Kraus, Nicolai (2015) Truncation levels in homotopy type theory. PhD thesis, University of Nottingham.

TRUNCATION LEVELS IN HOMOTOPY TYPE THEORY

by

Nicolai Kraus

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

July 2015

Summary

We present several original results in homotopy type theory which are related to the truncation level of types, a concept due to Voevodsky. To begin, we give a few simple criteria for determining whether a type is 0-truncated (a *set*), inspired by a well-known theorem by Hedberg, and these criteria are then generalised to arbitrary n. This naturally leads to a discussion of functions that are weakly constant, i.e. map any two inputs to equal outputs. A weakly constant function does in general not factor through the propositional truncation of its domain. However, the factorisation is (among other cases) always possible for weakly constant *endo*functions, which we use to define a propositional notion of existence. Further, we present a couple of constructions which are only possible with the judgmental computation rule for the truncation, for example an invertibility puzzle that seemingly inverts the canonical map from $\mathbb{N}$ to $\|\mathbb{N}\|$.

One of the two main results is the construction of strict n-types in Martin-Löf type theory with a hierarchy of univalent universes (and without higher inductive types), and a proof that the universe $\mathcal{U}_n$ is not n-truncated. The other main result of this thesis is a generalised universal property of the propositional truncation, using a construction of *coherently constant* functions. We show that the type of such coherently constant functions from A to B is equivalent to the type $\|A\| \to B$. In the general case the definition requires an infinite tower of conditions, which exists if the type theory has Reedy limits of diagrams over ω^{op}. If B is an n-type for some given finite n, (non-trivial) Reedy limits are unnecessary, allowing us to construct functions $\|A\| \to B$ in homotopy type theory without further assumptions. To obtain these results, we develop some theory on equality diagrams, especially equality semi-simplicial types. In particular, we show that the semi-simplicial equality type over any type satisfies the Kan condition, which can be seen as the simplicial version of the result by Lumsdaine, and by van den Berg and Garner, that types are weak ω-groupoids.

Finally, we present some results related to formalisations of infinite structures. For example, we show how the category Δ_+ of finite non-empty sets and strictly increasing functions can be implemented so that the categorical rules hold strictly. In the presence of *very dependent types*, we speculate that this makes the "Reedy approach" for the famous open problem of defining semi-simplicial types work.

Acknowledgements

First of all, I want to express my deep and honest gratitude to Thorsten Altenkirch. It has been a privilege to be his PhD student. During the last years, he has always given me freedom to pursue my own ideas, and offered me guidance whenever I could benefit from it. When I encountered a problem, he always readily provided suggestions and support (even at times when other things kept him busy as well). He made it possible that I could visit the univalent foundations special year in Princeton and many other interesting events. He is an excellent teacher of all sorts of academic topics and an inspiring discussion partner for research ideas. Maybe most importantly, he has always been a thoughtful and caring mentor for me.

Special thanks goes to Christian Sattler. In the last decade, I have learned far more mathematics from him than from any of my professors. He is an exceptional academic colleague and an invaluable friend.

I am thankful for many interesting discussions, especially with Martín Escardó and Paolo Capriotti. Without all the ideas Martín and I have shared, the contents and probably even the title of my thesis would be different today, which I believe says it all. With Paolo, I could always discuss all sorts of questions, and I have learned a lot from him and from all our reading groups. I also thank everyone else for their interest and contributions to our regular meetings, in particular Venanzio Capretta and, of course, Christian and Thorsten, but also Gabe Dijkstra, Ambrus Kaposi, Nuo Li and, in the end, Manuel Bärenz.

My two thesis examiners, Julie Greensmith (internal examiner) and Steve Awodey (external examiner) have both spent a lot of time with my thesis. I truly appreciate their work, which has helped me in several ways. Their advice has enabled me to improve the general style of the thesis and the readability of multiple text passages, and, of course, to fix various smaller typographical mistakes. Steve's comments on my research itself have been valuable and have led to not only interesting ideas for future research, but also to a couple of remarks that I have added in the current (final) version of the thesis.

