1–21
10.1093/philmat/nkv014
Philosophia Mathematica

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

James Ladyman* and Stuart Presnell**

Department of Philosophy, University of Bristol, Bristol BS6 6JL, U.K.
*E-mail: james.ladyman@bristol.ac.uk
**E-mail: stuart.presnell@bristol.ac.uk

ABSTRACT

Homotopy Type Theory (HoTT) is a proposed new language and foundation for mathematics, combining algebraic topology with logic. An important rule for the treatment of identity in HoTT is *path induction*, which is commonly explained by appeal to the *homotopy interpretation* of the theory's types, tokens, and identities as (respectively) spaces, points, and paths. However, if HoTT is to be an *autonomous* foundation then such an interpretation cannot play a fundamental role. In this paper we give a derivation of path induction, motivated from pre-mathematical considerations, without recourse to homotopy theory.

1. INTRODUCTION

Homotopy Type Theory (HoTT) [The Univalent Foundations Program, 2013] 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 generated considerable excitement because it facilitates automatic proof verification [The Univalent Foundations Program, 2013, p. 10]. HoTT is based on the constructive intensional type theory due to Martin-Löf [1975] and hence is very

†We would like to thank Steve Awodey, Urs Schreiber, Richard Pettigrew, Øystein Linnebo, Teru Thomas, Patrick Walsh, and John Wigglesworth for reading and commenting on earlier drafts of this paper. We are very grateful to two anonymous referees for *Philosophia Mathematica* for giving us lots of very helpful and insightful comments. The research on which this paper is based, "Applying Homotopy Type Theory in Logic, Metaphysics, and Philosophy of Physics", is funded by a Leverhulme Trust Research Project Grant (RPG-2013-228).

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]

As well as 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 [Mayberry, 1994] and requires that the system be *autonomous* in something like the sense of Linnebo and Pettigrew [2011].

The former sense, that of a framework in which to do mathematics, consists of some mathematical entities that are taken to be 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 sets, and the symbols for the membership relation plus the background logical language.

Similarly, HoTT is intended to provide a formalism in which all mathematical entities can be described in a type-theoretic language, and in which the theorems of mathematics can be proved by application of rules that embody a form of constructive logic. Indeed, one of the most impressive achievements of those working on HoTT is that they have successfully formalised it in two different ways, using the languages Coq and Agda in a way that is completely self-contained, and have used it to recover substantial other mathematical theories.[2] Therefore it seems, and we are happy to grant, that HoTT can serve as a framework for mathematics. In any case, we do not examine this question further, deferring to the discussion in the HoTT Book.[3] Rather, our concern is with the justification of HoTT as standardly formulated.

[1]Note that [The Univalent Foundations Program, 2013] (which we henceforth refer to as the 'HoTT Book') has the subtitle '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. This axiom is an important idea in HoTT, and has major implications for the understanding of identity (see [Awodey, 2014]). More generally, we think the history and philosophy of HoTT is of tremendous interest and is worthy of the attention of philosophers of mathematics for all sorts of reasons. However, our interests and aim in this paper are very narrowly circumscribed, and in particular we focus on identity types in the the basic language of HoTT which does not include Univalence. An exploration of philosophical issues relating to Univalence will be the subject of a subsequent paper.

[2]The code for these formalisations can be found at https://github.com/HoTT/HoTT and https://github.com/HoTT/HoTT-Agda.

[3]See [HoTT Book, Chapter 10] and in particular Theorem 10.5.8 on the reconstruction of set theory, the cumulative hierarchy, and the axioms of ZFC in HoTT.

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 (some, if not all, of) the existing objects and proofs of mathematics. Demonstrating that a proposed system qualifies as a foundation (in this sense) may be done by giving a direct reconstruction of familiar parts of mathematics in the new language (although this can only give a defeasible demonstration, since at any stage some parts of mathematics remain untranslated). Alternatively (or additionally), one may proceed by spelling out how exactly some already-recognised foundational system (such as ZFC) can be encoded in the proposed language, and thus any foundational claims of the earlier system are directly inherited by the new system.[4]

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 is the one more usually of interest to philosophers. A foundation in this second sense, which Linnebo and Pettigrew [2011] 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 [Mayberry, 1994]. Likewise it must be possible to take any mathematical proof and break it down into steps whose justification requires no pre-existing mathematical understanding. Set theory is generally taken to provide a foundation for mathematics in this stronger sense, because the notion of 'a collection of entities' is sufficiently primitive to require no mathematical explanation.[5]

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.

[4]Such a reconstruction of an existing foundation may not always be possible or desirable. For example, some varieties of constructive mathematics cannot reproduce all the claims of classical mathematics since they explicitly reject them, and moreover hold the rejection of these claims to be virtues rather than failings of their approach. See, for example, [Richman, 1996].

[5]However, arguably what is intuitive is *naïve* set theory, which is not consistent, whereas the foundation in ZFC is less clearly intuitive: in particular axioms such as replacement and infinity are hardly self-evident [Mayberry, 2000].

