0,x) = f(x)$ and $h(1,x) = g(x)$. We think of h as providing a continuous interpolation from f to g, or a smooth distortion from the image of X under f to the image of X under g. If there is such a function h then we say f and g are *homotopic*, written $f \sim g$.

Two spaces X and Y are *homotopy equivalent* if there are maps $f : X \to Y$ and $f' : Y \to X$ such that $f' \circ f \sim id_X$ and $f \circ f' \sim id_Y$. Homotopy equivalence is reflexive, symmetric and transitive, so we can define the equivalence class $[X]$ of all topological spaces homotopy equivalent to X. This is called the *homotopy type* of X. Homotopy theory does not distinguish between spaces that are homotopy equivalent, and thus homotopy types, rather than the topological spaces themselves, are the basic objects of study in homotopy theory.

For example, consider the topological space P consisting of a single point, and D^2, the unit disc in $\mathbb{R}^2$. There is a continuous function f from D^2 to P, and a continuous function f' from P to D^2 which picks out some point $p \in D^2$. The composition $f \circ f'$ is just the identity on P, while $f' \circ f$ maps the entire disc to the point p. To define a homotopy h between $f' \circ f$ and id_{D^2} we just pick out, for each $x \in D^2$, the straight line γ_x between x and p, parameterised by the interval $[0,1]$, and then define $h(t,x)$ as $\gamma_x(t)$. This shows that the unit disc D^2 is homotopy equivalent to a single point, or 'contractible'. That is, homotopy theory does not distinguish between the unit disc and the single point.

The homotopy interpretation of HoTT describes types as spaces (i.e. homotopy types), tokens `a`, `b` in a type as points in the space (i.e. functions from the single point into the space), and identifications `p` between tokens as paths between points (i.e. functions from $[0,1]$ to the space having those points as its end-points) [1, p. 5]. Thus the identity type $\mathtt{Id_A(a,b)}$ corresponds to the *path space* consisting of all paths from the point a to the point b in space A.

Using this interpretation we can give an account of the structure amongst identifications that justifies the principle of path induction. Given a fixed point a in space A, the *constant* path at a, corresponding to the trivial self-identification $\mathtt{refl_a}$, is the function $k_a : [0,1] \to A$ that sends every point in the interval to a. Given any point b and any path p from a to b, we can define a homotopy h between k_a and p by $h(t,x) = p(t \times x)$.

This relation of homotopy between paths is the structure in the identity types

required to justify path induction. Since the paths p and k_a are homotopic, any properties of k_a that respect homotopy must be shared by the path p. Thus, to show that all paths starting at a have such a property, it suffices to show that the constant path at a has that property, which is the homotopy-theoretic counterpart to the statement of based path induction. A similar argument gives the counterpart to path induction. In short, if we are free to vary one or both ends of a path, then any path can be *retracted* to a constant path at some point.

While this argument provides a justification for path induction, it clearly relies upon the details of homotopy theory for its motivation, and so this approach to justifying path induction is not suitable for an autonomous foundation for mathematics. In order to defend HoTT's claims to provide such a foundation, a different argument for path induction is needed.[21]

6 Understanding path induction without the homotopy interpretation

In this section we introduce two basic principles and give elementary arguments from pre-mathematical grounds that justify them. We then show that path induction follows straightforwardly from them, thus providing a justification for the elimination rule for identity types that does not depend upon sophisticated mathematics such as homotopy theory. It is known to type theorists that the two principles discussed below entail path induction – see, for example, [12, pp. 23-29], but as far as we are aware they do not put this fact to use in the way we do here. In particular, as mentioned above, in the HoTT Book there is no motivation for path induction independently of the homotopy interpretation.[22]

6.1 Uniqueness principles for types

When we introduced the token formers for product, coproduct, and dependent types in Section 2 we said that the constructors for a type give us all the tokens of

[21] We might wonder how much of the sophistication of homotopy theory is really required to get the above argument moving, and imagine that a stripped-down version might be given that avoids these technical details. That is, we might seek to give an argument for path induction modelled on the above but using only an intuitive pre-mathematical notion of *space*, *point*, and *path*. However, it is not at all clear that our ordinary common-sense notion of space supports the features required by the above argument: that identifications between points can be understood as paths, that paths can be continuously retracted without losing their essential features, and so on.