I was very lucky that I happened to be a student at the same time as Ambrus and Nuo. We have shared many interests, academic ones and non-academic ones. In the early days of my PhD studies, I have received support from Andreas Abel and Neil Sculthorpe. Already a decade before that, my interest for mathematics was stimulated by my high school teacher Markus Jakob.

There are many more people who would deserve to be mentioned. I thank the participants (especially the organisers, Steve Awodey, Thierry Coquand, and Vladimir Voevodsky) of the special year in Princeton and all other meetings for a lot of stimulating input, in particular Thierry Coquand and Michael Shulman. The first has given me advice on several occurrences and I had the pleasure of working with him, as well as with Martín and Thorsten, on a joint project. The latter has inspired me a lot through his numerous amazing blog posts. I am also grateful for the interesting remarks of Vladimir Voevodsky on one of my main

results, and I thank Andrea Vezzosi for his contributions to one of the projects I have worked on. Many people have given me feedback on my work, and I would like to thank everyone who did, as this has been very valuable for me. Some of the comments I could understand immediately, and others required (or still require) me to spend some time before I could fully benefit from them. I explicitly want to include the anonymous reviewers of the work that I have published during the time of my PhD studies, who have all given very helpful feedback. In general, the community of this research area is welcoming and friendly, making it easy for a student to become a part of it.

I am also grateful to Venanzio Capretta for spending time with my work and for making valuable suggestions during my annual reviews. Apart from Martín and Thorsten, I especially thank Graham Hutton for his general support in many situations. All the members of the functional programming lab in Nottingham have made the last years truly enjoyable.

I want to restrict these acknowledgements to the mostly academic component. I am not someone who likes to make details of his private life public. Nevertheless, the support that many people (especially Irmgard and Norbert Kraus, and Jocelyn Chen) have given me outside of the academic environment has been extremely important, and I know they are aware of my thankfulness.

Contents

Chapter 1

Introduction

Homotopy type theory is a new branch of mathematics. It forms a bridge between seemingly very distant topics: Only ten years ago, very few, if any, type theorists would have expected to get involved in algebraic topology or the theory of weak ω-categories, and neither would researchers who feel at home when it comes to the fundamental groups of spaces have believed that a significant amount of their discoveries can be formalised and computer-verified in an elegant way, using a foundation of mathematics that is based on something known as *Martin-Löf type theory*.

This thesis presents several results on *truncation levels*, informally, the higher homotopical structure of types. The important observation that this concept can be formulated internally in type theory is due to Voevodsky [Voe10a]. The thesis is subdivided into nine chapters. At the beginning of each chapter, we give a very concise overview over its contents. In this introductory Chapter 1, we first present a short historical outline (Section 1.1), and the ideas of truncation levels is explained in Section 1.2. We then give a detailed overview over the contents of this thesis and the results with their developments. In particular, a list stating which results I consider my main contributions to the field of homotopy type theory can be found at the end of Section 1.3. An important aspect of the considered field of research are computer-verified formalisations. Because of this, the current thesis has an electronic appendix with such formalisations, and some details are described in Section 1.4. Finally, in Section 1.5, we provide additional information on journal and conference publications that have been based on the contents of this thesis. Much of the work has been done in collaboration, and we strive to give a detailed statement on authorships.

There are many excellent introductions to homotopy type theory, both in terms of its development and its concepts and results. Although some information is provided in this thesis, in particular in Chapter 2, a beginner is advised to read through an introduction that covers the basic concepts in higher detail. The canonical reference is certainly the book *Homotopy Type Theory: Univalent Foundations of Mathematics* [Uni13], written by the participants of the 2012/13 Univalent

Foundations Program at the Institute for Advanced Study, Princeton. Others include the overview by Awodey [Awo12], the notice of Awodey, Pelayo, and Warren for the AMS [APW13], and the introduction by Pelayo and Warren [PW12].