The presentation of HoTT given in the HoTT Book freely makes use of an analogy between types and spaces (where the latter are understood as in homotopy theory), and between identifications and paths.[6] 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.[7] 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 similar to that given in [HoTT Book, Chapter 1], but with a greater emphasis on the motivation from pre-mathematical considerations, which is explained in considerably more detail in [Ladyman and Presnell, forthcoming].)

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.[8] In the presentation in the HoTT Book identifications such as $a = b$ are thought of as paths in (homotopy) spaces, and the clearest and most accessible 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 that would of course undermine any claims for HoTT as an autonomous foundation for mathematics (although, of course, its status as a foundation in the first, weaker, sense of a framework would not be impugned). In this paper we offer an alternative to these justifications of path induction by giving a justification for this principle that depends only upon pre-mathematical ideas.

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 novel contribution of this paper is in Section 6 where we give a justification of path induction that relies only upon

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

[7] Some presentations of HoTT simply state the type-theoretic definitions without giving any justification. This of course avoids any dependence upon homotopy theory or other sophisticated mathematics but fails to provide a foundation in the stronger sense in which we are interested here.

[8] The words 'identity' and 'equality' are used interchangeably in the literature on HoTT but we generally only use the former.

pre-mathematical ideas, without reference to homotopy or to any other pre-existing mathematics.[9]

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.[10] 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 '$\mathtt{x} : \mathtt{A}$' to denote that $\mathtt{x}$ is a token of type $\mathtt{A}$.

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. Thus, unlike in ZFC, HoTT 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* the same.[11] 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.[12] Rather, the linguistic entities are *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

[9]The mathematical result that path induction follows from the particular principles we use is known (see, for example, [Coquand, 2011, pp. 23–29]). However, the philosophical work to which we put this result to provide an autonomous presentation of the theory is novel.

[10]Tokens are sometimes called 'witnesses' or 'proofs' to their corresponding propositions, but we prefer to avoid this terminology, for reasons explained in [Ladyman and Presnell, forthcoming].

[11]A referee points out that Fermat's Last Theorem has not yet been proved in HoTT. For another example, consider 'the fundamental group of the circle' and 'the integers $\mathbb{Z}$', which have been proved equivalent in HoTT [HoTT Book, Section 8.1].

[12]In the HoTT Book the word 'term' is sometimes used where we use 'token'. However, in some literature on type theory 'term' is used to refer to syntactic elements; so we avoid this terminology. The HoTT Book also uses words such as 'object', 'point', and 'element' where we use 'token', but these words also have connotations that we would prefer to avoid: 'object' begs the question of how tokens are to be interpreted, likewise 'point' implies a spatial interpretation of types, and 'element' is reminiscent of set theory. (We are grateful to an anonymous referee who notes that at least some

be thought of as manipulations 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 [Martin-Löf, 1975; HoTT Book, 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.[13]

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 [Martin-Löf, 1996].)

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, which is introduced in Section 3. (For a more detailed exposition see [HoTT Book, Chapter 1] or [Ladyman and Presnell, forthcoming].)

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, in lambda calculus), thus producing an expression naming a token of type $\mathtt{B}$.[14]

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 to produce a token of $\mathtt{B}$. Since tokens of types are understood as certificates to the truth of their corresponding propositions, this suggests that the proposition

of the authors of the HoTT Book reject the use of the words 'term' and 'object' for similar reasons to ours, but endorse the use of 'point' and 'element' because they specifically want to evoke the connotations indicated above.)

[13]This way of thinking is not explicit in the HoTT Book, but we recommend it for reasons explained in [Ladyman and Presnell, forthcoming]. 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.)

[14]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.

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 [Ladyman and Presnell, forthcoming, Section 3.3].

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 $\mathtt{A}$ and $\mathtt{B}$.
- A certificate to the truth of a *disjunction* of two propositions $\mathtt{A}$ and $\mathtt{B}$ is a token that is either a certificate to $\mathtt{A}$ or a certificate to $\mathtt{B}$. 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 $\mathtt{A}$ and $\mathtt{B}$.[15]
- 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 $\mathtt{0}$ is a type that by definition has no token constructors. [16]

Just as $\mathtt{0}$ corresponds to a proposition that is by definition false, we also introduce a type $\mathtt{1}$ corresponding to a proposition that is by definition true. This Unit type is defined to have exactly one token, denoted $*$.[17] 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.

- A certificate to the statement that all tokens of type $\mathtt{A}$ satisfy some condition is something that provides, for each $\mathtt{x} : \mathtt{A}$, a certificate to the fact that $\mathtt{x}$ does indeed satisfy that condition, *i.e.*, a function that takes tokens of $\mathtt{A}$ as inputs and returns these certificates as outputs.
- A certificate to the statement that there exists a token of type $\mathtt{A}$ that satisfies some condition consists of a pair $(\mathtt{x}, \mathtt{c})$, where $\mathtt{x}$ is a token of $\mathtt{A}$ and $\mathtt{c}$ is a certificate to the fact that this particular $\mathtt{x}$ satisfies the condition.[18]

[15] Note that our usage of 'disjunction' is the same as that of Martin-Löf [1996], whereas the HoTT Book reserves the word 'disjunction' for a different but related type, called the 'truncation' $\|\mathtt{A} + \mathtt{B}\|$ of the coproduct. For more details on the distinction, see [HoTT Book, Definition 3.7.1 and Section 3.10]. This distinction does not affect any of the arguments in this paper.

