

Notions of anonymous existence in Martin-Löf type theory

Kraus, Nicolai ; Escardo, Martin; Coquand, Thierry; Altenkirch, Thorsten

DOI:

[10.23638/LMCS-13\(1:15\)2017](https://doi.org/10.23638/LMCS-13(1:15)2017)

License:

Creative Commons: Attribution-NoDerivs (CC BY-ND)

Document Version

Publisher's PDF, also known as Version of record

Citation for published version (Harvard):

Kraus, N, Escardo, M, Coquand, T & Altenkirch, T 2017, 'Notions of anonymous existence in Martin-Löf type theory', *Logical Methods in Computer Science*, vol. 13, no. 1, 15, pp. 1-36. [https://doi.org/10.23638/LMCS-13\(1:15\)2017](https://doi.org/10.23638/LMCS-13(1:15)2017)

[Link to publication on Research at Birmingham portal](#)

General rights

Unless a licence is specified above, all rights (including copyright and moral rights) in this document are retained by the authors and/or the copyright holders. The express permission of the copyright holder must be obtained for any use of this material other than for purposes permitted by law.

- Users may freely distribute the URL that is used to identify this publication.
- Users may download and/or print one copy of the publication from the University of Birmingham research portal for the purpose of private study or non-commercial research.
- User may use extracts from the document in line with the concept of 'fair dealing' under the Copyright, Designs and Patents Act 1988 (?)
- Users may not further distribute the material nor use it for the purposes of commercial gain.

Where a licence is displayed above, please note the terms and conditions of the licence govern your use of this document.

When citing, please reference the published version.

Take down policy

While the University of Birmingham exercises care and attention in making items available there are rare occasions when an item has been uploaded in error or has been deemed to be commercially or otherwise sensitive.

If you believe that this is the case for this document, please contact UBIRA@lists.bham.ac.uk providing details and we will remove access to the work immediately and investigate.

NOTIONS OF ANONYMOUS EXISTENCE IN MARTIN-LÖF TYPE THEORY*

NICOLAI KRAUS^a, MARTÍN HÖTZEL ESCARDÓ^b, THIERRY COQUAND^c,
AND THORSTEN ALTENKIRCH^d

^{a,d} University of Nottingham, School of Computer Science, Nottingham NG8 1BB, UK
e-mail address: {nicolai.kraus, thorsten.altenkirch}@nottingham.ac.uk

^b University of Birmingham, School of Computer Science, Birmingham B15 2TT, UK
e-mail address: m.escardo@cs.bham.ac.uk

^c Chalmers University, Department of Computer Science and Engineering, SE-412 96 Göteborg, Sweden
e-mail address: thierry.coquand@cse.gu.se

ABSTRACT. As the groupoid model of Hofmann and Streicher proves, identity proofs in intensional Martin-Löf type theory cannot generally be shown to be unique. Inspired by a theorem by Hedberg, we give some simple characterizations of types that do have unique identity proofs. A key ingredient in these constructions are weakly constant endofunctions on identity types. We study such endofunctions on arbitrary types and show that they always factor through a propositional type, the *truncated* or *squashed* domain. Such a factorization is impossible for weakly constant functions in general (a result by Shulman), but we present several non-trivial cases in which it can be done. Based on these results, we define a new notion of anonymous existence in type theory and compare different forms of existence carefully. In addition, we show possibly surprising consequences of the judgmental computation rule of the truncation, in particular in the context of homotopy type theory.

All the results have been formalized and verified in the dependently typed programming language Agda.

Key words and phrases: homotopy type theory, Hedberg’s theorem, anonymous existence, weakly constant functions, factorization, truncation, squash types, bracket types, coherence conditions.

* Part of this article appeared as “Generalizations of Hedberg’s Theorem” in the proceedings of Typed Lambda Calculus and Applications 2013.

^a Supported by the EPSRC grant EP/M016994/1.

^c Supported by the ERC project 247219, and grants of The Ellentuck and The Simonyi Fund.

^d Supported by the EPSRC grants EP/G03298X/1 and EP/M016994/1 and by a grant of the Institute for Advanced Study, as well as by USAF, Airforce office for scientific research, award FA9550-16-1-0029.

1. INTRODUCTION

Although the identity type $\text{Id}(a, b)$ is defined as an inductive type with only one single constructor refl , it is a concept in Martin-Löf type theory [17] [15] [16] that is hard to get intuition for. The reason is that it is, as a type family, parametrized twice over the same type, while the constructor only expects one argument: $\text{refl}_a : a = a$, where $a = b$ is an alternative notation for $\text{Id}(a, b)$. In fact, it is the simplest and most natural occurrence of this phenomenon.

A result by Hofmann and Streicher [7] is that we can not prove refl_a to be the only inhabitant of the type $a = a$, that is, the principle of *unique identity proofs* (UIP) is not derivable. Some time later, Hedberg [6] formulated a sufficient condition on a type to satisfy UIP, namely that its equality is decidable.

The core argument of the proof by Hofmann and Streicher is that types can be interpreted as groupoids, i.e. categories of which all morphisms are invertible. Their conjecture that the construction could also be performed using higher groupoids was only made precise more than ten years later. Awodey and Warren [2] as well as, independently, Voevodsky [29] explained that types can be regarded as, roughly speaking, topological spaces. Consequently, an exciting new direction of constructive formal mathematics attracted researchers from originally very separated areas of mathematics, and *homotopy type theory* [27] was born.

The current article is not only on homotopy type theory, but on Martin-Löf type theory in general, even though we expect that the results are most interesting in the context of homotopy type theory. We start with Hedberg's Theorem [6] and describe multiple simple ways of strengthening it, one of them involving *propositional truncation* [27], also known as *bracket types* [1] or *squash types* [3].

Propositional truncation is a concept that provides a sequel to the *Propositions-as-Types* paradigm [8]. If we regard a type as the correspondent of a mathematical statement, a *proposition*, and its inhabitants of proofs thereof, we have to notice that there is a slightly unsatisfactory aspect. A proof of a proposition in mathematics is usually not thought to contain any information apart from the fact that the proposition is true; however, a type can have any number of inhabitants, and therefore any number of witnesses of its truth. Hence it seems natural to regard only *some* types as propositions, namely those which have at most one inhabitant. The notion of propositional truncation assigns to a type the proposition that this type is inhabited. To make the connection clearer, these types are even called *propositions*, or *h-propositions*, in homotopy type theory. With this in mind, we want to be able to say that a type is inhabited without having to reveal an inhabitant explicitly. This is exactly what propositional truncation $\|-\| : \mathcal{U} \rightarrow \mathcal{U}$ (where we write \mathcal{U} for the universe of types) makes possible. On the other hand, should A have only one inhabitant up to the internal equality, this inhabitant can be constructed from an inhabitant of $\|A\|$. This is a crucial difference between propositional truncation and double negation. We consider a weak version of $\|-\|$ which does not have judgmental computation properties.

After discussing direct generalizations of Hedberg's Theorem, we attempt to transfer the results from the original setting, where they talk about equality types (of *path spaces*), to arbitrary types. This leads to a broad discussion of *weakly constant functions*: we say that $f : A \rightarrow B$ is *weakly constant* if it maps any two elements of A to equal elements of B . The attribute *weakly* comes from the fact that we do not require these actual equality proofs to fulfil further conditions, and a weakly constant function does not necessarily appear to be constant in the topological models. For exactly this reason, it is in general not possible

to factor the function f through $\|A\|$; however, we can do it in certain special cases, and we analyze why. This has, for example, the consequence that the truncated sum of two proposition already has the universal property of their *join*, which is defined as a *higher inductive type* in homotopy type theory.

Particularly interesting are weakly constant *endofunctions*. We show that these can always be factored through the propositional truncation, based on the observation that the type of fixed points of such a function is a proposition. This allows us to define a new notion of existence which we call *populatedness*. We say that A is populated if any weakly constant endofunction on A has a fixed point. This property is propositional and behaves very similar to $\|A\|$, but we show that it is strictly weaker. On the other hand, it is strictly stronger than the double negation $\neg\neg A$, another notion of existence which, however, is often not useful as it generally only allows to prove negative statements. It is worth emphasizing that our populatedness is not a component that has to be *added* to type theory, but a notion that can be *defined* internally. We strongly suspect that this is not the case for even the weak version of propositional truncation, but we lack a formal proof.

It turns out to be interesting to consider the assumption that every type has a weakly constant endofunction. The empty type has a trivial such endofunction, and so does a type of which we know an explicit inhabitant; however, from the assumption that a type has a weakly constant endofunction, we have no way of knowing in which case we are. In a minimalistic theory, we do not think that this assumption implies excluded middle. However, it implies that all equalities are decidable, i.e. a strong version of excluded middle holds for equalities.

Finally, we show that the judgmental computation rule of the propositional truncation, if it is assumed, does have some interesting consequences for the theory. One of our observations is that we can construct a term $\mathsf{myst}_{\mathbb{N}}$ such that $\mathsf{myst}_{\mathbb{N}}(|n|)$ is judgmentally equal to n for any natural number n , which shows that the projection map $|-| : \mathbb{N} \rightarrow \|\mathbb{N}\|$ does not lose meta-theoretic information, in a certain sense.

Some parts of the Sections 3, 4, 6 and 7 of this article have been published in our previous conference paper [13].

Formalization. We have formalized [14] all of our results in the dependently typed programming language and proof assistant *Agda* [19]. It is available in browser-viewable format and as plain source code on the first-named author’s academic homepage. All proofs type-check in Agda version 2.4.2.5.

As most of our results are internal statements in type theory, they can be formalized directly in a readable way, understandable even for readers who do not have any experience with the specific proof assistant or formalized proofs in general. We have tried our best and would like to encourage the reader to have a look at the accompanying formalization.

Contents. In Section 2, we specify the type theory that we work in, a standard version of Martin-Löf type theory. We also state basic definitions, but we try to use standard notation and we hope that all notions are as intuitive as possible. We then revisit Hedberg’s Theorem in Section 3 and formulate several generalizations. Next, we move on to explore weakly constant functions between general types. We show that a weakly constant endofunction has a propositional type of fixed points and factors through $\|-\|$ in Section 4. It is known that the factorization can not always be done for functions between different types,

but we discuss some cases in which it is possible in Section 5. Section 6 is devoted to *populatedness*, a new definable notion of anonymous existence in type theory, based on our previous observations of weakly constant endofunctions. We examine the differences between inhabitation, populatedness, propositional truncation and double negation, all of which are notions of existence, carefully in Section 7. In particular, we show that if every type has a weakly constant endofunction, then all equalities are decidable. Finally, Section 8 discusses consequences of the judgmental computation rule of propositional truncation, and Section 9 presents a summary and questions which we do not know the answer to.

2. PRELIMINARIES

Our setting is a standard version of intensional Martin-Löf type theory (MLTT) with type universes that have coproducts, dependent sums, dependent products and identity types. We give a very rough specification of these constructions below. For a rigorous treatment, we refer to our main reference [27, Appendix A.1 or A.2]. We use standard notation whenever it is available. If it improves the readability, we allow ourselves to implicitly uncurry functions and write $f(x, y)$ instead of $f(x)(y)$ or fxy .

Type Universes. MLTT usually comes equipped with a hierarchy $\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \dots$ of universes, where \mathcal{U}_{n+1} is the type of \mathcal{U}_n . With very few exceptions, we only need one universe \mathcal{U} and therefore omit the index. \mathcal{U} can be understood as a generic universe or, for simplicity, as the lowest universe \mathcal{U}_0 . If we say that X is a type, we mean $X : \mathcal{U}$, possibly in some context.

Coproducts. If X and Y are types, then so is $X + Y$. If we have $x : X$ or $y : Y$, we get $\text{inl } x : X + Y$ or $\text{inr } y : X + Y$, respectively. To prove a statement for all elements in $X + Y$, it is enough to consider those that are of one of these two forms.

Dependent Pairs. If X is a type and $Y : X \rightarrow \mathcal{U}$ a family of types, indexed over X , then $\Sigma_X Y$ is the corresponding *dependent pair type*, sometimes called a *dependent sum* or just Σ -*type*. For $x : X$ and $y : Y(x)$, we have $(x, y) : \Sigma_X Y$, and to eliminate out of $\Sigma_X Y$, it is enough to consider elements of this form. We prefer to write $\Sigma_{x:X} Y(x)$ instead of $\Sigma_X Y$, hoping to increase readability. Instead of $\Sigma_{x_1:X} \Sigma_{x_2:X} Y(x_1, x_2)$, we write $\Sigma_{x_1, x_2:X} Y(x_1, x_2)$. In the special case that Y does not depend on X , it is standard to write $X \times Y$.

Dependent Functions. Given $X : \mathcal{U}$ and $Y : X \rightarrow \mathcal{U}$ as before, we have the type $\Pi_X Y$, called the *dependent functions type* or Π -type. It is sometimes also referred to as the *dependent product type*, although that notion can be confusing as it would fit for Σ -types as well. If, for any given $x : X$, the term t is an element in $Y(x)$, we have $\lambda x. t : \Pi_X Y$. Similarly to $\Pi_X Y$, we write $\Pi_{x:X} Y(x)$, and, if Y does not depend on X , we write $X \rightarrow Y$. Instead of $\Pi_{x_1:X} \Pi_{x_2:X} Y(x_1, x_2)$, we write $\Pi_{x_1, x_2:X} Y(x_1, x_2)$.

Identity Types. Given a type X with elements $x, y : X$, we have the *identity type* or the type of *equalities*, written $x =_X y$. An inhabitant $p : x =_X y$ is thus called an *equality*, an *equality proof*, or, having the interpretation of a type as a space in mind, a *path* from x to y . Similarly, $x =_X y$ is called a *path space*. In the past, p often used to be called a *propositional equality*. We avoid this terminology and reserve the word “propositional” for types with at most one element, as explained in the introduction and in Definition 2.1.