1.1 Historical Outline

Martin-Löf type theory (MLTT), more precisely *intensional* Martin-Löf type theory, and sometimes also referred to as *Intuitionistic* or *Constructive type theory*, was introduced and pushed forward by Martin-Löf [ML98; ML75; ML82; ML84]. It constitutes a branch of mathematical logic with many applications in computer science, especially in the theory of programming languages. At the same time, it is powerful enough to serve as a framework for the formalisation of huge parts of mathematics. These two, namely "programming" and "proving" (loosly speaking), can indeed be viewed as the main applications of MLTT. Obviously, this connection is based on the *Curry-Howard Correspondence* [How80] and one could argue that the two concepts are the same thing; however, in praxis, someone using MLTT for programming often has slightly different requirements than someone who is trying to prove a theorem.

Among mathematicians, fairly well known is the proof assistant Coq which is based on a variant of MLTT, the *Calculus of Inductive Constructions* [CH88]. Coq has acquired much of its publicity when it was utilised by Gonthier and Werner to formalise a proof of the famous Four Colour Theorem [Gon08] which says that at most four colours are necessary to colour a map such that adjacent countries do not have the same colour. For more recent work in Coq, we want to mention the Feit-Thompson Odd Order Theorem [Gon+13] and the ForMath project.

Another implementation of MLTT is *Agda* [Nor07]. Of course, it can be used as a proof assistant, and indeed, we have used it to formalise many of our results presented in this thesis. Yet, it is often viewed as a programming language, even though there is in theory (close to) no difference between a dependently typed programming language and a proof assistant. Programming in a dependently typed language bears huge advantages. The rich type system can be utilised to provide an immediate precise formal specification or correctness proof of a program. Moreover, even though beginners of Agda who come from another functional programming language such as Haskell can find the powerful type system a burden, this opinion changes as they get accustomed to it. If a program does not type check in Agda, something is wrong, and a second thought would be required in Haskell as well, the only difference being that Haskell would not tell the programmer.

A core aspect of MLTT is computation: *terms* are identified with their *normal forms*. For concrete implementations, such as Agda and Coq, this means that we have an automatic simplification of expressions. For a programmer, this is an obvious necessity. On the other hand, in a proof on paper, such a simplification would have to be done manually by the mathematician, and we believe that the

computational behaviour of type theory can be seen as one of its main features that make it valuable for the mathematical community.

One more particularly interesting (and crucial) concept in MLTT is equality. Type theory knows two different forms of equality: first, there is the so-called *definitional* or *judgmental* equality, based on what we have just described: terms are identified if they behave identically from the computational point of view, meaning that they have the same normal form (that is, they are identical after being evaluated). In a more abstract sense, judgmental equality is a meta-theoretic concept of MLTT that is used for type checking. In intensional type theory, judgmental equality, and thus type checking, is decidable, a demand that corresponds to the very basic usage of proof assistants: if we have a potential proof p for a "proposition" P, the system should be able to check automatically whether p is indeed a *correct* proof of P. Judgmental equality in concrete implementations typically consists of β-equality and some forms of η-equality. If we want to express that a and b are judgmentally equal, we write $a \equiv b$. If we further want to express that we *define* a to be b, causing them trivially to be judgmentally equal, we write $a :\equiv b$.

As we want judgmental equality to be decidable, it is clear that this is a very strict notion of equality. Often, two mathematical objects are equal, but proving so can be arbitrarily hard. The corresponding terms in type theory will generally not be judgmentally equal, but only *propositionally* equal: for any two terms a and b of the same type A, there is the type $\mathsf{Id}_A(a, b)$ of proofs that a and b are propositionally equal (as it is standard nowadays, we will later just write $a =_A b$ or even $a = b$). Propositional equality is thus an *internal* concept, making the formulation of mathematical theorems involving equality possible.

A caveat it required here. There is an *extensional* form of type theory with the characteristic feature that it does not distinguish between judgmental and propositional equality[1] which makes type checking undecidable. Compared to intensional type theory, the extensional variant has not received as much attention in the literature due to its obvious weakness. In particular, it is of no interest for us and when we talk about MLTT, we always implicitly mean *intensional* MLTT.