[16] Consistency of the system then consists in the claim that $\mathtt{0}$ has no tokens.

[17] Here we set aside a detail that we examine more carefully in Section 6.1.

[18] As in footnote 15, note that our usage follows that of Martin-Löf whereas the HoTT Book introduces a 'truncation', as explained in [HoTT Book, Definition 3.7.1 and Section 3.10]. As above, this distinction does not affect any of the arguments in this paper.

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 must 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 define this as a function taking tokens of A as input and returning a *type* as output.[19] 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 P(x), when given y : A it returns a token of P(y), and so on. We write the type to which these dependent functions belong as '$\prod_{a:A} P(a)$'. This notation recalls the set-theoretic notation for dependent products. A token of this type is a certificate to the universally quantified statement that all tokens of A satisfy predicate P.
- 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 P(x), if the first component is y : A then the second component is a token of type P(y), and so on. We write the type to which these dependent pairs belong as '$\sum_{a:A} P(a)$'. This notation recalls the set-theoretic notation for disjoint sums/coproducts. A token of this type is a certificate to the existentially quantified statement that there exists a token of A that satisfies predicate P.

Aside from allowing us to express existentially quantified propositions as above, dependent-pair types are also useful in two other ways. First, we can use them to form the type-theoretic analogue of disjoint unions in set theory: given a type I and a function P mapping each token of I to one of a collection of types A, B, C, . . ., the dependent-pair type $\sum_{i:I} P(i)$ corresponds to the disjoint union of A, B, C, ... Secondly, we can also think of a dependent-pair type as a way of forming a *subtype* by imposing a predicate: the type $\sum_{a:A} P(a)$ consists of those tokens of A that satisfy the predicate P, each accompanied by a certificate to that fact.[20]

[19] In the light of the definition of functions above, according to which they return a *token* of their output type, it is clear that for predicates to be functions requires the introduction of a higher-order type that has other types as its tokens. This must be done with care to avoid paradox. In particular we must disallow this type from being a token of itself. For the technical details of how this is done by the introduction of 'universes' see [HoTT Book, Section 1.3]. This is not required for the argument of the present paper.

[20] Note that dependent-pair types are not quite like subsets. When we pick out a subset of a set by applying a predicate, each element satisfying the predicate occurs exactly once in the subset. In contrast, in a dependent-pair type $\sum_{a:A} P(a)$, for a given token a : A there may be multiple distinct

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. [21]

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 new type, given 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. This may be required to satisfy particular conditions, which are expressed as a 'computation rule'.

For example, for the product type described above:

- the type former takes two types $\mathtt{A}$ and $\mathtt{B}$ and returns the type $\mathtt{A} \times \mathtt{B}$.
- the token constructor takes a token $\mathtt{a} : \mathtt{A}$ and a token $\mathtt{b} : \mathtt{B}$ and returns a token $(\mathtt{a}, \mathtt{b}) : \mathtt{A} \times \mathtt{B}$.
- the eliminators are the projectors $\mathtt{fst} : \mathtt{A} \times \mathtt{B} \to \mathtt{A}$ and $\mathtt{snd} : \mathtt{A} \times \mathtt{B} \to \mathtt{B}$ that return the first (resp. second) component of a given pair. The computation rules in this case are $\mathtt{fst}(\mathtt{a}, \mathtt{b}) :\equiv \mathtt{a}$ and $\mathtt{snd}(\mathtt{a}, \mathtt{b}) :\equiv \mathtt{b}$ which ensure that these functions operate as intended.

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

tokens $(\mathtt{a}, \mathtt{p}), (\mathtt{a}, \mathtt{q}), \ldots$ if $\mathtt{P}(\mathtt{a})$ has multiple tokens $\mathtt{p}, \mathtt{q}, \ldots$. One response to this discrepancy is to restrict the use of the word 'subtype' to the case where all tokens $(\mathtt{a}, \mathtt{p}), (\mathtt{a}, \mathtt{q}), \ldots$ for any given $\mathtt{a} : \mathtt{A}$ are identical. (This is one reason for taking existentials to be truncated, as is done in the HoTT Book, as noted in footnote 18). However, the analogy between dependent-pair types and subsets is sufficiently useful that we think the appropriate response is instead to expand the use of the word 'subtype' to dependent-pair types in general and to bear in mind the above caveat, rather than imposing such a restriction. Either way, this does not affect the argument of this paper.