The only introduction rule for the identity types is that, for any $x : X$, there is $\text{refl}_x : x =_X x$. The elimination rule (called J) says that, if $P : (\Sigma_{x,y:X} x =_X y) \rightarrow \mathcal{U}$ is a type family, it suffices to construct an inhabitant of $\Pi_{x:X} P(x, x, \text{refl}_x)$ in order to get an element of $P(p)$ for any $p : \Sigma_{x,y:X} x =_X y$. We do explicitly not assume other elimination

rules such as *Streicher's K* or *uniqueness of identity proofs (UIP)* [24]. If the common type of x, y can be inferred or is unimportant, we write $x = y$ instead of $x =_X y$.

In contrast to the identity type, *definitional* (also called *judgmental*) equality is a meta-level concept. It refers to two terms, rather than two (hypothetical) elements, with the same β (and, sometimes, η in a restricted sense) normal form. Recently, it has become standard to use the symbol \equiv for judgmental equality in order to use $=$ solely for the type of equalities [27]. Note that the introduction rule of the latter says precisely that we have a canonical equality proof for any two judgmentally equal terms, viewed as elements of some type. For definitions, we use the notation $:\equiv$.

Applying the eliminator J is also referred to as *path induction* [27]. A variant of J that is sometimes more useful is due to Paulin-Mohring [21]: given a point $x : X$ and a type family $P : (\Sigma_{y:X} x =_X y) \rightarrow \mathcal{U}$, it is enough to construct an inhabitant of $P(x, \text{refl}_x)$ in order to construct an inhabitant of $P(y, q)$ for any pair (y, q) . This elimination principle, called *based path induction*, is equivalent to J .

As a basic example, we show that equality proofs satisfy the *groupoid laws* [7], where reflexivity plays the role of identity morphisms. If we have $p : x =_X y$ and $q : y =_X z$, we can construct a path $p \cdot q : x =_X z$ (the *composition* of p and q): by based path induction, it is enough to do this under the assumption that $(z, q) : \Sigma_{z:X} y =_X z$ is (y, refl_y) . But in that case, the composition $p \cdot q$ is given by p . Similarly, for $p : x =_X y$, there is $p^{-1} : y =_X x$. It is easy to see (again by path induction) that the types $p \cdot \text{refl}_y =_X p$ and $\text{refl}_x \cdot p =_X p$ as well as $p \cdot p^{-1} =_X \text{refl}_x$ are inhabited, and similarly, so are all the other types that are required to give a type the structure of a groupoid.

An important special case of the eliminator J is *substitution* or *transportation*: if $P : X \rightarrow \mathcal{U}$ is a family of types and $x, y : X$ are two elements (or points) that are equal by $p : x =_X y$, then an element of $e : P(x)$ can be *transported along the path* p to get an element of $P(y)$, written

$$p_*(e) : P(y). \quad (2.1)$$

Another useful function, similarly easily derived from J , is the following: if $f : X \rightarrow Y$ is a function and $p : x =_X y$ a path, we get an inhabitant of $f(x) = f(y)$ in Y ,

$$\text{ap}_f p : f(x) = f(y). \quad (2.2)$$

Note that we omit the arguments x and y in the notation of ap_f .

Identity types also enable us to talk about isomorphism, or (better) *equivalence*, of types. We say that X and Y are equivalent, written $X \simeq Y$, if there are functions in both directions which are the inverses of each other,

$$f : X \rightarrow Y \quad (2.3)$$

$$g : Y \rightarrow X \quad (2.4)$$

$$p : \prod_{x:X} g(f(x)) =_X x \quad (2.5)$$

$$q : \prod_{y:Y} f(g(y)) =_Y y. \quad (2.6)$$

Technically, (f, g, p, q) only constitute what is usually called a *type isomorphism*, but from any such isomorphism, an equivalence (in the sense of homotopy type theory) can be constructed; and the only difference is that an equivalence requires a certain coherence between the components p and q , which will not be important for us. In this sense, we do not distinguish between isomorphisms and equivalences, and only choose the latter terminology on principle. For details, we refer to [27, Chapter 4]. We call types *logically equivalent*, written $X \Leftrightarrow Y$, if

there are functions in both directions (that is, we only have the components (2.3) and (2.4)). We write $X \Leftrightarrow Y \Leftrightarrow Z$ if X, Y, Z are pairwise logically equivalent, and $X \Rightarrow Y \Rightarrow Z$ as a shorthand notation for $(X \rightarrow Y) \times (Y \rightarrow Z)$.

Equivalent types share all internalizable properties. In fact, Voevodsky's univalence axiom (e.g. [27], [29]) has the consequence that equivalent types are equal. For the biggest part of our article, we do not need to assume the univalence axiom; however, it will play some role in Section 8.

We sometimes use other additional principles (namely function extensionality and propositional truncation, as introduced later). However, we treat them as assumptions rather than parts of the core theory and state clearly in which cases they are used.

In order to support the presentation from the next section on, we define a couple of notions. Our hope is that all of these are as intuitive as possible, if not already known. The only notion that is possibly ambiguous is *weak constancy*, meaning that a function maps any pair of possible arguments to equal values.

Definition 2.1. We say that a type X is *propositional*, or is a *proposition*, if all its inhabitants are equal:

$$\text{isProp } X \equiv \Pi_{x,y:X} x = y. \quad (2.7)$$

It is a well-known fact that the path spaces of a propositional type are not only inhabited but also propositional themselves. This stronger property is called *contractible*,

$$\text{isContr } X \equiv X \times \text{isProp } X. \quad (2.8)$$

It is easy to see that any contractible type is equivalent to the unit type. An important well-known lemma is that types are contractible if they are represented as *singletons*, sometimes called *path-to/path-from* types: for any $a_0 : A$, the type

$$\Sigma_{a:A} a_0 = a \quad (2.9)$$

is contractible, as any inhabitant is by based path induction easily seen to be equal to (a_0, refl_{a_0}) .

Further, X satisfies *UIP*, or is a *set*, if its path spaces are all propositional:

$$\text{isSet } X \equiv \Pi_{x,y:X} \text{isProp}(x = y). \quad (2.10)$$

X is *decidable* if it is either inhabited or empty,

$$\text{decidable } X \equiv X + \neg X. \quad (2.11)$$

We therefore say that X has *decidable equality* if the equality type of any two inhabitants of X is decidable. Based on the terminology in [18], we also call a type with decidable equality *discrete*:

$$\text{isDiscrete } X \equiv \Pi_{x,y:X} \text{decidable}(x = y). \quad (2.12)$$

A function (synonymously, map) $f : X \rightarrow Z$ is *weakly constant*, or *1-constant*, if it maps any two elements to the same inhabitant of Z :

$$\text{wconst } f \equiv \Pi_{x,y:X} f(x) = f(y). \quad (2.13)$$

As weak (or 1-) constancy is the only notion of constancy that we consider in this article (if we ignore factorizability through $\|- \|$), we call such a function f just *constant* for simplicity. However, note that this notion is indeed very weak as soon as we consider functions into

types that are not sets, as we will see later. It will be interesting to consider the type of constant endomaps on a given type:

$$\text{constEndo } X := \Sigma_{f:X \rightarrow X} \text{wconst } f. \quad (2.14)$$

Finally, we may say that X has constant endomaps on all path spaces:

$$\text{pathConstEndo } X := \Pi_{x,y:X} \text{constEndo } (x = y). \quad (2.15)$$

For some statements, but only if clearly indicated, we use *function extensionality*. This principle says that two functions f, g of the same type $\Pi_X Y$ are equal as soon as they are pointwise equal:

$$(\Pi_{x:X} f(x) = g(x)) \rightarrow f = g. \quad (2.16)$$

An important equivalent formulation due to Voevodsky [28] is that the type of propositions is closed under Π ; more precisely,

$$(\Pi_{x:X} \text{isProp } (Y x)) \rightarrow \text{isProp } (\Pi_X Y). \quad (2.17)$$

In the case of non-dependent functions, this means that $X \rightarrow Y$ is propositional as soon as Y is.

A principle that we do not assume, but which will appear in some of our discussions, is the *law of excluded middle* in the form for propositions and in the form for general types [27, Chapter 3.4]. In the first form, it says that every proposition is decidable, while the second says the the same without the restriction to propositions.

$$\text{LEM} := \Pi_{P:\mathcal{U}} (\text{isProp } P) \rightarrow P + \neg P \quad (2.18)$$

$$\text{LEM}_\infty := \Pi_{X:\mathcal{U}} X + \neg X. \quad (2.19)$$

Note that LEM_∞ can be considered the natural formulation under the *Propositions-as-Types* view. However, the view that we adapt in this work (as in homotopy type theory) is the one that only type-theoretical propositions in the sense of Definition 2.1 really represent mathematical propositions; general types carry more structure. In particular, LEM_∞ includes a very strong form of choice which is inconsistent with the univalence axiom of homotopy type theory. Therefore, we consider LEM the “correct” formulation in our work.

We do not explicitly use this fact, but it may be helpful to note that, assuming function extensionality, all of the above definitions that are called “is...” ($\text{isProp } X$, $\text{isContr } X$, $\text{isSet } X$, $\text{isDiscrete } X$) are propositional in the sense of Definition 2.1. For $\text{isProp } X$, $\text{isContr } X$, $\text{isSet } X$, this is proved in [27, Theorem 7.1.10], and for $\text{isDiscrete } X$, this is a consequence of Hedberg’s Theorem that we discuss in Section 3. It will also follow that $\text{pathConstEndo } X$ is propositional. The statements of LEM and (2.17) are propositional as well, while $\text{wconst } f$, $\text{constEndo } X$, (2.16), and LEM_∞ are in general not propositional.

3. HEDBERG’S THEOREM

Before discussing possible generalizations, we want to state Hedberg’s Theorem.

Theorem 3.1 (Hedberg [6]). *Every discrete type satisfies UIP,*

$$\text{isDiscrete } X \rightarrow \text{isSet } X. \quad (3.1)$$

We briefly give Hedberg’s original proof, consisting of two steps.

Lemma 3.2. *If a type has decidable equality, its path spaces have constant endofunctions:*

$$\text{isDiscrete } X \rightarrow \text{pathConstEndo } X. \quad (3.2)$$

Proof. Given inhabitants x and y of X , we get by assumption either an inhabitant of $x = y$ or an inhabitant of $\neg(x = y)$. In the first case, we construct the required constant function $(x = y) \rightarrow (x = y)$ by mapping everything to this given path. In the second case, we have a proof of $\neg(x = y)$, and the canonical function is constant automatically. \square

Lemma 3.3. *If the path spaces of a type have constant endomaps, the type satisfies UIP:*

$$\text{pathConstEndo } X \rightarrow \text{isSet } X. \quad (3.3)$$

Proof. Assume f is a parametrized constant endofunction on the path spaces, meaning that, for any $x, y : X$, we have a constant function $f_{x,y} : x = y \rightarrow x = y$. Let p be a path from x to y . We claim that

$$p = (f_{x,x}(\text{refl}_x))^{-1} \cdot f_{x,y}(p). \quad (3.4)$$

By path induction, we only have to give a proof if the triple (x, y, p) is in fact (x, x, refl_x) , in which case (3.4) is one of the groupoid laws that equality satisfies. Using the fact f is constant on every path space, the right-hand side of the above equality is independent of p , and in particular, equal to any other path of the same type. \square

Hedberg’s proof [6] is just the concatenation of the two lemmata. A slightly more direct proof can be found in the HoTT Coq repository [26] and in a post by the first named author on the HoTT blog [9].

Let us analyse the ingredients of the original proof. Lemma 3.2 uses the rather strong assumption of decidable equality. In contrast, the assumption of Lemma 3.3 is logically equivalent to its conclusion, so that there is no space for a strengthening. We include a proof of this simple claim in Theorem 3.10 below and concentrate on weakening the assumption of Lemma 3.3. Let us first introduce the notions of *stability* and *separatedness*.

Definition 3.4. For a type X , define

$$\text{stable } X := \neg\neg X \rightarrow X, \quad (3.5)$$

$$\text{separated } X := \prod_{x,y:X} \text{stable}(x = y). \quad (3.6)$$

We can see $\text{stable } X$ as a classical condition, similar to $\text{decidable } X \equiv X + \neg X$, but strictly weaker. Indeed, we get a first strengthening of Hedberg’s Theorem as follows:

Lemma 3.5 ([27, Corollary 7.2.3]). *Assuming function extensionality, any separated type is a set,*

$$\text{separated } X \rightarrow \text{isSet } X. \quad (3.7)$$

Proof. There is, for any $x, y : X$, a canonical map $(x = y) \rightarrow \neg\neg(x = y)$. Composing this map with the proof that X is separated yields an endofunction on the path spaces. With function extensionality, the first map has a propositional codomain, implying that the endofunction is constant and thereby fulfilling the requirements of Lemma 3.3. \square

We remark that full function extensionality is actually not needed here. Instead, a weaker version that only works with the empty type is sufficient. Similar statements hold true for all further applications of extensionality in this paper.

In a constructive setting, the question how to express that “there exists something” in a type X is very subtle. One possibility is to ask for an inhabitant of X , but in many cases, this is too strong to be fulfilled. A second possibility, which corresponds to our above definition of *separated*, is to ask for a proof of $\neg\neg X$. Then again, this is very weak, and

often too weak, as one can in general only prove negative statements from double-negated assumptions.

This fact has inspired the introduction of *squash types* (Constable [3]), and similar, *bracket types* (Awodey and Bauer [1]). These lie in between of the two extremes mentioned above. In our intensional setting, we talk of *propositional truncations*, or *-1-truncations* [27, Chapter 3.7]. For any type X , we postulate that there is a type $\|X\|$ that is a *proposition*, representing the statement that X is inhabited. The rules are that if we have a proof of X , we can, of course, get a proof of $\|X\|$, and from $\|X\|$, we can conclude the same statements as we can conclude from X , but only if the actual representative of X does not matter:

Definition 3.6. We say that a type theory has *weak propositional truncations* if, for every type X , we have a type $\|X\| : \mathcal{U}$ which satisfies the following properties:

- (i) $|-| : X \rightarrow \|X\|$
- (ii) $\mathbf{h}_{\text{tr}} : \text{isProp}(\|X\|)$
- (iii) $\text{rec}_{\text{tr}} : \Pi_{P:\mathcal{U}} \text{isProp } P \rightarrow (X \rightarrow P) \rightarrow \|X\| \rightarrow P$.