[22] Aside from the argument via the Yoneda lemma mentioned in footnote 15.

that type. So, for example, we said that every token in $\mathtt{A} + \mathtt{B}$ is either $\mathtt{inl}(\mathtt{a})$ for some $\mathtt{a} : \mathtt{A}$ or $\mathtt{inr}(\mathtt{b})$ for some $\mathtt{b} : \mathtt{B}$, because $\mathtt{inl}$ and $\mathtt{inr}$ are the two constructors for the coproduct. Likewise, we said that every token of $\mathtt{A} \times \mathtt{B}$ is of the form $(\mathtt{a}, \mathtt{b})$ for some $\mathtt{a} : \mathtt{A}$ and some $\mathtt{b} : \mathtt{B}$.

However, now that we've introduced identity types we can turn this claim into a formal statement that can be expressed within the language of HoTT. That is, we can state formally that every token of a given type is *equal* to the output of one of the token constructors for that type. Thus, for example, for every token $\mathtt{c} : \mathtt{A} + \mathtt{B}$ we have either a token of type $\mathtt{Id}_{\mathtt{A}+\mathtt{B}}(\mathtt{c}, \mathtt{inl}(\mathtt{a}))$ for some $\mathtt{a} : \mathtt{A}$ or a token of type $\mathtt{Id}_{\mathtt{A}+\mathtt{B}}(\mathtt{c}, \mathtt{inr}(\mathtt{b}))$ for some $\mathtt{b} : \mathtt{B}$. More formally:

$$\prod_{\mathtt{c}:\mathtt{A}+\mathtt{B}} \left(\sum_{\mathtt{a}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}+\mathtt{B}}(\mathtt{c}, \mathtt{inl}(\mathtt{a})) \; + \; \sum_{\mathtt{b}:\mathtt{B}} \mathtt{Id}_{\mathtt{A}+\mathtt{B}}(\mathtt{c}, \mathtt{inr}(\mathtt{b})) \right)$$

Not only can we express these statements entirely formally within the language of HoTT, we can also prove them [1, Chapter 1]. We call these *uniqueness principles* for the respective types.

A special case of this is the uniqueness principle for the Unit type $\mathtt{1}$ defined in Section 2.2. In this case we want to say that there is just a single token of $\mathtt{1}$, namely $* : \mathtt{1}$. However, since the language of HoTT cannot talk about the *absence* of tokens we must express this by saying that there is just one token *up to identity in* $\mathtt{1}$. We can state this as

$$\prod_{\mathtt{i}:\mathtt{1}} \mathtt{Id}_{\mathtt{1}}(*, \mathtt{i})$$

i.e. 'any token $\mathtt{i}$ of $\mathtt{1}$ is equal to $* : \mathtt{1}$'.

These formal statements capture more precisely the sense in which the token constructors for a type give us all the tokens of that type. In the light of this, we see that there are two subtly distinct ways to understand this claim. The first way to interpret it is that literally *every* token of the type is the output of one of the token constructors. But this is not exactly what the formal statements above say. Rather, the statements say that the token constructors give us every token of the type *up to identity*. That is, we cannot say that every token of a type *is* the output of one of the constructors, but rather that every token is *equal to* the output of one of the constructors, where the equality is witnessed by a token of the appropriate identity type. This subtle distinction is important in the case of the identity type itself and furthermore the uniqueness principle for identity types does not say that every identification is equal to the output of the token constructor. The fact that there may be other identifications is part of the novelty and power of HoTT.

6.2 The uniqueness principle for identity types

As explained above, the only token constructor for identity types is $\mathtt{refl}$ which appears to indicate that the only tokens of identity types that we could have are the trivial self-identifications of the form $\mathtt{refl}_{\mathtt{x}}$ for each $\mathtt{x} : \mathtt{A}$. However, in light of the above discussion, we should say instead that the constructor $\mathtt{refl}$ doesn't give us all the identifications, but rather all the identifications *up to identity in the appropriate type*.