[21] The elimination rules for the types described above are fairly obvious, and will not be needed in what follows here. For details see [HoTT Book] or [Ladyman and Presnell, forthcoming].

notions, such as substitution and conjunction. Everything in HoTT$^-$ is therefore eligible to form part of an autonomous foundation for mathematics.[22]

3. IDENTITY TYPES AND PATH INDUCTION

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 identical.[23] HoTT$^-$ is quite powerful: for example, we can introduce a type $\mathbb{N}$ whose tokens correspond to natural numbers, and we can define all the usual operations of arithmetic on them. But without identity types many elementary mathematical truths cannot be expressed, because many such truths are about identity in mathematical theories. Without a notion of identity we cannot do even the most basic number theory because there are no equations, and we cannot 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. Identity is obviously an essential component of a language that claims to serve as a foundation for mathematics.

One thing we can do is introduce a symbol into the language to denote identity between expressions that name types and tokens. If the expressions exp_1 and exp_2 both name types or tokens then the expression $exp_1 \equiv exp_2$ says that these expressions in fact name the same type or token. Thus any occurrence of exp_1 may always be replaced by exp_2 in any circumstance without changing meaning.[24] This is called 'external' or 'judgmental' equality. For example, recalling the projectors of the product type discussed above, we have the judgmental equality $\mathtt{fst(snd(a, (b, c))) \equiv b}$.

However, this is not sufficient to allow us to express mathematical facts involving identity, such as the number-theoretic facts mentioned above. For example, if $\mathtt{x}$ and $\mathtt{y}$ are variables of type $\mathbb{N}$ then the expressions '$\mathtt{x + y}$' and '$\mathtt{y + x}$' are not judgmentally equal since addition is not commutative *by definition*; rather, the commutativity of addition is a proposition that must be proved from the definition of the addition function.

A statement of identity between two tokens is a proposition, and for any other kind of mathematical proposition that we can express there is a corresponding type. The resources of HoTT$^-$ defined above, even supplemented as just suggested, do not provide a way to form a type corresponding to such propositions. We must therefore supplement the language by introducing *identity types*. This gives an 'internal' or 'propositional' identity, allowing us to treat identifications as objects of study in

[22]We examine the autonomy of HoTT$^-$ and related issues in a companion paper [Ladyman and Presnell, forthcoming].

[23]There are other features of the full theory of HoTT missing as well, such as Univalence and function extensionality (both of which require identity for their definition). 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.

[24]We have implicitly made use of this notion of identity in stating the computation rule for product types above, and in general computation rules may be thought of as statements of identity between expressions.

their own right. Moreover, since the type theory is constructive and intensional, representing identity internally in the type theory allows identity to take on the rich and interesting structure of infinity groupoids that makes HoTT of so much mathematical interest and allows the homotopy interpretation of the type theory. (We explain the beginnings of this structure below but it is discussed in much greater detail in [HoTT Book, Chapter 2]).

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 Sections 4 and 5 we briefly consider some possible justifications for this rule and find that none of them is compatible with the autonomy of HoTT. Then in Section 6 we give our own justification of path induction that avoids the use of sophisticated mathematical ideas.

3.1. Identity Types

For any two tokens $\mathtt{a} : \mathtt{A}$ and $\mathtt{b} : \mathtt{A}$ of the same type we can state the proposition that a and b are identical. Since every token belongs to exactly one type, tokens that are not of the same type cannot be identified with one another. Thus the type former for the identity type must take as inputs a type A and two tokens $\mathtt{a} : \mathtt{A}$ and $\mathtt{b} : \mathtt{A}$. The type corresponding to the proposition that $\mathtt{a} : \mathtt{A}$ and $\mathtt{b} : \mathtt{A}$ are identical is written as '$\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$', or sometimes as '$\mathtt{a} =_{\mathtt{A}} \mathtt{b}$'. A token of $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$ is a certificate to the proposition that the tokens a and b are identical. We call such a token an *identification* of a and b. As with any type in the theory, in general there may be multiple tokens of $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$, or just one, or none at all. That is, a given pair of tokens are not simply 'identical' or 'non-identical', but rather there may be multiple different identifications between two tokens. [25]

The token constructor for the identity type provides the identifications that should be included as part of the *definition* of identity. It is clear that we should not just be able to create tokens of $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$ freely for arbitrary a and b, since this corresponds to proving that any two tokens of a type are identical (for example, that two arbitrary natural numbers are identical). The only identifications that are guaranteed to exist (with no further assumptions or premises) are the *trivial self-identifications*. That is, for any type A we must have a certificate to the proposition that each token of 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 A and any token $\mathtt{x} : \mathtt{A}$, a token of $\mathtt{Id}_{\mathtt{A}}(\mathtt{x},\mathtt{x})$, which we write as $\mathtt{refl}_{\mathtt{x}} : \mathtt{Id}_{\mathtt{A}}(\mathtt{x},\mathtt{x})$.