Note that this amounts to saying that the operator $\|-|$ is left adjoint to the inclusion of the subcategory of propositions into the category of all types. Therefore, it can be seen as the *propositional reflection*. For $x, y : \|X\|$, we will write $\mathbf{h}_{\text{tr},x,y}$ for the proof of $x =_{\|X\|} y$ that we get from \mathbf{h}_{tr} .

In contrast to other sources [27] we do *not* assume the judgmental β -rule

$$\text{rec}_{\text{tr}}(P, h, f, |x|) \equiv_{\beta} f(x) \quad (3.8)$$

as it is simply not necessary for our results and we do not want to make the theory stronger than required. This is the reason why we use the attribute *weak*. We do think that (3.8) is often useful, but we also think it is interesting to make clear in which sense (3.8) makes the theory actually stronger, rather than more convenient. We will discuss this in Section 8. A practical advantage of not assuming (3.8) is that the truncation can be implemented in existing proof assistants more easily. Of course, the β -rule holds propositionally as both sides of the equation inhabit the same proposition.

Adopting the terminology of [27, Chapter 3.10], we say that X is *merely* inhabited if $\|X\|$ is inhabited. We may also say that X *merely* holds. However, we try to always be precise by giving the formal type expression to support the informal statement.

The *non-dependent eliminator* (or *recursion principle*, see [27, Chapter 5.1]) rec_{tr} lets us construct the *dependent* one (the *induction principle*):

Lemma 3.7 (see [27, Exercise 3.17]). *The propositional truncation admits the following induction principle: Given a type X , a family $P : \|X\| \rightarrow \mathcal{U}$ with $h : \Pi_{z:\|X\|} \text{isProp}(P(z))$, a term $f : \Pi_{x:X} P(|x|)$ gives rise to an inhabitant of $\Pi_{z:\|X\|} P(z)$.*

Proof. We have a map $j : X \rightarrow \Sigma_{z:\|X\|} P(z)$ by $\lambda x. (|x|, f(x))$. Observe that the codomain of j is a proposition, combining the fact that $\|X\|$ is one with h . Therefore, we get $\|X\| \rightarrow \Sigma_{z:\|X\|} P(z)$, and this is sufficient, using that $y =_{\|X\|} z$ for any $y, z : \|X\|$. \square

In analogy to the notation rec_{tr} , we may write ind_{tr} for the term witnessing this induction principle. However, most of our further developments will not require the induction principle and will be proved with rec_{tr} .

Note that $\|-|$ is functorial in the sense that any function $f : X \rightarrow Y$ gives rise to a function $\|f\| : \|X\| \rightarrow \|Y\|$, although the proof of $\|g \circ f\| = \|g\| \circ \|f\|$ requires function

extensionality. It is easy to see that $\|- \|$ is a modality (an idempotent monad) in the sense of [27, Chapter 7.7]. In particular, we have $\|\|X\|\| \simeq \|X\|$.

It is well-known that there is a type expression which is logically equivalent to the propositional truncation:

Theorem 3.8. *For any given $X : \mathcal{U}$, we have the logical equivalence*

$$\|X\| \Leftrightarrow \Pi_{P:\mathcal{U}} \text{isProp } P \rightarrow (X \rightarrow P) \rightarrow P. \quad (3.9)$$

Under the assumption of function extensionality, the expression on the right-hand side of (3.9) is propositional, and the logical equivalence (\Leftrightarrow) is thus an actual equivalence. A potential problem with this expression is that it does not live in the universe \mathcal{U} . This size issue is the only thing that keeps us from using it as the definition for $\|X\|$. All other properties of the above Definition 3.6 are satisfied, at least under the assumption of function extensionality. Voevodsky [28] suggests *resizing rules* to resolve the issue.

Proof of Theorem 3.8. The direction “ \rightarrow ” of the statement is no more than a rearrangement of the assumptions of property (iii) in the definition of $\|X\|$. For the other direction, we only need to instantiate P with $\|X\|$ and observe that the properties (i) and (ii) are exactly what is needed. \square

With this definition at hand, we can provide an even stronger variant of Hedberg’s Theorem. Completely analogously to the notions of stability and separatedness, we define what it means to say that a type has *split support* and is *h-separated*:

Definition 3.9. For a type X , define

$$\text{splitSup } X \equiv \|X\| \rightarrow X, \quad (3.10)$$

$$\text{hSeparated } X \equiv \Pi_{x,y:X} \text{splitSup}(x = y). \quad (3.11)$$

We observe that $\text{hSeparated } X$ is a weaker condition than $\text{separated } X$. Not only can we conclude $\text{isSet } X$ from $\text{hSeparated } X$, but the converse holds as well. In the following theorem, we also include the simple fact that having constant endomaps on path spaces is equivalent to these statements.

Theorem 3.10. *For a type X in MLTT with propositional truncation, the following are equivalent:*

- (i) X is a set
- (ii) X has constant endomaps on its path spaces
- (iii) X is h-separated.

Further, each of the three types is propositional.

Proof. We first show the logical equivalence of the three types. “(ii) \Rightarrow (i)” is simply Lemma 3.3. “(i) \Rightarrow (iii)” uses simply the definition of the propositional truncation: given $x, y : X$, the fact that X is a set tells us exactly that $x = y$ is propositional, implying that we have a map $\|x = y\| \rightarrow (x = y)$. Concerning “(iii) \Rightarrow (ii)”, it is enough to observe that the composition of $|-| : (x = y) \rightarrow \|x = y\|$ and the map $\|x = y\| \rightarrow (x = y)$, provided by the fact that X is h-separated, is a parametrized constant endofunction.

(i) is known to be a proposition. If (ii) or (iii) are inhabited, then X is a set, implying that (ii) and (iii) are propositions. \square

We observe that using propositional truncation in some cases makes it unnecessary to appeal to functional extensionality. In Lemma 3.5, we have given a proof for the simple statement that separated types are sets in the context of function extensionality. Let us now drop function extensionality and assume instead that propositional truncation is available. Every separated type is h-separated — more generally, we have

$$(\neg\neg X \rightarrow X) \rightarrow (\|X\| \rightarrow X) \quad (3.12)$$

for a type X — and every h-separated space is a set. Notice that $\neg X \rightarrow \neg\|X\|$ and thus also $\|X\| \rightarrow \neg\neg X$ and (3.12) do not require function extensionality. Therefore, the mere availability of propositional truncation suffices to solve a gap that function extensionality would usually fill. In Section 8.2 below, we will see that propositional truncation with the judgmental β -rule (3.8) makes it possible to derive function extensionality.

A variant of Theorem 3.10, more precisely of the direction “(iii) \Rightarrow (i)”, can be formulated without propositional truncation. We say that a *reflexive propositionally-valued relation* on X is an $R : X \times X \rightarrow \mathcal{U}$ such that $R(x, y)$ is always propositional and $R(x, x)$ always contractible. If R implies identity, that is $\prod_{x, y : X} R(x, y) \rightarrow x = y$, then X is a set. This is a statement given in the standard text book on homotopy type theory [27, Theorem 7.2.2] and is sometimes called “Rijke’s Theorem”.

To conclude this part of the article, we want to mention that there is a slightly stronger version of Hedberg’s Theorem which applies to types where equality might only be decidable *locally*. In fact, nearly everything we stated or proved can be done locally, and thus made stronger. In the proof of Lemma 3.2, we have not made use of the fact that we were dealing with path spaces at all: any decidable type trivially has a constant endofunction. Concerning Lemma 3.3, we observe:

Lemma 3.11 (Local form of Lemma 3.3). *A type X that locally has constant endomaps on path spaces does locally satisfy UIP. That means, for any $x_0 : X$, we have*

$$(\prod_{y : X} \text{constEndo}(x_0 = y)) \rightarrow \prod_{y : X} \text{isProp}(x_0 = y). \quad (3.13)$$

Proof. The proof is identical to the one of Lemma 3.3, with the only difference that we need to apply based path induction instead of path induction. \square

This enables us to prove the local variant of Hedberg’s Theorem:

Theorem 3.12 ([20],[9]; Local form of Theorem 3.1). *A locally discrete type X is locally a set, i.e. for any $x_0 : X$,*

$$(\prod_{y : X} \text{decidable}(x_0 = y)) \rightarrow \prod_{y : X} \text{isProp}(x_0 = y). \quad (3.14)$$

\square

In the same simple way, we immediately get that the assumption of local separatedness is sufficient.

Lemma 3.13 (Local form of Lemma 3.5). *Under the assumption of function extensionality, a locally separated type locally is a set, i.e. for any $x_0 : X$,*

$$(\prod_{y : X} \text{stable}(x_0 = y)) \rightarrow \prod_{y : X} \text{isProp}(x_0 = y). \quad (3.15)$$

\square

Similarly, the local forms of the characterizations of Theorem 3.10 are still equivalent.

Theorem 3.14 (Local form of Theorem 3.10). *For a type X in MLTT with propositional truncation with a point $x_0 : X$, the following are equivalent:*

- (i) *for all $y : X$, the type $x_0 = y$ is propositional*
- (ii) *for all $y : X$, the type $x_0 = y$ has a constant endomap*
- (iii) *for all $y : X$, the type $x_0 = y$ has split support.*

□

Note that most of our arguments can be generalized to higher truncation levels [27, Chapter 7] in a reasonable and straightforward way. Details can be found in the first-named author's PhD thesis [12].

4. SPLIT SUPPORT FROM CONSTANT ENDOFUNCTIONS

If we unfold the definitions in the statements of Theorem 3.10, they all involve the path spaces over some type X :

- (i) $\prod_{x,y:X} \text{isProp}(x = y)$
- (ii) $\prod_{x,y:X} \text{constEndo}(x = y)$
- (iii) $\prod_{x,y:X} \text{splitSup}(x = y)$.

We have proved that these statements are logically equivalent. It is a natural question to ask whether this is true for types that are not necessarily path spaces. The possibilities that path spaces offer are very powerful and we have used them heavily. Indeed, if we formulate the above properties for an arbitrary type A instead of path types,

- (i) $\text{isProp } A$
- (ii) $\text{constEndo } A$
- (iii) $\text{splitSup } A$,

we notice immediately that (i) is significantly and strictly stronger than the other two properties. (i) says that A has at most one inhabitant, (ii) says that there is a constant endofunction on A , and (iii) gives us a possibility to get an explicit inhabitant of A from the proposition that A has an anonymous inhabitant. A propositional type has the other two properties trivially, while the converse is not true. In fact, as soon as we know an inhabitant $a : A$, we can very easily construct proofs of (ii) and (iii), while it does not help at all with (i).

The implication (iii) \Rightarrow (ii) is also simple: if we have $h : \|A\| \rightarrow A$, the composition $h \circ |-| : A \rightarrow A$ is constant, as for any $a, b : A$, we have $|a| = |b|$ and therefore $h(|a|) = h(|b|)$.

In summary, we have (i) \Rightarrow (iii) \Rightarrow (ii) and we know that the first implication cannot be reversed. What is less clear is the reversibility of the second implication: If we have a constant endofunction on A , can we get a map $\|A\| \rightarrow A$? Put differently, what does it take to get out of $\|A\|$? Of course, a proof that A has split support is fine for that, but does a constant endomap on A also suffice? Surprisingly, the answer is positive, and there are interesting applications (Section 6). The main ingredient of our proof, and of much of the rest of the paper, is the following crucial lemma about fixed points:

Lemma 4.1 (Fixed Point Lemma). *Given a constant endomap f on a type X , the type of its fixed points is propositional, where this type is defined by*

$$\text{fix } f \equiv \Sigma_{x:X} x = f(x). \quad (4.1)$$

Before we can give the proof, we first need to formulate two observations. Both of them are simple on their own, but important insights for the Fixed Point Lemma. Let X and Y be two types.

Auxiliary Lemma 4.2 ([27, Theorem 2.11.3]). *Assume $h, k : X \rightarrow Y$ are two functions and $t : x = y$ as well as $p : h(x) = k(x)$ are paths. Then, transporting along t into p can be expressed as a composition of paths:*

$$t_*(p) = (\mathbf{ap}_h t)^{-1} \cdot p \cdot \mathbf{ap}_k t. \quad (4.2)$$

Proof. This is immediate by path induction on t . \square

Even if the latter proof is trivial, the statement is essential. In the proof of Lemma 4.1, we need a special case where x and y are the same. However, this special version cannot be proved directly. We consider the second observation a key insight for the Fixed Point Lemma:

Auxiliary Lemma 4.3. *If $f : X \rightarrow Y$ is constant and $x_1, x_2 : X$ are points, then $\mathbf{ap}_f : x_1 =_X x_2 \rightarrow f(x_1) =_Y f(x_2)$ is constant. In particular, \mathbf{ap}_f maps every loop around x (that is, path from x to x) to $\mathbf{refl}_{f(x)}$.*

Proof. If c is the proof of $\mathbf{wconst} f$, then \mathbf{ap}_f maps a path $p : x = y$ to $c(x, x)^{-1} \cdot c(x, y)$. This is easily seen to be correct for (x, x, \mathbf{refl}_x) , which is enough to apply path induction. As the expression is independent of p , the function \mathbf{ap}_f is constant. The second part follows from the fact that \mathbf{ap}_f maps \mathbf{refl}_x to $\mathbf{refl}_{f(x)}$. \square

With these lemmata at hand, we give a proof of the Fixed Point Lemma:

Proof of Lemma 4.1. Assume $f : X \rightarrow X$ is a function and $c : \mathbf{wconst} f$ is a proof that it is constant. For any two pairs (x, p) and $(x', p') : \mathbf{fix} f$, we need to construct a path connection them.

First, we simplify the situation by showing that we can assume that x and x' are the same: By composing $p : x = f x$ with $c(x, x') : f(x) = f(x')$ and $(p')^{-1} : f(x') = x'$, we get a path $p'' : x = x'$. By a standard lemma [27, Theorem 2.7.2], a path between two pairs corresponds to two paths: One path between the first components, and one between the second, where transporting along the first path is needed. We therefore now get that $(x, (p'')^{-1} \cdot p')$ and (x', p') are equal: p'' is a path between the first components, which makes the second component trivial. Write q for the term $(p'')^{-1} \cdot p'$.