For some time, it was unknown whether *uniqueness of identity proofs* (UIP) is derivable, i. e. whether, given p and q of type $\mathsf{Id}_A(a, b)$, one can construct an inhabitant of the type $\mathsf{Id}_{\mathsf{Id}_A(a,b)}(p, q)$. This question was answered negatively by Hofmann and Streicher, who observed that type theory can be interpreted in the category of groupoids [HS96]. They also speculated that there might be models using higher groupoids, and even ω-groupoids, but were lacking an appropriate framework for the construction of such an interpretation.

UIP was often considered desirable: it was believed that a proof that a equals b should be the mere information thereof, without containing additional data. The homotopical view does not only show why UIP can not be derived nevertheless but also helps to explain what its absence means. A type can be seen as a topological

[1] Altenkirch argues that the common name "extensional type theory" is a misnomer for type theory with this so-called *reflection rule*, as "extensional" should better refer to equality that identifies expressions that behave equally.

space, and an equality proof can be understood as a *path* in this space; but paths are, in general, not unique. However, there might be a path between paths, traditionally called a *homotopy*, and higher homotopies between homotopies, and so on, giving a space the structure of a *weak ω-groupoid*. As Lumsdaine [Lum09] and, independently, van den Berg and Garner [BG11] explained, types do indeed carry the structure of a weak ω-groupoid.

In his PhD thesis, Warren [War08] generalised the Hofmann-Streicher groupoid model (see also his article [War11]). Instead of ordinary groupoids, he uses *strict ω-groupoids* to model MLTT. He thereby proves that, for any n, the principle UIP_n can not be derived, where UIP_n is (the judgmental version of) the statement that, for any type A, iterating the process of taking two points and considering their path space $n-1$ times always leads to a type with unique identity proofs. In particular, he shows that having UIP_m for all types is strictly stronger than UIP_n if $m < n$. Voevodsky's model in simplicial sets [Voe10a] can be understood as a further improvement of Warren's construction. Instead of strict ω-groupoids, Voevodsky uses *Kan simplicial sets*, also known as *weak ω-groupoids*.

Let us discuss how a new variant of MLTT, because this is exactly what homotopy type theory is, could have become so popular. While the mathematical community seems to appreciate the existence of proof assistants in principle, their practical usage is still mostly restricted to those subjects that are close to logic, or, looking at the Four Colour Theorem, those cases that require a case analysis so vast that it is unfeasible to do it by hand. Two reasons for that restriction are certainly the vast overhead that formalisations often require, and certain behaviours of type theory that are not understood sufficiently.

However, some years ago, progress in the semantics of MLTT lead to a development that has improved the situation with respect to both of these issues. Traditionally, a number of different views on types existed, including types as sets (Russel [Rus03]) or propositions (Curry and Howard [How80]); see [PW12] for a discussion. In addition to these, Voevodsky [Voe06; Voe10a] and, independently, Awodey and Warren [AW09] noticed that types may also be regarded as, roughly speaking, topological spaces, with the space of paths between two points corresponding to the identity type of two terms. This new interpretation, the details of which needed some time to be worked out, has helped to explain a lot of the behaviour of MLTT regarding equality.

As a side node, we want to remark that another connection between type theory and topology was found much earlier. Very briefly, a set of elements of a type (in whichever sense the notion might be appropriate in a specific setting) can be seen as open if it is semi-decidable whether a given element is a member of the set. In the same vein, if equality (again, in whichever sense it is appropriate) is decidable, then every element forms an open (and closed) set, and the type can be called *discrete*, see Proposition 3.1.1. A canonical reference is Vicker's textbook [Vic96] and various publications, e.g. [Vic99; Vic01; Vic05]. An early and seminal contribution to the development was made by Scott (*Continuous Lattices*, [Sco72]). Regarding more recent work which considers topology and type

theory explicitly, there is various work by Escardó and Xu [XE13; Esc15a], Escardó and Olivia, e.g. [EO10], and Escardó, e.g. [Esc15a; Esc15b].