Since the token constructor for the identity type allows us to construct one certificate to $\mathtt{Id}_{\mathtt{A}}(\mathtt{x},\mathtt{x})$ 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 subtler than this.

[25] This is due to 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 [HoTT Book, pp. 71, 128] for further details.

3.2. Path Induction and Based Path Induction

The elimination rule for the identity type is called *path induction*.[26] The essential idea of path induction is as follows: to prove that a property holds for *all* identifications between arbitrary 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 *all* properties, that if the base case satisfies that property then all identifications do.[27] Thus we are left to prove just the base case (*i.e.*, that the property holds for all trivial self-identifications $\mathtt{refl_a}$).

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 [Paulin-Mohring, 1993]; [HoTT Book, Section 1.12.2] but a little easier to explain.[28]

3.2.1. *Based Path Induction*

We can form an identity type $\mathtt{Id_A(a,x)}$ for any pair of tokens $\mathtt{a}$ and $\mathtt{x}$ of a given type $\mathtt{A}$.[29] Thus for any $\mathtt{a : A}$ we can define a predicate $\mathtt{Id_A(a,-)}$ that, when given a token $\mathtt{b}$, returns the identity type $\mathtt{Id_A(a,b)}$. That is, $\mathtt{Id_A(a,-)}$ is the 'identical-to-a' predicate.

Fixing a particular $\mathtt{a : A}$, we can use the predicate $\mathtt{Id_A(a,-)}$ to form the dependent-pair type $\sum_{\mathtt{x:A}} \mathtt{Id_A(a,x)}$, which we call the *based identity type*. As a dependent-pair type, the tokens of this type are pairs $\mathtt{(b,p)}$, where $\mathtt{b}$ is a token of $\mathtt{A}$ and $\mathtt{p}$ is a token of $\mathtt{Id_A(a,b)}$ (*i.e.*, an identification of $\mathtt{a}$ with that particular $\mathtt{b}$). Reading this as the type-theoretic counterpart of an existentially quantified proposition, the based identity type corresponds to the proposition 'there exists a token of $\mathtt{A}$ that is identical to $\mathtt{a}$'. This proposition is of course true, since identity is reflexive and so the token $\mathtt{(a, refl_a)}$ 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 $\mathtt{(b,p)}$ and returns a type involving $\mathtt{b}$ and $\mathtt{p}$ (corresponding to a proposition about identification $\mathtt{p}$). 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 $\mathtt{Y(a, refl_a)}$ — *i.e.*, a certificate to the fact that $\mathtt{refl_a}$ satisfies the property $\mathtt{Y}$ — then we can produce a certificate

[26] While the elimination rule was part of Martin-Löf's intensional type theory [Martin-Löf, 1975], this name for it comes from the homotopy interpretation; see Section 5.

[27] In the homotopy interpretation the inductive nature of path induction is more apparent where it corresponds to the notion of retraction of paths (see Section 5).

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

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

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.

3.2.2. *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 in the based identity type we fixed one token of $\mathtt{A}$ and let the other range over all values in $\mathtt{A}$, in the total identity type we let 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 explicitly rules out this interpretation of path induction [HoTT Book, Remark 1.12.1]. 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 [HoTT Book,

Chapter 2]. Moreover, even with path induction it does not follow that all identifications are trivial self-identifications (indeed, attempting to express this in the language of HoTT results in something that is not even well-typed; see Section 6.2). Nor does it follow that all self-identifications are trivial self-identifications [HoTT Book, Exercise 1.14]. 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.

The rule for the treatment of identity types that we are calling (following the HoTT Book) 'path induction' is part of the original formulation of Martin-Löf's type theory [1975].[30] He motivates it by observing that identity is the 'least reflexive relation': *i.e.*, identity is a reflexive relation, and for any other reflexive relation Q, if identity holds between two tokens then Q holds between them as well. However, while this may be adequate as a heuristic motivation for introducing path induction, it is not suitable as part of a foundation. The notion of reflexivity itself depends upon the notion of identity, and therefore cannot be part of the definition of the identity relation on pain of circularity. One might attempt to circumvent this problem by appealing to judgmental equality as a starting point, defining a relation to be reflexive iff it holds of pairs of tokens that are judgmentally equal. This avoids circularity, since it refers only to judgmental equality (which is defined independently of the internal identity relation). However, since identity in the theory does not entail judgmental equality, this will not capture the relevant notion of reflexivity, which pertains to the former, not the latter. That is, if we think the notion of identity between tokens is captured by the internal identity relation (rather than external judgmental equality) then the notion of reflexivity should likewise refer to the internal identity relation.[31]

The HoTT Book gives an alternative approach to the definition of identity types as an 'inductive family' of types [HoTT Book, Section 5.8]. However, the latter notion is mathematically advanced, and therefore cannot be part of an autonomous foundation that must depend only upon elementary principles. We might alternatively try to state the definition of identity types as a particular 'freely generated functor' [HoTT Book, Section 5.8], perhaps invoking the Yoneda lemma. However, any such definition requires the use of at least some of the ideas of category theory, again making it unsuitable for a foundation that is supposed to be autonomous in the sense of Section 1.