We are now in the (nicer) situation that we have to construct a path between (x, p) and $(x, q) : \mathbf{fix} f$. Again, such a path can be constructed from two paths for the two components. Let us assume that we use some path $t : x = x$ for the first component. We then have to show that $t_*(p)$ equals q . In the situation with (x, p) and (x', p') , it might have been tempting to use p'' as a path between the first components, and that would correspond to choosing \mathbf{refl}_x for t . However, one quickly convinces oneself that this cannot work in the general case.

By Auxiliary Lemma 4.2, with the identity for h and f for k , the first of the two terms, i. e. $t_*(p)$, corresponds to $t^{-1} \cdot p \cdot \mathbf{ap}_f t$. With Auxiliary Lemma 4.3, that term can be further simplified to $t^{-1} \cdot p$. What we have to prove is now just $t^{-1} \cdot p = q$, so let us just choose $p \cdot q^{-1}$ for t , thereby making it into a straightforward application of the standard lemmata. \square

A more elegant but possibly less revealing proof of the Fixed Point Lemma was given by Christian Sattler:

Second Proof of Lemma 4.1 (Sattler). Given $f : X \rightarrow X$ and $c : \mathbf{wconst} f$ as before, assume $(x_0, p_0) : \mathbf{fix} f$. For any $x : X$, we have an equivalence of types,

$$f(x) = x \simeq f(x_0) = x, \quad (4.3)$$

given by precomposition with $c(x_0, x)$. Therefore, we also have the equivalence

$$\Sigma_{x:X} f(x) = x \simeq \Sigma_{x:X} f(x_0) = x. \quad (4.4)$$

The second of these types is a singleton and thus contractible, while the first is just $\mathbf{fix} f$. This shows that any other inhabitant of $\mathbf{fix} f$ is indeed equal to (x_0, p_0) . \square

We will exploit Lemma 4.1 in different ways. For the following corollary note that, given an endomap f on X with constancy proof c , we have a canonical projection

$$\mathbf{fst} : \mathbf{fix} f \rightarrow X \quad (4.5)$$

and a function

$$\epsilon : X \rightarrow \mathbf{fix} f \quad (4.6)$$

$$\epsilon(x) := (f(x), c(x, f(x))). \quad (4.7)$$

Corollary 4.4. *In basic MLTT, for a type X with a constant endofunction f , the type $\mathbf{fix} f$ is a proposition that is logically equivalent to X . In particular, $\mathbf{fix} f$ satisfies the conditions (i)–(iii) of Definition 3.6. Therefore, for a type with a constant endomap, the weak propositional truncation is actually definable. If $\|- \|$ is part of the theory, $\|X\|$ and $\mathbf{fix} f$ are equivalent, $\|X\| \simeq \mathbf{fix} f$. \square*

We are now in the position to prove the statement that we have announced at the beginning of the section.

Theorem 4.5. *A type X has a constant endomap if and only if it has split support in the sense that $\|X\| \rightarrow X$.*

Proof. As already mentioned in earlier, the “if”-part is simple: given $\|X\| \rightarrow X$, we just need to compose it with $|-| : X \rightarrow \|X\|$ to get a constant endomap. The other direction is an immediate consequence of Corollary 4.4. \square

We want to add the remark that $\mathbf{constEndo} X$ can be replaced by a seemingly weaker assumption. The following statement (together with the Theorem 4.5) shows that it is enough to have $f : X \rightarrow X$ which is *merely* constant:

Theorem 4.6. *For a type X , the following are logically equivalent:*

- (i) *X has a constant endomap*
- (ii) *X has an endomap f with a proof $\|\mathbf{wconst} f\|$.*

The first direction is trivial, but its reversibility is interesting. We do *not* think that $\|\mathbf{wconst} f\|$ allows us to construct an element of $\mathbf{wconst} f$.

Proof of the nontrivial direction of Theorem 4.6. Assume f is an endofunction on X . From Lemma 4.1, we know that

$$\mathbf{wconst} f \rightarrow \mathbf{isProp}(\mathbf{fix} f). \quad (4.8)$$

Using the recursion principle with the fact that the statement $\text{isProp}(\text{fix } f)$ is a proposition itself yields

$$\|\text{wconst } f\| \rightarrow \text{isProp}(\text{fix } f). \quad (4.9)$$

Previously, we have constructed a map

$$\text{wconst } f \rightarrow \|X\| \rightarrow \text{fix } f. \quad (4.10)$$

Let us write this function as

$$\|X\| \rightarrow \text{wconst } f \rightarrow \text{fix } f. \quad (4.11)$$

This makes it trivial to define a function

$$\|X\| \times \|\text{wconst } f\| \rightarrow \text{wconst } f \rightarrow \text{fix } f. \quad (4.12)$$

We assume $\|X\| \times \|\text{wconst } f\|$. From (4.9), we conclude that $\text{fix } f$ is a proposition. Therefore, we may apply the recursion principle of the truncation and get

$$\|X\| \times \|\text{wconst } f\| \rightarrow \|\text{wconst } f\| \rightarrow \text{fix } f, \quad (4.13)$$

which, of course, gives us

$$\|X\| \rightarrow \text{fix } f \quad (4.14)$$

under the assumption (ii) of the theorem. Composing $|-|$ with (4.14) and with the first projection, we get a constant function $g : X \rightarrow X$. \square

It seems to be impossible to show that the constructed function g is equal to f . On the other hand, it is easy to prove the truncated version of this statement:

$$\|\Pi_{x:X} f x = g x\|. \quad (4.15)$$

The detailed proof can be found in our formalization [14].

5. FACTORING WEAKLY CONSTANT FUNCTIONS

In Theorem 4.5 we have seen that a type X with a constant function $f : X \rightarrow X$ always has split support. In fact, what we have done is actually slightly more: the constructed map $\bar{f} : \|X\| \rightarrow X$ has the property that the triangle

$$\begin{array}{ccc} X & \xrightarrow{f} & X \\ & \searrow \scriptstyle |-| & \nearrow \scriptstyle \bar{f} \\ & \|X\| & \end{array}$$

commutes pointwise (in the sense that we have a family of equality proofs).

It seems a natural question to ask whether the fact that f is an endofunction is required: given a (weakly) constant function $f : X \rightarrow Y$, can it be factored in this sense through $\|X\|$? With Theorem 4.5 in mind, it may be surprising that the answer is negative. In the presence of univalence, Shulman has constructed a family of weakly constant functions such that it is impossible that all of them factor [23]. From another result by the first-named author, it follows that functions $\|X\| \rightarrow Y$ can be constructed from *coherently* constant functions $X \rightarrow Y$, where the proof of weak constancy comes with a tower of coherence conditions [11]. However, there are special cases in which the factorization is possible only assuming weak constancy, and some of these are discussed in the current section.

Let us start by giving a precise definition.

Definition 5.1. Given a function $f : X \rightarrow Y$ between two types, we say that f *factors* through a type Z if there are functions $f_1 : X \rightarrow Z$ and $f_2 : Z \rightarrow Y$ such that

$$\prod_{x:X} f_2(f_1(x)) =_Y f(x). \quad (5.1)$$

In particular, we say that f factors through $\|X\|$ if there is a function $\bar{f} : \|X\| \rightarrow Y$ such that

$$\prod_{x:X} \bar{f}(|x|) =_Y f(x). \quad (5.2)$$

As we will discuss later, assuming judgmental computation for $\|- \|$, a factorization in the above sense allows us to construct a judgmental factorization (see Section 8).

A related known result is that any function $f : X \rightarrow Y$ factors through its *image* (see [27, Chapter 7.6]), where the image $\text{im}(f)$ is defined as

$$\text{im}(f) := \Sigma_{y:Y} \|\Sigma_{x:X} f(x) =_Y y\|. \quad (5.3)$$

If $\text{im}(f)$ is propositional, this answers positively the question that we want to discuss. We will see that this is what happens if Y is a set and f is constant (Theorem 5.4). However, in general, $\text{im}(f)$ is not necessarily propositional even if f is constant: One can check easily that Y is a set if and only if all the functions $\mathbf{1} \rightarrow X$ have a propositional image (which of course means that all those images are contractible).

Constructing a function out of the propositional truncation of a type is somewhat tricky. A well-known [27, Chapter 3.9] strategy for defining a map $\|X\| \rightarrow Y$ is to construct a proposition P together with functions $X \rightarrow P$ and $P \rightarrow Y$. We have already implicitly done this in previous sections. We can make this method slightly more convenient to use if we observe that P does not need to be a proposition, but it only needs to be a proposition under the assumption that X is inhabited:

Principle 5.2. *Let X, Y be two types. Assume P is a type such that $P \rightarrow Y$. If X implies that P is contractible, then there is a function $\|X\| \rightarrow Y$. In particular, if $f : X \rightarrow Y$ is a function that factors through P , then f factors through $\|X\|$.*

Let us briefly justify this principle. Assume that P has the assumed property. Utilizing that the statement that P is contractible is propositional itself, we see that $\|X\|$ is sufficient to conclude that P is a proposition. This allows us to prove $\|X\| \times P$ to be propositional. The map $P \rightarrow Y$ clearly gives rise to a map $\|X\| \times P \rightarrow Y$, and the map $X \rightarrow \|X\| \times P$ is given by $|-|$ and the fact that P is contractible under the assumption X . \square

There are several situations in which this principle can be applied. The following theorem does not need it as it is mostly a restatement of our previous result from Section 4.

Theorem 5.3. *A weakly constant function $f : X \rightarrow Y$ factors through $\|X\|$ in any one of the following cases, of which the equivalent (iii) and (iv) generalize all others:*

- (i) X is empty, i.e. $X \rightarrow \mathbf{0}$
- (ii) X is inhabited, i.e. $\mathbf{1} \rightarrow X$
- (iii) X has split support, i.e. $\|X\| \rightarrow X$
- (iv) X has a weakly constant endofunction, i.e. $\Sigma_{f:X \rightarrow X} \text{wconst } f$
- (v) we have any function $g : Y \rightarrow X$.

Proof. Each of (i) and (ii) let us conclude (iii). Further, (v) gives us (iv) as the composition $g \circ f$ is a constant endofunction on X . The logical equivalence of (iii) and (iv) is Theorem 4.5. Thus, it is sufficient to prove the statement for (iii), so assume $s : \|X\| \rightarrow X$. The required

conclusion is then immediate as f is pointwise equal to the composition of $|-| : X \rightarrow \|X\|$ and $f \circ s$. \square

Our next statement implies what we mentioned at the beginning of Section 5: under the assumption of unique identity proofs, the factorization is always possible.

Theorem 5.4. *Let X, Y be again two types and $f : X \rightarrow Y$ a constant function. If Y is a set, then f factors through $\|X\|$.*

Proof. The crucial observation is that the image of f is propositional in the considered case. In detail, we proceed as follows. We define P to be the image of f , that is,

$$P \equiv \Sigma_{y:Y} \|\Sigma_{x:X} f(x) =_Y y\|. \quad (5.4)$$

In order to apply Principle 5.2, we need to know that f factors through P . This is obvious from the following diagram:

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ & \searrow \lambda x.(f(x), |x, \text{refl}_{f(x)}|) & \nearrow \text{fst} \\ & P & \end{array}$$

We need to prove that P is propositional. That is, given two elements (y_1, p_1) and (y_2, p_2) in P , we want to show that they are equal. Let us once more construct the equality via giving a pair of paths. For the second component, there is nothing to do as p_1 and p_2 live in propositional types. To show $y_1 =_Y y_2$, observe that this type is propositional as Y is a set and we may thus assume that we have inhabitants $(x_1, q_1) : \Sigma_{x_1:X} f(x_1) =_Y y_1$ and $(x_2, q_2) : \Sigma_{x_2:X} f(x_2) =_Y y_2$ instead of p_1 and p_2 . But $f(x_1) = f(x_2)$ by constancy, and therefore $y_1 = y_2$. The maps $X \rightarrow P$ and $P \rightarrow Y$ are the obvious ones and the claim follows by Principle 5.2 (or rather the preceding comment, the strengthened version is not needed). \square

It is not hard to see that, assuming function extensionality, the implication of Theorem 5.4 gives rise to an equivalence

$$(\Sigma_{f:X \rightarrow Y} \mathbf{wconst} f) \simeq (\|X\| \rightarrow Y), \quad (5.5)$$

where we use in particular that $\mathbf{wconst} f$ is propositional under the given conditions. This is the simplest non-trivial special case of the result that functions $\|X\| \rightarrow Y$ correspond to *coherently* constant functions $X \rightarrow Y$ [11].

Our last example of a special case in which the factorization can be done is more involved. However, it is worth the effort as it provides valuable intuition and an interesting application, as we will discuss below. The proof we give benefits hugely from a simplification by Sattler who showed to us how reasoning with type equivalences can be applied here.

Theorem 5.5. *Assume that function extensionality holds. If $f : X \rightarrow Y$ is constant and X is the coproduct of two propositions, then f factors through $\|X\|$.*

Proof. Assume $X \equiv Q + R$, where Q and R are propositions, and assume that $c : \mathbf{wconst} f$ is the witness of constancy. Define P to be the following Σ -type with four components:

$$\begin{aligned} P &:= \Sigma(y : Y) \\ &\quad \Sigma(s : \Pi_{q:Q} y = f(\mathbf{inl} q)) \\ &\quad \Sigma(t : \Pi_{r:R} y = f(\mathbf{inr} r)) \\ &\quad \left(\Pi_{q:Q} \Pi_{r:R} s(q)^{-1} \cdot t(r) = c(\mathbf{inl} q, \mathbf{inr} r) \right) \end{aligned} \tag{5.6}$$

In order to apply Principle 5.2 we need to construct a function $P \rightarrow Y$ and a proof that X implies that P is contractible.