To make this statement precise, we must formalise it in the language of HoTT. The most obvious way to do this, modelled on the formal statements of the previous section, would be to say 'for all identifications $\mathtt{p}$ there is a token $\mathtt{x}$ such that $\mathtt{p}$ is equal to $\mathtt{refl}_{\mathtt{x}}$ (in some appropriate type that has both $\mathtt{p}$ and $\mathtt{refl}_{\mathtt{x}}$ as tokens)'. But when we come to ask *which* type they are both found in we see that this can't be right, since $\mathtt{p}$ is a token of $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$ and $\mathtt{refl}_{\mathtt{x}}$ is a token of $\mathtt{Id}_{\mathtt{A}}(\mathtt{x},\mathtt{x})$, and each token belongs to exactly one type. We therefore need to find a type in which suitable *counterparts* of $\mathtt{p}$ and $\mathtt{refl}_{\mathtt{x}}$ can be found together.

Recall from Section 3 that given any $\mathtt{a} : \mathtt{A}$ we can define the based identity type $\sum_{\mathtt{x}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$, whose tokens are pairs $(\mathtt{b},\mathtt{p})$ consisting of a token of $\mathtt{A}$ and an identification between that token and the given fixed token $\mathtt{a}$. In particular, this type has the token $(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$. In based identity types we can therefore find (counterparts to) arbitrary identifications alongside (counterparts to) trivial self-identifications. For brevity, we write $\sum_{\mathtt{x}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$ as $\mathtt{E}_{\mathtt{a}}$.

We can therefore formalise a modified version of the above statement, saying that the counterparts to $\mathtt{p}$ and $\mathtt{refl}_{\mathtt{a}}$ are equal in $\mathtt{E}_{\mathtt{a}}$. We express this formally by saying that the following type

$$\prod_{(\mathtt{b},\mathtt{p}):\mathtt{E}_{\mathtt{a}}} \mathtt{Id}_{\mathtt{E}_{\mathtt{a}}}((\mathtt{a},\mathtt{refl}_{\mathtt{a}}),(\mathtt{b},\mathtt{p}))$$

is inhabited. This is the uniqueness principle for identity types. It doesn't say that the *only* token of $\sum_{\mathtt{x}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$ is $(\mathtt{a},\mathtt{refl}_{\mathtt{a}})$, as we might have expected from our first understanding of the token constructor for identity types. Rather, it says that this is the only token *up to identity* in $\sum_{\mathtt{x}:\mathtt{A}} \mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{x})$.

As another way of understanding this, recall from Section 2 that we can read dependent pair types $\sum_{\mathtt{b}:\mathtt{B}} \mathtt{P}(\mathtt{b})$ in two different ways: as an existentially quantified proposition, and as a subtype consisting of the tokens of $\mathtt{B}$ that satisfy the predicate $\mathtt{P}$ (where each such token is accompanied by a certificate to that fact). On this understanding, the based identity type $\mathtt{E}_{\mathtt{a}}$ is the collection of tokens of $\mathtt{A}$ satisfying the 'equal to $\mathtt{a}$' predicate. Naïvely we would say that the only token of $\mathtt{A}$ that is equal to $\mathtt{a}$ is $\mathtt{a}$ itself, and so we would expect $\mathtt{E}_{\mathtt{a}}$ to have just a single token. The

type above corresponds to exactly this proposition. Just as in the case of the Unit type in Section 6.1 above, we have a distinguished token $(\mathtt{a}, \mathtt{refl}_{\mathtt{a}})$ that belongs to this type, and a uniqueness principle that says that all tokens are equal to this one. Thus the type $\mathtt{E}_{\mathtt{a}}$ does indeed have just one token, up to identity.[23]