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. Consider 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

[30] The name 'path induction' derives from the homotopy interpretation of identity, explained in the next section. 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 [2009].

[31] We are grateful to an anonymous referee for pushing us to strengthen our objection to this characterisation of identity.

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 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 $\mathtt{refl}_{\mathtt{x}}$" [HoTT Book, p. 67]. 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 $\mathtt{a}, \mathtt{b}$ in a type as points in the space (*i.e.*, functions from the single point into the space), and identifications $\mathtt{p}$ between tokens as paths between points (*i.e.*, functions from $[0,1]$ to the space, having those points as end-points) [HoTT Book, p. 5]. Thus the identity type $\mathtt{Id}_{\mathtt{A}}(\mathtt{a},\mathtt{b})$ 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 $\texttt{refl}_a$, 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 $h(t,x) = p(tx)$.

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.[32]

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 but as far as we are aware they do not put this fact to use in the way we do here.[33] In particular, as mentioned above, in the HoTT Book there is no adequate motivation for path induction independent of either the homotopy interpretation or other sophisticated mathematics.

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 that type. So, for example, we said that every token in $\texttt{A} + \texttt{B}$ is either $\texttt{inl(a)}$ for some $\texttt{a} : \texttt{A}$

[32] 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.

[33] See, for example, [Coquand, 2011, pp. 23–29]. That the principles followed from path induction was certainly known much earlier by Martin-Löf.

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 have 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 *identical* 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:

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

These statements can be expressed entirely formally within the language of HoTT, and moreover they can be proved as theorems [HoTT Book, 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 $* : \mathtt{1}$. We express this by saying that any token $\mathtt{i}$ of $\mathtt{1}$ is identical to $* : \mathtt{1}$. We can state this as

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

i.e., there is just one token *up to identity in* $\mathtt{1}$.

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 in the type*. 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 *identical to* the output of one of the constructors, where the identity is witnessed by a token of the appropriate identity type. This subtle distinction is important in the case of the identity type itself.

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 $\mathtt{refl}$ does not 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 identical 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 cannot be

right, since p is a token of $\mathtt{Id_A(a,b)}$ and $\mathtt{refl_x}$ is a token of $\mathtt{Id_A(x,x)}$, and each token belongs to exactly one type. We therefore need to find a type in which suitable *counterparts* of p and $\mathtt{refl_x}$ can be found together.

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

We can therefore formalise a modified version of the above statement, saying that the counterparts to p and $\mathtt{refl_a}$ are identical in E(a). We express this formally by saying that, for every token a of type A, the following type

$$\prod_{\mathtt{(b,p):E(a)}} \mathtt{Id_{E(a)}}\big((\mathtt{a},\mathtt{refl_a}),(\mathtt{b},\mathtt{p})\big)$$

is inhabited. This is the uniqueness principle for identity types. It does not say that the *only* token of $\sum_{\mathtt{x:A}} \mathtt{Id_A(a,x)}$ is $(\mathtt{a}, \mathtt{refl_a})$, 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 $\sum_{\mathtt{x:A}} \mathtt{Id_A(a,x)}$.

As another way of understanding this, recall from Section 2 that we can read dependent-pair types $\sum_{\mathtt{b:B}} \mathtt{P(b)}$ in different ways: as an existentially quantified proposition, and as a subtype consisting of the tokens of B that satisfy the predicate P, where each such token is accompanied by a certificate to that fact (and bearing in mind the caveat of footnote 20). On this understanding, the based identity type E(a) is the collection of tokens of A satisfying the 'identical to a' predicate. Naïvely we would say that the only token of A that is identical to a is a itself, and so we would expect E(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 $(\mathtt{a}, \mathtt{refl_a})$ that belongs to this type, and a uniqueness principle that says that all tokens are identical to this one. Thus the type E(a) does indeed have just one token, up to identity.[34]

The uniqueness principle for identity types is the first of the two basic principles we introduce.

6.3. The Substitution of Identicals for Identicals

The second basic principle is 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

[34] Alternatively, we could interpret E(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 E(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 E(a) is proved in [HoTT Book, 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.

statement while *saving the truth* of that statement. This 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 $\mathtt{s}$ and $\mathtt{t}$ then anything that is true of one is true of the other. In HoTT, 'something being true of $\mathtt{s}$' corresponds to the existence of a token of type $\mathtt{Q}(\mathtt{s})$ for some predicate $\mathtt{Q}$. Hence we can state the principle as follows: for any type $\mathtt{B}$, any predicate $\mathtt{Q}$ on $\mathtt{B}$, and any $\mathtt{s} : \mathtt{B}$ and $\mathtt{t} : \mathtt{B}$, there is a function of type