The function $P \rightarrow Y$ is, of course, given by a simple projection. For the other part, let a point of X be given. Without loss of generality, we assume that this inhabitant is $\mathbf{inl} q_0$ with $q_0 : Q$. It would be possible to construct a point in P and show that this point is equal to any other point. However, constructing a chain of equivalences yields a more elegant proof. This strategy was proposed to us by Christian Sattler.

As Q is contractible with center q_0 , it suffices to only consider q_0 instead of quantifying over all elements of Q . Applying this twice shows that P is equivalent to the following type:

$$\begin{aligned} &\Sigma(y : Y) \\ &\quad \Sigma(s : y = f(\mathbf{inl} q_0)) \\ &\quad \Sigma(t : \Pi_{r:R} y = f(\mathbf{inr} r)) \\ &\quad \left(\Pi_{r:R} s^{-1} \cdot t(r) = c(\mathbf{inl} q_0, \mathbf{inr} r) \right). \end{aligned} \tag{5.7}$$

The first two Σ -components together have the shape of a singleton, showing that this part is contractible with the canonical inhabitant $(f(\mathbf{inl} q_0), \mathbf{refl})$. We may thus remove these Σ -components (see [27, Theorem 3.11.9 (ii)]) and the above type further simplifies to

$$\begin{aligned} &\Sigma(t : \Pi_{r:R} f(\mathbf{inl} q_0) = f(\mathbf{inr} r)) \\ &\quad \left(\Pi_{r:R} \mathbf{refl}^{-1} \cdot t(r) = c(\mathbf{inl} q_0, \mathbf{inr} r) \right). \end{aligned} \tag{5.8}$$

We apply the distributivity principle of Π and Σ (see [27, Theorem 2.15.7]), together with standard simplifications, to further simplify to

$$\begin{aligned} &\Pi_{r:R} \Sigma(t : f(\mathbf{inl} q_0) =_B f(\mathbf{inr} r)) \\ &\quad (t = c(\mathbf{inl} q_0, \mathbf{inr} r)). \end{aligned} \tag{5.9}$$

For any $r : R$, the dependent pair part is contractible as it is, once more, a singleton, and function extensionality allows us to conclude the stated result. \square

Theorem 5.5 was inspired by a discussion on the *homotopy type theory mailing list* [25]. Shulman observed that, for two propositions Q and R , their *join* $Q * R$ [27, Chapter 6.8], defined as the (homotopy) pushout of the diagram $Q \xleftarrow{\mathbf{fst}} Q \times R \xrightarrow{\mathbf{snd}} R$, is equivalent to $\|Q + R\|$. This means that, in the presence of *higher inductive types* [27, Chapter 6], the type $\|Q + R\|$ has the (seemingly) stronger elimination rule of the join. The second named author then asked whether higher inductive types do really improve the elimination properties of $\|Q + R\|$ in this sense. This was discussed shortly before we could answer the question negatively with the result of Theorem 5.5: its statement about $\|Q + R\|$ corresponds exactly to the elimination property of $Q * R$. Thus, the join of two propositions already exists in a minimalistic setting that involves truncation but no other higher inductive types.

6. POPULATEDNESS

In this section we discuss a notion of *anonymous existence*, similar to, but weaker (see Section 7.2) than propositional truncation. It crucially depends on the Fixed Point Lemma 4.1. Let us start by discussing another perspective on what we have explained in Section 4.

Trivially, for a type X , we can prove the statement

$$\|X\| \rightarrow (\|X\| \rightarrow X) \rightarrow X. \quad (6.1)$$

By Lemma 4.5, this is equivalent to

$$\|X\| \rightarrow \text{constEndo } X \rightarrow X, \quad (6.2)$$

and hence

$$\text{constEndo } X \rightarrow \|X\| \rightarrow X, \quad (6.3)$$

which can be read as: If we have a constant endomap on X and we wish to get an inhabitant of X (or, equivalently, a fixed point of the endomap), then $\|X\|$ is sufficient to do so. We can additionally ask whether it is also necessary: can we replace the first assumption $\|X\|$ by something weaker? Looking at formula 6.1, it would be natural to conjecture that this is not the case, but it is. In this section, we discuss what it can be replaced by, and in Section 7.2, we give a proof that it is indeed weaker.

For answering the question what is needed to get from $\text{splitSup } X$ to X , let us define the following notion:

Definition 6.1 (populatedness). For a given type X , we say that X is populated, written $\langle\langle X \rangle\rangle$, if every constant endomap on X has a fixed point:

$$\langle\langle X \rangle\rangle \equiv \prod_{f:X \rightarrow X} \text{wconst } f \rightarrow \text{fix } f, \quad (6.4)$$

where $\text{fix } f$ is the type of fixed points, defined as in Lemma 4.1.

The notion of populatedness (which, to add a caveat, is not functorial; see Theorem 7.7) allows us to comment on the question raised above. If $\langle\langle X \rangle\rangle$ has an element and X has a constant endomap, then X has an inhabitant, as such an inhabitant can be extracted from the type of fixed points by projection. Hence, $\langle\langle X \rangle\rangle$ instead of $\|X\|$ in 6.3 would be sufficient as well. Therefore,

$$\langle\langle X \rangle\rangle \rightarrow (\|X\| \rightarrow X) \rightarrow X. \quad (6.5)$$

At this point, we have to ask ourselves whether (6.5) is an improvement over (6.3). But indeed, we have the following property:

Theorem 6.2. *Any merely inhabited type is populated. That is, for a type X , we have*

$$\|X\| \rightarrow \langle\langle X \rangle\rangle. \quad (6.6)$$

Proof. Assume f is a constant endofunction on X . The claim follows directly from Corollary 4.4. \square

In Section 7 we will see that $\langle\langle X \rangle\rangle$ is in fact strictly weaker than $\|X\|$.

In the presence of propositional truncation, we can give an alternative characterization of populatedness. Recall that we indicate propositional truncation with the attribute *merely*.

Lemma 6.3. *In MLTT with propositional truncation, a type is populated if and only if the statement that it merely has split support implies that it is merely inhabited, or equivalently, if and only if the statement that X has split support allows the construction of an element of X . Formally, the following types are logically equivalent:*

- (i) $\langle\langle X \rangle\rangle$
- (ii) $\|\|X\| \rightarrow X\| \rightarrow \|X\|$
- (iii) $(\|X\| \rightarrow X) \rightarrow X$.

Proof. We have already discussed (i) \Rightarrow (iii) above, see (6.5). (iii) \Rightarrow (ii) follows from the functoriality of the truncation operator. For (ii) \Rightarrow (i), assume we have a constant endofunction f on X . Hence, we have a function $\|X\| \rightarrow X$, thus $\|\|X\| \rightarrow X\|$ and, by assumption, $\|X\|$. But $\|X\|$ is enough to construct a fixed point of f by Corollary 4.4. \square

A nice feature of the notion of populatedness is that it is definable in MLTT, and it can thus be used without making further assumptions. For the rest of this section, let us explicitly not assume that the type theory has propositional truncations. We can give one more characterization of populatedness, and a strong parallel to mere inhabitation, as follows:

Theorem 6.4. *In MLTT, a type X is populated if and only if any proposition that is logically equivalent to it holds,*

$$\langle\langle X \rangle\rangle \Leftrightarrow \Pi_{P:\mathcal{U}} \text{isProp } P \rightarrow (P \rightarrow X) \rightarrow (X \rightarrow P) \rightarrow P. \quad (6.7)$$

Note that the only difference to the type expression in Theorem 3.8 is that we only quantify over *sub-propositions* of X , i.e. over those that satisfy $P \rightarrow X$, while we quantify over all propositions in the case of $\|X\|$. This again shows that $\|X\|$, if it exists, is at least as strong as $\langle\langle X \rangle\rangle$.

Proof. Let us first prove the direction “ \rightarrow ”. Assume a proposition P is given, together with functions $X \rightarrow P$ and $P \rightarrow X$. Composition of these gives us a constant endomap on X , exactly as in the proof of Theorem 3.10. But then $\langle\langle X \rangle\rangle$ makes sure that this constant endomap has a fixed point, which is (or allows us to extract) an inhabitant of X . Using $X \rightarrow P$ again, we get P .

For the direction “ \leftarrow ”, assume we have a constant endomap f . We need to construct an inhabitant of $\text{fix } f$. In the expression on the right-hand side, choose P to be $\text{fix } f$, and everything follows from Corollary 4.4. \square

The similarities between $\|X\|$ and $\langle\langle X \rangle\rangle$ do not stop here. The following statement, together with the direction “ \rightarrow ” of the theorem that we have just proved, should be compared to the definition of $\|X\|$ (that is, Definition 3.6):

Theorem 6.5. *For any X , the type $\langle\langle X \rangle\rangle$ has the following properties:*

- (i) $X \rightarrow \langle\langle X \rangle\rangle$
- (ii) $\text{isProp}(\langle\langle X \rangle\rangle)$ (if function extensionality holds).

Proof. The first point can be shown using the map ϵ as defined in (4.7). For the second, we use that $\text{fix } f$ is a proposition (Lemma 4.1). By function extensionality, a (dependent) function type is propositional if the codomain is (see Section 2) and we are done. \square

The following result, shown without using propositional truncation, is the analog to Theorem 4.5.

Theorem 6.6. *Let X be a type. If we have a constant endomap on X , then $(\langle\langle X \rangle\rangle \rightarrow X)$. Assuming function extensionality, this implication can be reversed. If X is propositional, then $\text{constEndo } X$ and $(\langle\langle X \rangle\rangle \rightarrow X)$ are both inhabited (not requiring function extensionality).*

Proof. Given a constant endofunction f on X , an inhabitant of $\langle\langle X \rangle\rangle$ gives us $\text{fix } f$ and thus X by projection. For the other direction, if we have $(\langle\langle X \rangle\rangle \rightarrow X)$, then the composition with (Theorem 6.5.(i)) gives a constant endofunction on X . If X is propositional, then the identity is clearly constant. \square

As remarked above, $\|-\|$ is an idempotent monad in an appropriate sense, while $\langle\langle - \rangle\rangle$ is not even functorial (see Theorem 7.7). However, we do have the following:

Theorem 6.7. *Assuming function extensionality, the notion of populatedness is idempotent in the sense that, for a type X , we have an equivalence*

$$\langle\langle\langle\langle X \rangle\rangle\rangle\rangle \simeq \langle\langle X \rangle\rangle. \quad (6.8)$$

Proof. Theorem 6.5 shows that both sides are propositional and that there is a map “ \leftarrow ”. A map “ \rightarrow ” is given by Theorem 6.6. \square

7. TABOOS AND COUNTER-MODELS

In this section we look at the differences between the various notions of (anonymous) inhabitation we have encountered. We have, for a type X , the following chain of implications:

$$X \Rightarrow \|X\| \Rightarrow \langle\langle X \rangle\rangle \Rightarrow \neg\neg X. \quad (7.1)$$

The first implication is trivial and the second is given by Theorem 6.2. Maybe somewhat surprisingly, the last implication does not require function extensionality, as we do not need to prove that $\neg\neg X$ is propositional: to show

$$\langle\langle X \rangle\rangle \rightarrow \neg\neg X, \quad (7.2)$$

let us assume $f : \neg X$. But then, f can be composed with the unique function from the empty type into X , yielding a constant endomap on X , and obviously, this function cannot have a fixed point in the presence of f . Therefore, the assumption of $\langle\langle X \rangle\rangle$ would lead to a contradiction, as required.

Under the assumption of LEM, all implications of the chain (7.1) except the first can be reversed as it is easy to show

$$\Pi_{X:\mathcal{U}}(\|X\| + \neg\|X\|) \rightarrow \neg\neg X \rightarrow \|X\|. \quad (7.3)$$

Constructively, none of the implications of (7.1) should be reversible. To make that precise, we use what we call *taboos*, showing that the provability of a statement would imply the provability of another better understood statement which is known to be not provable. A taboo is essentially a type-theoretic *Brouwerian counterexample* (“constructive taboo”) or a homotopical analog (“homotopical taboo”).

In this section, we present the following discussions:

- (i) We start by assuming that the first implication can be reversed, i.e. that we have a function $\Pi_{X:\mathcal{U}} \|X\| \rightarrow X$. It is easy to see that this assumption implies that all types are sets. We show the more interesting result that all equalities are decidable. As an additional argument, if every type has split support, a form of choice that does not belong to type theory is implied. Moreover, we observe that $\|X\| \rightarrow X$ can be read as “the map $|-| : X \rightarrow \|X\|$ is a split epimorphism” (where the latter notion must be read with care), and we show that already the weaker assumption that it is an epimorphism implies that all types are sets.
- (ii) It would be nice if the second implication could be reversed, as this would imply that propositional truncation is definable in MLTT. However, this is logically equivalent to a certain weak version of the axiom of choice discussed below, which is not provable (but holds under LEM).
- (iii) Assuming function extensionality, the last implication can be reversed if and only if LEM holds.

7.1. Inhabited and Merely Inhabited. We first examine the question whether the first part of the chain (7.1) can be reversed. If X is a type, it is weaker to have an inhabitant of $\|X\|$ than to have an inhabitant of X . It is unreasonable to expect that we can show in type theory that every type has split support, but it is interesting to see what it would imply.

First of all, if we assume that all types have split support, then this in particular holds for path spaces, and by Theorem 3.10, every type is a set. This assumption also implies the axiom of choice [27, Chapter 3.8]. If we have univalence for propositions and set quotients, this allows us to use Diaconescu’s proof of LEM ([4], see [27, Theorem 10.1.14]). We want to present a similar construction in the much more minimalistic theory that we consider in the current article.

Using Theorem 4.5, we can formulate the assumption that all types have split support without using truncations as “every type has a constant endofunction”,

$$\Pi_{X:\mathcal{U}} \text{constEndo } X. \quad (7.4)$$

From a constructive point of view, this is an interesting assumption. It clearly follows from LEM_∞ : if we know an inhabitant of a type, we can immediately construct a constant endomap, and for the empty type, considering the identity function is sufficient. The assumption (7.4) contains some form of choice, but we do not expect that the general principle LEM_∞ can be derived in our setting. Hence, we may understand (7.4) as a weak form of LEM_∞ . However, what we can derive is LEM_∞ for all path spaces, i.e. that all types are discrete, see Lemma 7.1 and Theorem 7.2 below.