6.3 The substitution of equals for equals

The principle of *substitution salva veritate*, according to which if s and t are identical then one can be substituted for the other in any statement while *saving the truth* of that statement, is a fundamental part of our pre-mathematical understanding of identity, and any formalisation of identity must respect it. In HoTT this principle states that if there is an identification between s and t then anything that is true of one is true of the other. In HoTT, 'something being true of s' corresponds to the existence of a token of type $\mathtt{P}(\mathtt{s})$ for some predicate P. Hence we can state the principle as follows: for any type B, any predicate P on B, and any $\mathtt{s} : \mathtt{B}$ and $\mathtt{t} : \mathtt{B}$, there is a function of type

$$\mathtt{Id}_{\mathtt{B}}(\mathtt{s}, \mathtt{t}) \times \mathtt{P}(\mathtt{t}) \to \mathtt{P}(\mathtt{s}) \tag{1}$$

That is, given an identification of s with t and a certificate to a proposition about t, we can produce a certificate to the corresponding proposition about s.[24]

From this we can derive other principles. Recall from Section 3.1.1 that for any $\mathtt{t} : \mathtt{B}$ we can define a predicate $\mathtt{Id}_{\mathtt{B}}[\mathtt{t}]$ that says of any $\mathtt{x} : \mathtt{B}$ that it is equal to t (that is, given an x it returns the identity type $\mathtt{Id}_{\mathtt{B}}(\mathtt{t}, \mathtt{x})$). Using this predicate in Equation 1 gives a function

$$\mathtt{Id}_{\mathtt{B}}(\mathtt{s}, \mathtt{t}) \times \mathtt{Id}_{\mathtt{B}}(\mathtt{t}, \mathtt{t}) \to \mathtt{Id}_{\mathtt{B}}(\mathtt{t}, \mathtt{s})$$

Since we always have the token $\mathtt{refl}_{\mathtt{t}} : \mathtt{Id}_{\mathtt{B}}(\mathtt{t}, \mathtt{t})$ that can be given as the second argument to this function, we derive the symmetry of identity:

$$\mathtt{Id}_{\mathtt{B}}(\mathtt{s}, \mathtt{t}) \to \mathtt{Id}_{\mathtt{B}}(\mathtt{t}, \mathtt{s})$$

[23] Alternatively, we could interpret $\mathtt{E}_{\mathtt{a}}$ as the *singleton* of a – i.e. the subtype of A consisting of all tokens of A that are identical with a – which should naturally have just one token up to identity. Yet another way of thinking about this, via the homotopy interpretation, is that $\mathtt{E}_{\mathtt{a}}$ is the *homotopy fibre* of the identity at a, or the *path space* at a. The uniqueness principle then says that this is *contractible* (Contractibility of $\mathtt{E}_{\mathtt{a}}$ is proved in the HoTT Book [1, Lemma 3.11.8], but the proof uses path induction.) However, neither of these interpretations is suitable for our purposes, of course, since the first depends on ideas from set theory while the second relies on the homotopy interpretation.

[24] This principle is called 'transport' in the HoTT Book, but rather than taking it as a basic principle it is derived from path induction [1, Lemma 2.3.1].

Similarly, for any $\mathtt{u} : \mathtt{B}$, if we use predicate $\mathtt{Id_B}[\mathtt{u}]$ in Equation 1 then we obtain a function

$$\mathtt{Id_B}(\mathtt{s},\mathtt{t}) \times \mathtt{Id_B}(\mathtt{u},\mathtt{t}) \to \mathtt{Id_B}(\mathtt{u},\mathtt{s})$$

from which, by applying symmetry, we derive the transitivity of identity:

$$\mathtt{Id_B}(\mathtt{s},\mathtt{t}) \times \mathtt{Id_B}(\mathtt{t},\mathtt{u}) \to \mathtt{Id_B}(\mathtt{s},\mathtt{u})$$

The above two properties are just the usual features of identity. However, we can also use *substitution salva veritate*, along with the uniqueness principle for identity types, to derive based path induction.