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

That is, given an identification of $\mathtt{s}$ with $\mathtt{t}$ and a certificate to a proposition about $\mathtt{t}$, we can produce a certificate to the corresponding proposition about $\mathtt{s}$.[35] We can formalise the quantification over $\mathtt{B}$ by using Π-types: thus, for any type $\mathtt{B}$ and any predicate $\mathtt{Q}$ on $\mathtt{B}$ we have a function

$$\mathtt{ssv}_{\mathtt{B},\mathtt{Q}} : \prod_{\mathtt{s}:\mathtt{B}} \prod_{\mathtt{t}:\mathtt{B}} \mathtt{Id}_{\mathtt{B}}(\mathtt{s},\mathtt{t}) \times \mathtt{Q}(\mathtt{t}) \to \mathtt{Q}(\mathtt{s}).$$

Furthermore, for any $\mathtt{s} : \mathtt{B}$, when the function $\mathtt{ssv}_{\mathtt{B},\mathtt{Q}}(\mathtt{s},\mathtt{s}) : \mathtt{Id}_{\mathtt{B}}(\mathtt{s},\mathtt{s}) \times \mathtt{Q}(\mathtt{s}) \to \mathtt{Q}(\mathtt{s})$ is given $\mathtt{refl}_{\mathtt{s}} : \mathtt{Id}_{\mathtt{B}}(\mathtt{s},\mathtt{s})$ as its first argument, we require that the resulting function of type $\mathtt{Q}(\mathtt{s}) \to \mathtt{Q}(\mathtt{s})$ is the identity function on $\mathtt{Q}(\mathtt{s})$ that leaves its input unchanged. Note that we do *not* require that applying $\mathtt{ssv}_{\mathtt{B},\mathtt{Q}}(\mathtt{s},\mathtt{s})$ to a token of $\mathtt{Id}_{\mathtt{B}}(\mathtt{s},\mathtt{s})$ other than $\mathtt{refl}_{\mathtt{s}}$ gives the identity function on $\mathtt{Q}(\mathtt{s})$. That is, a non-trivial identification of $\mathtt{s}$ with itself may lead to a corresponding non-trivial mapping of $\mathtt{Q}(\mathtt{s})$ to itself. However, substituting $\mathtt{s}$ for itself and using the *trivial* self-identification of $\mathtt{s}$ as the certificate corresponds to making no substitution at all. Thus, whatever token of $\mathtt{Q}(\mathtt{s})$ we had before this trivial substitution, we must have the same token of $\mathtt{Q}(\mathtt{s})$ afterwards.[36]

6.4. Deriving Based Path Induction

From the principle of *substitution salva veritate* we can derive other principles. Recall from Section 3.2.1 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 identical to $\mathtt{t}$ (that is, given an $\mathtt{x}$ it returns the identity type $\mathtt{Id}_{\mathtt{B}}(\mathtt{t},\mathtt{x})$). Thus for any tokens $\mathtt{s},\mathtt{t} : \mathtt{B}$ we have the function

$$\mathtt{ssv}_{\mathtt{B},\,\mathtt{Id}_{\mathtt{B}}(\mathtt{t},-)}(\mathtt{s},\mathtt{t}) : \mathtt{Id}_{\mathtt{B}}(\mathtt{s},\mathtt{t}) \times \mathtt{Id}_{\mathtt{B}}(\mathtt{t},\mathtt{t}) \to \mathtt{Id}_{\mathtt{B}}(\mathtt{t},\mathtt{s}).$$

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:

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

Similarly, for any $\mathtt{u} : \mathtt{B}$, using the predicate $\mathtt{Id}_{\mathtt{B}}(\mathtt{u},-)$ we obtain a function

$$\mathtt{ssv}_{\mathtt{B},\,\mathtt{Id}_{\mathtt{B}}(\mathtt{u},-)}(\mathtt{s},\mathtt{t}) : \mathtt{Id}_{\mathtt{B}}(\mathtt{s},\mathtt{t}) \times \mathtt{Id}_{\mathtt{B}}(\mathtt{u},\mathtt{t}) \to \mathtt{Id}_{\mathtt{B}}(\mathtt{u},\mathtt{s})$$

[35] This principle is called 'transport' in the HoTT Book. Whereas we take it as a basic principle, in the HoTT Book it is derived from path induction [HoTT Book, Lemma 2.3.1].

[36] We could express this formally by a statement saying that, in the case of such a trivial substitution, the token of $\mathtt{Q}(\mathtt{s})$ we get out is judgmentally equal to the token we began with.