Lemma 7.1. *In basic MLTT (without function extensionality and without propositional truncations), let A be a type and $a_0, a_1 : A$ two points. If the type $(a_0 = x) + (a_1 = x)$ has a constant endomap for all $x : A$, then $a_0 = a_1$ is decidable.*

As we will see in the proof, we need to know $0_2 \neq_2 1_2$ for Lemma 7.1, which can be proved using a universe. If we assume $0_2 \neq_2 1_2$, the lemma is true in an even weaker setting without a type universe. Before giving the proof of Lemma 7.1, we state an immediate corollary:

Theorem 7.2. *If every type has a constant endofunction then every type has decidable equality,*

$$(\Pi_{X:\mathcal{U}} \text{constEndo } X) \rightarrow \Pi_{X:\mathcal{U}} \text{isDiscrete } X. \quad (7.5)$$

□

Proof of Lemma 7.1. For (technical and conceptual) convenience, we regard the elements a_0, a_1 as a single map

$$a : \mathbf{2} \rightarrow A \quad (7.6)$$

and we use

$$E_x := \Sigma_{i:\mathbf{2}} a_i = x \quad (7.7)$$

in place of the type $(a_0 = x) + (a_1 = x)$. In a theory with propositional truncation, the *image* of a can be defined to be $\Sigma_{x:A} \|E_x\|$ [27, Definition 7.6.3]. By assumption, we have a family of constant endofunctions f_x on E_x , and by the discussion above, we can essentially regard the type

$$E := \Sigma_{x:A} \text{fix } f_x, \quad (7.8)$$

which can be unfolded to

$$\Sigma_{x:A} \Sigma_{(i,p):E_x} f_x(i, p) = (i, p), \quad (7.9)$$

as the image of a . It is essentially the observation that we can define this image that allows us to mimic Diaconescu's argument. Recall from (4.7) that ϵ is the canonical function that maps a point of a type to a fixed point of a given endofunction on that type. Clearly, a induces a map

$$r : \mathbf{2} \rightarrow E \quad (7.10)$$

$$r(i) := (a_i, \epsilon(i, \text{refl}_{a_i})). \quad (7.11)$$

Using that the second component is an inhabitant of a proposition, we have

$$r(i) = r(j) \Leftrightarrow a_i = a_j. \quad (7.12)$$

The type E can be understood as the quotient of $\mathbf{2}$ by the equivalence relation \sim , given by $i \sim j \equiv a_i = a_j$. If E was the image of a in the ordinary sense [27, Definition 7.6.3], the axiom of choice would be necessary to find a section of r (see [27, Theorem 10.1.14]). In our situation, this section is given by a simple projection,

$$s : E \rightarrow \mathbf{2} \quad (7.13)$$

$$s(x, ((i, p), q)) := i. \quad (7.14)$$

It is easy to see that s is indeed a section of r in the sense of $\Pi_{e:E} r(s(e)) = e$. Given $(x, ((i, p), q)) : E$, applying first s , then r leads to $(a_i, \epsilon(i, \text{refl}_{a_i}))$. Equality of these expressions is equality of the first components due to the propositional second component. But p is a proof of $a_i = x$. From that property, we can conclude that, for any $e_0, e_1 : E$,

$$e_0 = e_1 \Leftrightarrow s(e_0) = s(e_1). \quad (7.15)$$

Combining (7.12) and (7.15) yields

$$a_i = a_j \Leftrightarrow s(r(i)) = s(r(j)), \quad (7.16)$$

where the right-hand side is an equality in $\mathbf{2}$ and thus always decidable. In particular, $a_0 = a_1$ is hence decidable. □

Another consequence of the assumption (7.4) is a form of choice that does not belong to intuitionistic type theory. In order to formulate and prove this, we need a few definitions.

We say that a relation $R : X \times X \rightarrow \mathcal{U}$ is *propositionally valued* if

$$\Pi_{x,y:X} \text{isProp}(R(x,y)). \quad (7.17)$$

The R -image of a point $x : X$ is

$$R_x := \Sigma_{y:X} R(x,y). \quad (7.18)$$

We say that R is *functional* if its point-images are all propositions:

$$\Pi_{x:X} \text{isProp } R_x. \quad (7.19)$$

We say that two relations $R, S : X \times X \rightarrow \mathcal{U}$ have the same domain if

$$\Pi_{x:X} R_x \Leftrightarrow S_x, \quad (7.20)$$

and that S is a *subrelation* of R if

$$\Pi_{x,y:X} S(x,y) \rightarrow R(x,y). \quad (7.21)$$

Theorem 7.3. *If all types have constant endofunctions, then every binary relation has a functional, propositionally valued subrelation with the same domain.*

Proof. Assume that $R : X \times X \rightarrow \mathcal{U}$ is given. For $x : X$, let $k_x : R_x \rightarrow R_x$ be the constant map given by the assumption (7.4) that all types have constant endofunctions. Define further

$$S(x,y) := \Sigma_{a:R(x,y)} (y,a) = k_x(y,a). \quad (7.22)$$

Then S is a subrelation of R by construction. We observe that S_x is equivalent to $\text{fix}(k_x)$ and therefore propositional (by Lemma 4.1), proving that S is functional. Together with Corollary 4.4, this further shows

$$R_x \Leftrightarrow \text{fix } k_x \Leftrightarrow S_x, \quad (7.23)$$

showing that R and S have the same domain.

What remains to show is that $S(x,y)$ is always a proposition. Let $s, s' : S(x,y)$. As S_x is propositional we know $(y,s) =_{S_x} (y,s')$. By the standard lemma this type corresponds to a dependent pair type with components

$$p : y =_X y \quad (7.24)$$

$$q : p_*(s) =_{S(x,y)} s'. \quad (7.25)$$

In our case, as every type is a set, we have $p = \text{refl}_y$, and q gives us the required proof of $s =_{S(x,y)} s'$. \square

Instead of the logically equivalent formulation (7.4), let us now assume the original assumption that $|-|$ can be reversed, that is,

$$\Pi_{X:\mathcal{U}} \|X\| \rightarrow X. \quad (7.26)$$

Note that a map $h : \|X\| \rightarrow X$ is automatically a section of $|-| : X \rightarrow \|X\|$ in the sense of

$$\Pi_{z:\|X\|} |h(z)| = z \quad (7.27)$$

as any two inhabitants of $\|X\|$ are equal. Therefore, we may read (7.26) as:

$$\text{For any type } X, \text{ the map } |-| : X \rightarrow \|X\| \text{ is a split epimorphism.} \quad (7.28)$$

We want to consider a weaker assumption, namely

$$\text{For any type } X, \text{ the map } |-| : X \rightarrow \|X\| \text{ is an } \textit{epimorphism}, \quad (7.29)$$

where we call $e : U \rightarrow V$ an *epimorphism* if, for any type W and any two functions $f, g : V \rightarrow W$, we have

$$(\Pi_{u:U} f(e u) = g(e u)) \rightarrow \Pi_{v:V} f v = g v. \quad (7.30)$$

Of course, under function extensionality, e is an epimorphism if and only if, for all W, f, g , we have

$$f \circ e = g \circ e \rightarrow f = g. \quad (7.31)$$

A caveat is required. Our definition of *epimorphism* is the direct naive translation of the usual 1-categorical notion into type theory. However, the category of types and functions with the type of equalities is not only an ordinary category, but rather an $(\infty, 1)$ -category. The definition (7.30) makes sense in the category of sets [27, Chapter 10.1], where equalities are propositional. However, the property of being an epimorphism in our sense is not propositional and it could rightfully be argued that it might not be the “correct” definition in a context where not every type is a set, similarly to how we argued that LEM_∞ is a problematic version of the principle of excluded middle. Despite this, we use the notion as we think that it helps providing an intuitive meaning to the plain type expression (7.30).

Lemma 7.4. *Let Y be a type. If the map $|-| : (y_1 = y_2) \rightarrow \|y_1 = y_2\|$ is an epimorphism for any points $y_1, y_2 : Y$, then Y is a set.*

Proof. Assume Y, y_1, y_2 are given. Define two functions

$$f, g : \|y_1 = y_2\| \rightarrow Y \quad (7.32)$$

by

$$f(q) \equiv y_1, \quad (7.33)$$

$$g(q) \equiv y_2, \quad (7.34)$$

that is, f and g are constant at y_1 and y_2 , respectively.

With these concrete choices, our assumption (7.30) with $e \equiv |-|$ becomes

$$(y_1 = y_2 \rightarrow y_1 = y_2) \rightarrow (\|y_1 = y_2\| \rightarrow y_1 = y_2) \quad (7.35)$$

which, of course, gives us a function

$$\|y_1 = y_2\| \rightarrow y_1 = y_2. \quad (7.36)$$

The statement of the lemma then follows from Theorem 3.10. \square

The following result summarizes the statements of Theorem 7.2 and Lemma 7.4:

Theorem 7.5. *In basic MLTT with weak propositional truncation,*

- (i) *if $|-| : X \rightarrow \|X\|$ is a split epimorphism for every X , then all types have decidable equality*
- (ii) *if $|-| : X \rightarrow \|X\|$ is an epimorphism for every X , then all types are sets.*

Proof. The first part is a reformulation of Theorem 7.2, while the second part is a corollary of Lemma 7.4. \square

7.2. Merely Inhabited and Populated. Assume that the second step in (7.1) can be reversed, meaning that we have

$$\Pi_{X:\mathcal{U}} \langle\langle X \rangle\rangle \rightarrow \|X\|. \quad (7.37)$$

Repeated use of the Fixed Point Lemma leads to a couple of interesting logically equivalent statements.

In the previous subsection, we have discussed that we cannot show that every type has split support. However, a weaker version of this is provable:

Lemma 7.6. *For every type X , the statement that it has split support is populated,*

$$\langle\langle \|X\| \rightarrow X \rangle\rangle. \quad (7.38)$$

To demonstrate the different possibilities that the logically equivalent formulations of populatedness offer, we want to give more than one proof. The first one uses Definition 6.1:

First proof. Assume we are given a constant endofunction f on $\|X\| \rightarrow X$. We need to construct a fixed point of f , or correspondingly, any inhabitant of $\|X\| \rightarrow X$. By Theorem 4.5, a constant function $g : X \rightarrow X$ is enough for this. Given $x : X$, we may apply f to the function that is everywhere x , yielding an inhabitant of $\|X\| \rightarrow X$. Applying it to $|x|$ gives an element of X , and we define $g(x)$ to be this element. The proof that that f is constant immediately translates to a proof that g is constant. \square

Alternatively, we can use the logically equivalent formulation of populatedness, proved in Theorem 6.4:

Second proof. Assume P is a proposition and we have a proof of

$$P \Leftrightarrow (\|X\| \rightarrow X). \quad (7.39)$$

We need to show P . The logical equivalence above immediately provides an inhabitant of $X \rightarrow P$, and, by the rules of the propositional truncation, therefore $\|X\| \rightarrow P$. Assume $\|X\|$. We get P , thus $\|X\| \rightarrow X$ with the above equivalence, and therefore X (using the assumed $\|X\|$ again). This shows $\|X\| \rightarrow X$, and consequently, P . \square

Finally, we can also use that $\langle\langle - \rangle\rangle$ can be written in terms of $\|- \|$:

Third proof. Using Lemma 6.3(iii), the statement that needs to be shown becomes

$$(\| \|X\| \rightarrow X \| \rightarrow \|X\| \rightarrow X) \rightarrow (\|X\| \rightarrow X), \quad (7.40)$$

which is immediate. \square

The assumption that populatedness and mere inhabitation are equivalent has a couple of “suspicious” consequences, as we want to show now.

Theorem 7.7. *In MLTT with weak propositional truncation, the following are logically equivalent:*

- (i) *every populated type is merely inhabited,*

$$\Pi_{X:\mathcal{U}} \langle\langle X \rangle\rangle \rightarrow \|X\| \quad (7.41)$$

- (ii) *every type merely has split support,*

$$\Pi_{X:\mathcal{U}} \| \|X\| \rightarrow X \| \quad (7.42)$$

(iii) every proposition is projective in the following sense:

$$\Pi_{P:\mathcal{U}} \text{isProp } P \rightarrow \Pi_{Y:P \rightarrow \mathcal{U}} (\Pi_{p:P} \|Y(p)\|) \rightarrow \|\Pi_P Y\| \quad (7.43)$$

(note that this is the axiom of choice [27, Chapter 3.8] for propositions, without the requirement that Y is a family of sets)

(iv) $\langle\langle - \rangle\rangle : \mathcal{U} \rightarrow \mathcal{U}$ is functorial in the sense that

$$\Pi_{X,Y:\mathcal{U}} (X \rightarrow Y) \rightarrow (\langle\langle X \rangle\rangle \rightarrow \langle\langle Y \rangle\rangle), \quad (7.44)$$

where this naming is justified at least in the presence of function extensionality which implies that $\langle\langle X \rangle\rangle \rightarrow \langle\langle Y \rangle\rangle$ is propositional, ensuring $\langle\langle g \circ f \rangle\rangle = \langle\langle g \rangle\rangle \circ \langle\langle f \rangle\rangle$.

Further, (iv) can be formulated in MLTT without assumptions on the availability of propositional truncation. If it holds, then $\langle\langle - \rangle\rangle$ satisfies the recursion principle of the weak propositional truncation. Additionally assuming function extensionality, $\langle\langle - \rangle\rangle$ can then serve as an implementation of the weak propositional truncation.

Proof. Let us first show the final claim. If Y is propositional, then $\langle\langle Y \rangle\rangle \rightarrow Y$ by Theorem 6.6. Together with (iv), this gives the claimed recursion principle. The rest of the properties of the weak propositional truncation is given by Theorem 6.5.

Let us show the logical equivalence of the four types. The above observation immediately implies (iv) \Rightarrow (i). The direction (i) \Rightarrow (iv) is also immediate by functoriality of $\|-\|$. The logical equivalence of the first two points follows easily from what we already know. (i) \Rightarrow (ii) is an application of Lemma 7.6, while (ii) \Rightarrow (i) follows from Lemma 6.3.