Consider again the type $\mathtt{E_a}$ defined above which, recall, is an abbreviation for $\sum_{\mathtt{x:A}} \mathtt{Id_A}(\mathtt{a},\mathtt{x})$, where $\mathtt{a}$ is an arbitrary token of $\mathtt{A}$. Let $\mathtt{P}$ be any predicate on $\mathtt{E_a}$, and $(\mathtt{b},\mathtt{p})$ be any token of $\mathtt{E_a}$. Since Equation 1 holds for any two tokens, we have a function

$$\mathtt{Id_{E_a}}((\mathtt{b},\mathtt{p}),(\mathtt{a},\mathtt{refl_a})) \times \mathtt{P}(\mathtt{a},\mathtt{refl_a}) \to \mathtt{P}(\mathtt{b},\mathtt{p})$$

Recall that the uniqueness principle for identity types says that for *every* token $(\mathtt{b},\mathtt{p}) : \mathtt{E_a}$ the type $\mathtt{Id_{E_a}}((\mathtt{a},\mathtt{refl_a}),(\mathtt{b},\mathtt{p}))$ is inhabited. Thus by symmetry of identity so is $\mathtt{Id_{E_a}}((\mathtt{b},\mathtt{p}),(\mathtt{a},\mathtt{refl_a}))$, and so this token can always be given as the first argument to the above function. So for any predicate $\mathtt{P}$ on $\mathtt{E_a}$ and any token $(\mathtt{b},\mathtt{p}) : \mathtt{E_a}$ we have a function

$$\mathtt{P}(\mathtt{a},\mathtt{refl_a}) \to \mathtt{P}(\mathtt{b},\mathtt{p})$$

This is precisely the inductive step in an inductive proof that $\mathtt{P}$ holds for all tokens $(\mathtt{b},\mathtt{p})$. Thus, if we want to prove that an arbitrary predicate $\mathtt{P}$ holds for all identifications involving $\mathtt{a} : \mathtt{A}$, we only need to prove that it holds for the trivial self-identification $\mathtt{refl_a}$. This is exactly the statement of based path induction.

7 Summary and Conclusions

We began by distinguishing two notions of a foundation for mathematics. In this paper we focus only on the question of whether HoTT can be a foundation in the second, stronger sense. Although we have not elaborated in detail what is required for a theory to be foundation in this sense,[25] a necessary condition is that it be *autonomous*, i.e. that its motivation and formulation not depend upon existing advanced mathematics.

[25] We take this issue up in more detail in another paper [10].

We have shown that much of HoTT can be developed without any such recourse to advanced mathematics, and importantly without reference to the homotopy interpretation that is used for the exposition in the HoTT Book. We described how identity is treated in HoTT, including (based) path induction, the elimination rule for the identity type. We described the standard argument given in the HoTT Book for path induction, which depends upon the homotopy interpretation. Finally, we have given an account of based path induction that makes no reference to homotopy theory or any other advanced mathematical domains, depending only upon the uniqueness principle for identity types and the principle of *substitution salva veritate* which is a central defining characteristic of the notion of identity.

This argument demonstrates that based path induction can be justified on the basis of pre-mathematical principles, and is therefore not an obstruction to HoTT's providing an autonomous foundation for mathematics.

References

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

[2] Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73. Proceedings of the Logic Colloquium, Bristol, July 1973*, pages 73–118. 1974.

[3] Steve Awodey. Structuralism, Invariance, and Univalence. http://www.andrew.cmu.edu/user/awodey/preprints/siu.pdf, 2014.

[4] John P. Mayberry. What is Required of a Foundation for Mathematics? *Philosophia Mathematica*, 2(1):16–35, 1994.

[5] Øystein Linnebo and Richard Pettigrew. Category Theory as an Autonomous Foundation. *Philosophia Mathematica*, 19(3):227–254, 2011.

[6] John P. Mayberry. *The Foundations of Mathematics in the Theory of Sets*. Cambridge University Press, 2000.

[7] Authors. A Primer on Homotopy Type Theory. 2014.

[8] Steve Awodey and Michael Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1), 2009.

[9] Per Martin-Löf. On the Meanings of the Logical Constants and the Justifications of the Logical Laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[10] Authors. Does Homotopy Type Theory Provide a Foundation for Mathematics? 2014.

[11] Christine Paulin-Mohring. Inductive definitions in the system coq – rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993.

[12] Thierry Coquand. Equality and Dependent Type Theory. A talk given for the 24th AILA meeting, Bologna, February 2011.