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

$$\mathtt{Id}_B(s,t) \times \mathtt{Id}_B(t,u) \to \mathtt{Id}_B(s,u).$$

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 $\sum_{x:A} \mathtt{Id}_A(a,x)$, where a is an arbitrary token of A. Let Q be any predicate on $\mathtt{E}(a)$. The principle of *substitution salva veritate* says that we have a function

$$\mathtt{ssv}_{E(a),Q} : \prod_{s:E(a)} \prod_{t:E(a)} \mathtt{Id}_{E(a)}(s,t) \times Q(t) \to Q(s).$$

Thus, in particular, for any token $(b,p) : \mathtt{E}(a)$ we have a function

$$\mathtt{ssv}_{E(a),Q}\big((b,p),(a,\mathtt{refl}_a)\big) :$$
$$\mathtt{Id}_{E(a)}\big((b,p),(a,\mathtt{refl}_a)\big) \times Q(a,\mathtt{refl}_a) \to Q(b,p).$$

For the token $(a,\mathtt{refl}_a) : \mathtt{E}(a)$ we have the trivial self-identification $\mathtt{refl}_{(a,\mathtt{refl}_a)}$, and when we give this as the first argument to $\mathtt{ssv}_{E(a),Q}\big((a,\mathtt{refl}_a),(a,\mathtt{refl}_a)\big)$, we get a function of type

$$Q(a,\mathtt{refl}_a) \to Q(a,\mathtt{refl}_a)$$

corresponding to a trivial substitution and behaving as the identity function on $Q(a,\mathtt{refl}_a)$ that leaves its input unchanged.

More generally, the uniqueness principle for identity types says that for any token $(b,p) : \mathtt{E}(a)$ the type $\mathtt{Id}_{E(a)}\big((a,\mathtt{refl}_a),(b,p)\big)$ is inhabited. Thus by symmetry of identity so is $\mathtt{Id}_{E(a)}\big((b,p),(a,\mathtt{refl}_a)\big)$, and so we always have a token that can be given as the first argument to $\mathtt{ssv}_{E(a),Q}\big((b,p),(a,\mathtt{refl}_a)\big)$. So for any predicate Q on $\mathtt{E}(a)$ and any token $(b,p) : \mathtt{E}(a)$ we have a function of type

$$Q(a,\mathtt{refl}_a) \to Q(b,p).$$

Thus, if we want to prove that an arbitrary predicate Q holds for all identifications involving $a : A$, we only need to prove that it holds for the trivial self-identification $\mathtt{refl}_a$. The above functions then guarantee that we can extend this proof to all identifications involving a. This is the statement of based path induction.[37]

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

[37] The formal statement of the computation rule as given in [HoTT Book, Section 1.12.1], which involves a judgmental equality, can be recovered from the judgmental equality that we alluded to but did not spell out in footnote 36.

to be foundation in this sense,[38] a necessary condition is that it be *autonomous*, *i.e.*, that its motivation and formulation not depend upon existing advanced mathematics.

We have shown HoTT can be justified 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

Awodey, Steve [2014]: 'Structuralism, invariance, and univalence', *Philosophia Mathematica* (3) **22**, 1–11.

Awodey, Steve, and Michael Warren [2009]: 'Homotopy theoretic models of identity types', *Mathematical Proceedings of the Cambridge Philosophical Society* **146**, 45–55.

Coquand, Thierry [2011]: 'Equality and dependent type theory'. A talk given for the 24th AILA meeting, Bologna, February 2011. http://www.cse.chalmers.se/~coquand/equality.pdf. Accessed May 2015.

Ladyman, James, and Stuart Presnell [forthcoming]: 'A primer on homotopy type theory'. http://www. bristol.ac.uk/media-library/sites/arts/research/homotopy-type/documents/hott-primer-part-one.pdf. Accessed May 2015.

——— [forthcoming]: 'Does homotopy type theory provide a foundation for mathematics?', *British Journal for the Philosophy of Science*.

Linnebo, Øystein, and Richard Pettigrew [2011]: 'Category theory as an autonomous foundation', *Philosophia Mathematica* (3) **19**, 227–254.

Martin-Löf, Per [1975]: 'An intuitionistic theory of types: Predicative part', in H.E. Rose and J.C. Shepherdson, eds, *Logic Colloquium '73. Proceedings of the Logic Colloquium, Bristol, July 1973*, pp. 73–118. Studies in Logic and the Foundations of Mathematics; 80. Amsterdam: North-Holland and New York: American Elsevier.

——— [1996]: 'On the meanings of the logical constants and the justifications of the logical laws', *Nordic Journal of Philosophical Logic* **1**, 11–60.

Mayberry, John P. [1994]: 'What is required of a foundation for mathematics?', *Philosophia Mathematica* (3) **2**, 16–35.

——— [2000]: *The Foundations of Mathematics in the Theory of Sets*. Cambridge University Press.

Paulin-Mohring, Christine [1993]: 'Inductive definitions in the system Coq — rules and properties', in Marc Bezem and Jan Friso Groote, eds, *Typed Lambda Calculi and Applications*, pp. 328–345. Lecture Notes in Computer Science; 664. Springer.

Richman, Fred [1996]: 'Interview with a constructive mathematician', *Modern Logic* **6**, 247–271.

The Univalent Foundations Program Also cited as [HoTT Book] [2013]: *Homotopy Type Theory: Univalent Foundations of Mathematics*. Princeton: Institute for Advanced Study. http://homotopytypetheory.org/book. Accessed May 2015.

[38]We take this issue up in more detail in another paper [Ladyman and Presnell, forthcoming].