Let us now show (i) \Rightarrow (iii). Let P be some proposition and $Y : P \rightarrow \mathcal{U}$ some family of types. If we assume (i), it is then enough to prove

$$(\Pi_{p:P} \|Y(p)\|) \rightarrow \langle\langle \Pi_P Y \rangle\rangle. \quad (7.45)$$

By Lemma 6.3, it is enough to show

$$(\Pi_{p:P} \|Y(p)\|) \rightarrow (\|\Pi_P Y\| \rightarrow \Pi_P Y) \rightarrow \Pi_P Y. \quad (7.46)$$

Under several assumptions, one of them being that some $p_0 : P$ is given, we need to construct an inhabitant of $Y(p_0)$. Recall the principle of the *neutral contractible exponent* that we used in the proof of Theorem 5.5. Here, it allows us to replace $\Pi_P Y$ by $Y(p_0)$ and $\Pi_{p:P} \|Y(p)\|$ by $\|Y(p_0)\|$, and the type (7.46) becomes

$$\|Y(p_0)\| \rightarrow (\|Y(p_0)\| \rightarrow Y(p_0)) \rightarrow Y(p_0), \quad (7.47)$$

which is obvious.

(iii) \Rightarrow (ii) can be seen easily by taking P to be $\|X\|$ and Y to be constantly X . \square

Consider the third of the four statements in Theorem 7.7. When $Y(p)$ is a set with exactly two elements for every $p : P$, this amounts to *the world's simplest axiom of choice* [5], which fails in some toposes. We expect that this makes it possible to show that, in MLTT with weak propositional truncation, $\Pi_{X:\mathcal{U}} \langle\langle X \rangle\rangle \rightarrow \|X\|$ is not derivable.

7.3. Populated and Non-Empty. If we can reverse the last implication of the chain, we have

$$\Pi_{X:\mathcal{U}} \neg\neg X \rightarrow \langle\langle X \rangle\rangle. \quad (7.48)$$

To show that this cannot be provable, we show that it is equivalent to LEM, a constructive taboo.

Theorem 7.8. *With function extensionality, we have the following (logical) equivalence:*

$$(\Pi_{X:\mathcal{U}} \neg\neg X \rightarrow \langle\langle X \rangle\rangle) \Leftrightarrow \text{LEM}. \quad (7.49)$$

Proof. The direction “ \leftarrow ” is easy: from $X \rightarrow \langle\langle X \rangle\rangle$, we get $\neg\neg X \rightarrow \neg\neg\langle\langle X \rangle\rangle$. As $\langle\langle X \rangle\rangle$ is propositional, LEM gives us $\neg\neg\langle\langle X \rangle\rangle \rightarrow \langle\langle X \rangle\rangle$.

For the direction “ \rightarrow ”, assume that P is a proposition. Thus, the type $P + \neg P$ is a proposition as well, and hence, the identity function on $P + \neg P$ is constant.

It is straightforward to construct a proof of $\neg\neg(P + \neg P)$. By the assumption, this means that $P + \neg P$ is populated, i.e. every constant endomap on it has a fixed point. Therefore, we can construct a fixed point of the identity function, which is equivalent to proving $P + \neg P$. \square

8. PROPOSITIONAL TRUNCATION WITH JUDGMENTAL COMPUTATION RULE

Propositional truncation is often defined to satisfy the judgmental computation rule [27, Chapter 3.7],

$$\text{rec}_{\text{tr}}(P, h, f, |x|) \equiv_{\beta} f(x) \quad (8.1)$$

for any function $f : X \rightarrow P$ where $x : X$ and P is propositional. In our discussion, we did not assume it to hold so far. We certainly do not want to argue that a theory without this judgmental equation is to be preferred, we simply did not need it. We agree with the very common view (see the introduction of [27, Chapter 6]) that judgmental computation rules are often advantageous, not only for truncations, but for *higher inductive types* [27, Chapter 6] in general. Without them, some expressions will need to involve a ridiculous amount of transporting, just to make them type-check, and the “computation” will have to be done manually in order to simplify terms. If (8.1) is assumed, it suggests itself to also assume a judgmental computation rule for the induction principle, that is

$$\text{ind}_{\text{tr}}(P, h, f, |x|) \equiv_{\beta} f(x), \quad (8.2)$$

where $P : \|X\| \rightarrow \mathcal{U}$ might now be a type family and $f : \Pi_{z:\|X\|} P(z)$ is a dependent function rather than a simple function. Interestingly, it does not seem to be possible to construct ind_{tr} from rec_{tr} such that (8.2) holds if (8.1) holds. In particular, the term constructed in Lemma 3.7 does not have the expected judgmental computation rule.

Having said this, the judgmental β -rules do have some other noteworthy consequences. Unlike the previous results, the statements in this part of our article do need the computation rules to hold judgmentally. So far, all our lemmata and theorems have been internal to type theory. This is only partially the case for the results from this section, as any statement that some equality holds judgmentally is a meta-theoretic property. We thus can not implement such a statement as a type in a proof assistant such as Agda, but we can still use Agda to check our claims; for example, if

$$p : x = y \quad (8.3)$$

$$p \equiv \text{refl}_x \quad (8.4)$$

type-checks, we may conclude that the equality does hold judgmentally.

8.1. The Interval. The interval \mathbb{I} as a higher inductive type [27, Chapter 6.3] is a type in homotopy type theory that consists of two points $i_0, i_1 : \mathbb{I}$ and a path $\text{seg} : i_0 =_{\mathbb{I}} i_1$ between them. Its *recursion*, or *non-dependent elimination* principle says: Given

$$Y : \mathcal{U} \tag{8.5}$$

$$y_0 : Y \tag{8.6}$$

$$y_1 : Y \tag{8.7}$$

$$p : y_0 = y_1, \tag{8.8}$$

there exists a function $f : \mathbb{I} \rightarrow Y$ such that

$$f(i_0) \equiv y_0 \tag{8.9}$$

$$f(i_1) \equiv y_1 \tag{8.10}$$

$$\text{ap}_f(\text{seg}) = p. \tag{8.11}$$

For the interval's induction principle, we refer to [27, Chapter 6.3]. The interval is a contractible type and as such equivalent to the unit type. However, this does not make it entirely boring; it is the *judgmental* equalities that matter. Note that the *computation rules* for the *points* are judgmental (8.9,8.10), while the rule for the path (8.11) is only given by an equality proof.

We will now show that $\|\mathbf{2}\|$ can be regarded as the interval.

Theorem 8.1. *For the type $\|\mathbf{2}\|$, the recursion principle of the interval (including the computational behavior) is derivable using (8.1), and the induction principle follows from (8.2).*

Proof. We only show that the recursion principle is derivable, which will be sufficient for the subsequent developments. The induction principle can be derived very similarly. We need to show that, under the assumptions (8.5-8.8), there is a function $f : \|\mathbf{2}\| \rightarrow Y$ such that

$$f(|0_2|) \equiv y_0 \tag{8.12}$$

$$f(|1_2|) \equiv y_1 \tag{8.13}$$

$$\text{ap}_f(\text{h}_{\text{tr}|0_2|,|1_2|}) = p. \tag{8.14}$$

We define

$$g : \mathbf{2} \rightarrow \Sigma_{y:Y} y_0 = y \tag{8.15}$$

$$g(0_2) \equiv (y_0, \text{refl}) \tag{8.16}$$

$$g(1_2) \equiv (y_1, p). \tag{8.17}$$

As $\Sigma_{y:Y} y_0 = y$ is contractible, g can be extended to a function $\bar{g} : \|\mathbf{2}\| \rightarrow \Sigma_{y:Y} y_0 = y$, and we define $f \equiv \text{fst} \circ \bar{g}$. It is easy to check that f has indeed the required judgmental properties (8.12) and (8.13). The equality (8.14) is only slightly more difficult: First, using the definition of f and a standard functoriality property of ap [27, Lemma 2.2.2 (iii)], we observe that $\text{ap}_f(\text{h}_{\text{tr}|0_2|,|1_2|})$ may be written as

$$\text{ap}_{\text{fst}}(\text{ap}_{\bar{g}}(\text{h}_{\text{tr}|0_2|,|1_2|})). \tag{8.18}$$

But here, the path $\text{ap}_{\bar{g}}(\text{h}_{\text{tr}|0_2|,|1_2|})$ is an equality in the contractible type $(y_0, \text{refl}) = (y_1, p)$ (note that both terms inhabit a contractible type themselves) and thereby unique. In particular, it is equal to the path which is built out of two components, the first of which can be chosen to be p (the second component can then be taken to be a canonically constructed inhabitant of $p_*(\text{refl}) = p$). \square

8.2. Function Extensionality. It is known that the interval \mathbb{I} with its judgmental computation rules implies function extensionality. We may therefore conclude that propositional truncation is sufficient as well.

Lemma 8.2 (Shulman [22]). *In a type theory with \mathbb{I} and the judgmental η -law for functions (which we assume), function extensionality is derivable.*

Proof. Assume X, Y are types and $f, g : X \rightarrow Y$ are functions, and $h : \prod_{x:X} f(x) = g(x)$ a proof that they are pointwise equal. Using the recursion principle of \mathbb{I} , we may then define a family

$$k : X \rightarrow \mathbb{I} \rightarrow Y \quad (8.19)$$

of functions, indexed over X , such that $k(x, i_0) \equiv f(x)$ and $k(x, i_1) \equiv g(x)$ for all $x : X$; of course, we use $h(x)$ as the required family of paths. Switching the arguments gives a function

$$k' : \mathbb{I} \rightarrow X \rightarrow Y \quad (8.20)$$

with the property that $k'(i_0) \equiv f$ and $k'(i_1) \equiv g$ (by η for functions), and thereby $\text{ap}_{k'}(\text{seg}) : f = g$. \square

The combination of Theorem 8.1 and Lemma 8.2 implies:

Corollary 8.3. *In type theory with propositional truncation that satisfies the judgmental computation rule, function extensionality can be derived.* \square

8.3. Judgmental Factorization. The judgmental computation rule of $\|-\|$ also allows us to factor any function *judgmentally* through the propositional truncation as soon as it can be factored in any way. This observation is inspired by and a generalization of the fact that $\|\mathbf{2}\|$ satisfies the judgmental properties of the interval (Theorem 8.1).

Theorem 8.4. *Any (non-dependent) function that factors through the propositional truncation can be factored judgmentally: assume types X, Y and a function $f : X \rightarrow Y$ between them. Assume that there is $\bar{f} : \|X\| \rightarrow Y$ such that*

$$h : \prod_{x:X} f(x) = \bar{f}(|x|). \quad (8.21)$$

Then, we can construct a function $f' : \|X\| \rightarrow Y$ such that, for all $x : X$, we have

$$f(x) \equiv f'(|x|), \quad (8.22)$$

which means that the type $\prod_{x:X} f(x) = f'(|x|)$ is inhabited by the function that is constantly refl .

Proof. We define a function

$$g : X \rightarrow \prod_{z:\|X\|} \sum_{y:Y} y = \bar{f}(z) \quad (8.23)$$

$$g(x) := \lambda z. \left(f(x), h(x) \cdot \text{ap}_{\bar{f}}(\text{htr}_{|x|,z}) \right) \quad (8.24)$$

By function extensionality and the fact that singletons are contractible, the codomain of g is contractible, and thus, we can extend g and get

$$\bar{g} : \|X\| \rightarrow \prod_{z:\|X\|} \sum_{y:Y} y = \bar{f}(z). \quad (8.25)$$

We define

$$f' := \lambda z : \|X\|. \text{fst}(\bar{g} z z) \quad (8.26)$$

and it is immediate to check that f' has the required properties. \square

Note that in the above argument we have only used (8.1). We have avoided (8.2) by introducing the variable z in (8.23), which is essentially a duplication of the first argument of the function, as it becomes apparent in (8.26).

Furthermore, we have assumed that f is a non-dependent function. The question does not make sense if f is dependent in the sense of $f : \Pi_{x:X} Y(x)$; however, it does for $f : \Pi_{x:X} Y(|x|)$. In this case, it seems to be unavoidable to use (8.2), but the above proof still works with minimal adjustments. We state it for the sake of completeness.

Theorem 8.5. *Let X be a type and $Y : \|X\| \rightarrow \mathcal{U}$ a type family. Assume we have functions*

$$f : \Pi_{x:X} Y(|x|) \quad (8.27)$$

$$\bar{f} : \Pi_{z:\|X\|} Y(z) \quad (8.28)$$

such that

$$\Pi_{x:X} f(x) =_{Y(|x|)} \bar{f}(|x|). \quad (8.29)$$

Then, we can construct a function $f' : \Pi_{z:\|X\|} B(z)$ with the property that for any $x : X$, we have the judgmental equality

$$f(x) \equiv f'(|x|). \quad (8.30)$$

Proof. Because we allow ourselves to use (8.2) the proof becomes actually simpler than the proof above. This time, we can define

$$g : \Pi_{x:X} \Sigma_{y:Y} y = \bar{f}(|x|) \quad (8.31)$$

$$g(x) :\equiv (f(x), h(x)). \quad (8.32)$$

Using (8.2), we get

$$\bar{g} : \Pi_{z:\|X\|} \Sigma_{y:Y} y = \bar{f}(z). \quad (8.33)$$

Then,

$$\text{fst} \circ \bar{g} \quad (8.34)$$

fulfils the required condition. \square

8.4. An Invertibility Puzzle. For a type X , the function $|-| : X \rightarrow \|X\|$ turns an element $x : X$ into an anonymous inhabitant $|x| : \|X\|$. It is thus reasonable to think of $|-|$ as a function that hides information. However, as we will demonstrate, this interpretation is only justified as long as we think of internal properties. We will show that the function $|-|$ does not erase any meta-theoretical information in the following sense: Assume $z : \|X\|$ is defined to be $|x|$ for some $x : X$. Without looking at this definition, we can recover x (e.g. in a proof assistant, z could be imported from another file; then, we do not need to open that file in order to find out x). To do this, we only need to observe how z computes in a suitable environment. To be precise, we construct a term myst_X such that, for any z as above, the expression $\text{myst}_X(z)$ is judgmentally equal to x .

The meta-theoretic statement that we can recover x from z is true, but in general, myst_X might not be a closed term (i.e. could depend on some assumptions which do not influence the computation). However, assuming the univalence axiom, myst_X can be constructed without any further assumptions for a non-trivial class of types including the natural numbers. That

is, in MLTT with propositional truncations and the univalence axiom, we can construct a term $\text{myst}_{\mathbb{N}}$ such that

$$\text{id}' : \mathbb{N} \rightarrow \mathbb{N} \quad (8.35)$$

$$\text{id}' \equiv \lambda n. \text{myst}_{\mathbb{N}}(|n|) \quad (8.36)$$

type-checks and id' is the identity function on \mathbb{N} , with a proof

$$\mathbf{p} : \prod_{n:\mathbb{N}} \text{id}'(n) = n \quad (8.37)$$

$$\mathbf{p} \equiv \lambda n. \text{refl}_n. \quad (8.38)$$

We think that the possibility to do this is counter-intuitive and surprising. In particular it may seem that we could apply $\text{ap}_{\text{myst}_{\mathbb{N}}}$ on the canonical inhabitant of $|0| =_{\|\mathbb{N}\|} |1|$ to conclude $0 =_{\mathbb{N}} 1$. However, this would only work if the type of $\text{myst}_{\mathbb{N}}$ was $\|\mathbb{N}\| \rightarrow \mathbb{N}$, which it is not; it is a Π -type that is not easier to write down than the full definition of its inhabitant $\text{myst}_{\mathbb{N}}$ itself. In the following, we show the full construction. For further discussion, see the homotopy type theory blog entry by the first named author [10], where this result was presented originally.

First, let us state two useful general definitions:

Definition 8.6 (Pointed Types [27, Definition 2.1.7]). A *pointed type* is a pair (X, x) of a type $X : \mathcal{U}$ and an inhabitant $x : X$. We write \mathcal{U}_{\bullet} for the type of pointed types,

$$\mathcal{U}_{\bullet} := \Sigma_{X:\mathcal{U}} X. \quad (8.39)$$

Definition 8.7 (Transitive Type). We say a type X is *transitive* and write $\text{isTransitive } X$ if it satisfies

$$\prod_{x,y:X} (X, x) =_{\mathcal{U}_{\bullet}} (X, y). \quad (8.40)$$

This is, of course, where univalence comes into play. It gives us the principle that a type X is transitive if, and only if, for every pair $(x, y) : X \times X$ there is an automorphism $e_{xy} : X \rightarrow X$ such that $e_{xy}(x) = y$.

We have the following examples of transitive types:

Example 8.8. Every type with decidable equality is transitive.

This is because decidable equality on X lets us define an endofunction on X which swaps x and y , and leaves everything else constant. Instances for this example include all contractible and, more generally, propositional types, but also our main candidate, the natural numbers \mathbb{N} .

Example 8.9. For any pointed type X with elements $x_1, x_2 : X$, the identity type $x_1 =_X x_2$ is transitive. In particular, the *loop space* $\Omega^n(X)$ [27, Definition 2.1.8] is transitive for any pointed type X .

Here, it is enough to observe that, for $p_1, p_2 : x_1 =_X x_2$, the function $\lambda q. q \cdot p_1^{-1} \cdot p_2$ is an equivalence with the required property.

As mentioned by Andrej Bauer in a discussion on this result [10], we also have the following:

Example 8.10. Any group [27, Definition 6.11.1] is a transitive type.

As for equality types, the reason is that there is an inverse operation, such that the automorphism $\lambda c. c \cdot a^{-1} \cdot b$ maps a to b .

Example 8.11. If X is any type and $Y : X \rightarrow \mathcal{U}$ is a family of transitive types, then $\Pi_{x:X} Y(x)$ is transitive.

In particular, \times and \rightarrow preserve transitivity of types.

We are now ready to construct **myst**: Assume that we are given a type X . We can define a map

$$f : X \rightarrow \mathcal{U}_\bullet \quad (8.41)$$

$$f(x) :\equiv (X, x). \quad (8.42)$$

If we know a point $x_0 : X$, we may further define

$$\bar{f} : \|X\| \rightarrow \mathcal{U}_\bullet \quad (8.43)$$

$$\bar{f}(z) :\equiv (X, x_0). \quad (8.44)$$

If X is transitive, we have

$$\Pi_{x:X} f(x) = \bar{f}(|x|). \quad (8.45)$$

By Theorem 8.4, there is then a function

$$f' : \|X\| \rightarrow \mathcal{U}_\bullet \quad (8.46)$$

such that, for any $x : X$, we have

$$f'(|x|) \equiv f(x) \equiv (X, x). \quad (8.47)$$

Let us define

$$\mathbf{myst}_X : \Pi_{z:\|X\|} \mathbf{fst}(f'(z)) \quad (8.48)$$

$$\mathbf{myst}_X :\equiv \mathbf{snd} \circ f'. \quad (8.49)$$

Note that while the type of \mathbf{myst}_X is *not* simply $\|X\| \rightarrow X$, we have that, for any $x : X$, the type of $\mathbf{myst}_X(|x|)$ is judgmentally equal to X , and we have $\mathbf{myst}_X(|x|) \equiv x$. This already proves the following:

Theorem 8.12. *Let X be an inhabited transitive type. Then, there is a term \mathbf{myst}_X such that the (dependent) composition*

$$\mathbf{myst}_X \circ |-| : X \rightarrow X \quad (8.50)$$

type-checks and is equal to the identity, where the proof

$$p : \Pi_{x:X} \mathbf{myst}_X(|x|) =_X x \quad (8.51)$$

$$p(x) :\equiv \mathbf{refl}_x \quad (8.52)$$

it trivial. □

It is tempting to unfold the type expression $\Pi_{z:\|X\|} \text{fst}(f'(z))$ in order to better understand it. Unfortunately, this is not feasible as this plain type expression involves the whole proof term f' , which, in turn, includes the complete construction of Theorem 8.4. We want to emphasize again that, while we do have $\text{h}_{\text{tr}|x|,|y|} : |x| =_{\|X\|} |y|$ for any $x, y : X$, we cannot conclude $\text{myst}_X(|x|) =_X \text{myst}_X(|y|)$ as the expression $\text{ap}_{\text{myst}_X}(\text{h}_{\text{tr}|x|,|y|})$ does not type-check.

Finally, we want to remark that the construction of myst does not need the full strength of Theorem 8.4. The weaker version in which $\bar{f} : \|X\| \rightarrow Y$ is replaced by a fixed $y_0 : Y$ is sufficient: in this case, \bar{f} can be understood to be *constant at y_0* . This leads to a simplification as the dependent function types in (8.23) and (8.25) can be replaced by their codomains.

It may be helpful to see the whole definition of myst explicitly in this variant, which is also how it was explained originally by the first named author [10]: We define

$$f : X \rightarrow \Sigma_{A:\mathcal{U}} A =_{\mathcal{U}} (X, x_0) \quad (8.53)$$

$$f(x) := ((X, x), \text{transitive}_X(x, x_0)), \quad (8.54)$$

where transitive_X is the proof that X is transitive. The function f in (8.41) is then simply the composition $\text{fst} \circ f$. As the codomain of f is a singleton, it is contractible (see Definition 2.1) and thereby propositional (let us write h for the proof thereof). Hence, we get

$$f' : \|X\| \rightarrow \Sigma_{A:\mathcal{U}} A =_{\mathcal{U}} (X, x_0) \quad (8.55)$$

$$f' := \text{rec}_{\text{tr}}(\Sigma_{A:\mathcal{U}} A =_{\mathcal{U}} (X, x_0)) \ h \ f. \quad (8.56)$$

We could now define myst'_X to be

$$\text{myst}'_X : \Pi_{\|X\|} \text{fst} \circ \text{fst} \circ f' \quad (8.57)$$

$$\text{myst}'_X \equiv \text{snd} \circ \text{fst} \circ f' \quad (8.58)$$

which has the same property as (8.49), even though it is not judgmentally the same term.

9. CONCLUSION AND OPEN PROBLEMS

In this article, generalizations of Hedberg's Theorem have led us to an exploration of what we call *weakly constant functions*. The attribute *weakly* indicates that higher coherence conditions of such a constancy proof are missing. As a consequence, it is not possible to derive a function $\|X\| \rightarrow Y$ from a weakly constant function $X \rightarrow Y$, but we have shown how to do this in several non-trivial special cases. Most interesting is certainly the case of endofunctions. A weakly constant endofunction can always be factored through the propositional truncation of its domain. Further, for a given X , the type which says that every constant endofunction on X has a fixed point is propositional, enabling us to use it as a notion of anonymous inhabitation $\langle\langle X \rangle\rangle$, and we have argued that it lies strictly in between of $\neg\neg X$ and $\|X\|$.

There are two questions for which we have not given an answer. The first is: Is weak propositional truncation definable in Martin-Löf type theory? This is commonly believed to not be the case. However, the standard models do have propositional truncation, making it hard to find a concrete proof. Moreover, populatedness, a similar notion of anonymous existence, is definable.

Our second question is about the consequences of the assumption that weakly constant functions factor in general. By Shulman's result [23], we know that this is inconsistent with

the univalence axiom. Is it possible to strengthen this result further? In particular, does it imply UIP for all types? We leave these questions open.

ACKNOWLEDGEMENTS

The first named author would like to thank Paolo Capriotti, Ambrus Kaposi, Nuo Li and especially Christian Sattler for many fruitful discussions. We are grateful to the anonymous referees for numerous helpful suggestions and remarks. We also thank Nils Anders Danielsson for his careful reading of our draft and for pointing out several typos.

REFERENCES

- [1] Steve Awodey and Andrej Bauer. Propositions as [types]. *Journal of Logic and Computation*, 14(4):447–471, 2004. doi: 10.1093/logcom/14.4.447.
- [2] Steve Awodey and Michael Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146:45–55, 2009. doi: 10.1017/S0305004108001783.
- [3] Robert Lee Constable. Constructive mathematics as a programming logic I: Some principles of theory. In *Annals of Mathematics*, volume 24, pages 21–37. Elsevier Science Publishers, B.V. (North-Holland), 1985. Reprinted from *Topics in the Theory of Computation*, Selected Papers of the International Conference on Foundations of Computation Theory, FCT ’83.
- [4] Radu Diaconescu. Axiom of choice and complementation. *Proc. Amer. Math. Soc.*, 51:176–178, 1975. doi: 10.1090/S0002-9939-1975-0373893-X.
- [5] Michael P. Fourman and Andre Šcedrov. The “world’s simplest axiom of choice” fails. *Manuscripta Math.*, 38(3):325–332, 1982. doi: 10.1007/BF01170929.
- [6] Michael Hedberg. A coherence theorem for Martin-Löf’s type theory. *Journal of Functional Programming*, 8(4):413–436, 1998. doi: 10.1017/S0956796898003153.
- [7] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Venice Festschrift*, pages 83–111. Oxford University Press, 1996.
- [8] William A. Howard. The formulas-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980.
- [9] Nicolai Kraus. A direct proof of Hedberg’s theorem, March 2012. Blog post at homotopytypetheory.org/2012/03/30.
- [10] Nicolai Kraus. The truncation map $| - | : N \rightarrow \|N\|$ is nearly invertible, October 2013. Blog post at homotopytypetheory.org/2013/10/28.
- [11] Nicolai Kraus. The general universal property of the propositional truncation. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 111–145, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi: 10.4230/LIPIcs.TYPES.2014.111.
- [12] Nicolai Kraus. *Truncation Levels in Homotopy Type Theory*. PhD thesis, School of Computer Science, University of Nottingham, Nottingham, UK, 2015.
- [13] Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. Generalizations of Hedberg’s theorem. In Masahito Hasegawa, editor, *Typed Lambda Calculus and Applications (TLCA)*, volume 7941 of *Lecture Notes in Computer Science*, pages 173–188. Springer-Verlag, 2013. doi: 10.1007/978-3-642-38946-7_14.
- [14] Nicolai Kraus, Martín Escardó, Thierry Coquand, and Thorsten Altenkirch. Notions of anonymous existence in Martin-Löf type theory (electronic appendix: Agda formalization), 2016.
- [15] 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*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.

- [16] Per Martin-Löf. Constructive mathematics and computer programming. In L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski, editors, *Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979*, volume 104 of *Studies in Logic and the Foundations of Mathematics*, pages 153–175. North-Holland, 1982. doi: 10.1016/S0049-237X(09)70189-2.
- [17] Per Martin-Löf. An intuitionistic theory of types. In Giovanni Sambin and Jan M. Smith, editors, *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 127–172. Oxford University Press, 1998.
- [18] Ray Mines, Fred Richman, and Wim Ruitenberg. *A Course in constructive algebra*. Universitext. Springer-Verlag, New York, 1988.
- [19] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, Göteborg, Sweden, 2007.
- [20] Erik Palmgren. Proof-relevance of families of setoids and identity in type theory. *Arch. Math. Log.*, 51(1-2):35–47, 2012.
- [21] 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 (TLCA)*, number 664 in Lecture Notes in Computer Science, 1993. doi: 10.1007/BFb0037116.
- [22] Michael Shulman. An interval type implies function extensionality, April 2011. Blog post at homotopytypetheory.org/2011/04/04.
- [23] Michael Shulman. Not every weakly constant function is conditionally constant, June 2015. Blog post at homotopytypetheory.org/2015/06/11.
- [24] Thomas Streicher. Investigations into intensional type theory, 1993. Habilitationsschrift, Ludwig-Maximilians-Universität München.
- [25] The HoTT and UF community. Homotopy type theory mailing list, since 2011. Google Groups.
- [26] The HoTT and UF community. HoTT GitHub repository, since 2011. Available online at github.com/HoTT.
- [27] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. homotopytypetheory.org/book, Institute for Advanced Study, 2013.
- [28] Vladimir Voevodsky. Foundations, since 2010. Coq Library based on Univalent Foundations, available at the author’s institutional webpage.
- [29] Voevodsky Voevodsky. The equivalence axiom and univalent models of type theory, 2010.