The ingenious idea that equality proofs can be seen as paths, however, has only come up around 2005 or 2006. In Voevodsky's simplicial set model (presentation by Streicher [Str11], and Kapulkin, Lumsdaine and Voevodsky [KLV12a], extending [KLV12b]) another interesting property is fulfilled: *equivalences* correspond to *equalities* of types. Consequently, it is consistent to assume Voevodsky's *univalence axiom*, which implies that isomorphic structures are actually equal and can directly be substituted for each other. Models that justify the univalence axiom have been a topic of active research. The Hofmann-Streicher groupoid model [HS96] can be seen as the first model of MLTT that had one univalent universe, although the terminology was not used at that time. Inspired by the ideas of Awodey, Voevodsky, and Warren, several new models of MLTT with identity types were discovered, and the construction of such models became a topic of very active research. Apart from those already discussed, we want to mention Arndt and Kapulkin's work on *Homotopy-theoretic models of type theory* [AK11], Garner and van den Berg's *Topological and simplicial models of identity types* [BG12], and Awodey's *Natural models of homotopy type theory* [Awo14].

This seems to be a key concept if we want type theory to be usable by working mathematicians as a tool for formal verification, or even for actually finding proofs, as mathematicians tend to identify isomorphic structures in informal proofs all the time. Hoping that type theory would finally be more accessible for mathematicians outside of the logic spectrum as he used to be himself, Voevodsky continued working on his *univalent foundations program*.

From the programmer's point of view, univalence ensures a form of abstraction that has been absent so far. Consider a type, say, the natural number $\mathbb{N}$, is implemented in two different ways. One could be the standard way, using the constructors zero and succ, while another implementation could use a dyadic (or binary) representation of $\mathbb{N}$. These definitions are equivalent (if performed properly) and every operation that works for one of them will also work for the other; however, traditionally, it has been necessary to reimplement all required functions. The univalence axiom makes the equality between those implementations available internally and all algorithms for one representation can directly be used for the other one as well. It probably should not go unmentioned that there are still problems to be solved here, in particular Voevodsky's *canonicity conjecture*, see [Voe10a], but the recent development of a constructive model in cubical sets by Bezem, Coquand, and Huber [BCH14] (see also the addition [Coq13] and variation [Coq14]) makes the community feel confident that this problem will be solved soon.

Soon after Awodey, Warren and Voevodsky made their ideas public, many researchers from fields that were considered very different from type theory, such as higher dimensional category theory and abstract topology, became fascinated by the surprising connection that allowed to transfer intuition, or even results, from one field to another. Traditional type theorists got excited because of the

striking consequences of the univalence axiom, some of which had been considered feasible (but hard to realise) before. These direct consequences of univalence include function extensionality (considered, e. g., in [Alt99]) and (as described above) an extensional universe [HS96]. The homotopical view later induced the idea of *higher inductive types* (HITs), yielding very well-behaved quotient types (as previously considered in [Men90; Hof95; AAL11]) as a special case. In particular, the wish for properties that previously led to the development of *observational type theory* [AM06; AMS07] are naturally satisfied, or conjectured to be satisfied, in type theory with the univalence axiom. Due to the homotopical nature of the type theory of interest, the broader topic became known as *homotopy type theory* (HoTT). The first public mentioning of this name was possibly Awodey's talk title at PSSL86[2] in 2007. The names *homotopy type theory* and *univalent foundations* have often been used synonymously. However at present, it appears that *univalent foundations* refers mainly to Voevodsky's research program of developing a system to formalise mathematics in.

During the following years, various meetings took place, including a workshop in Oberwolfach [Awo+11]. The steady growth of interest culminated in the year-long special program on univalent foundations at the Institute for Advanced Study in Princeton 2012/13, co-organized by Awodey, Coquand and Voevodsky, with around 60 participants, long- and short-term visitors, with myself being one of them. This was also where *Homotopy Type Theory: Univalent Foundations of Mathematics* [Uni13] was collaboratively written, in the community often referred to as "the HoTT book" or even as "the book", which will serve as our main reference for the basic properties of HoTT that we present in Chapter 2.

Especially during the program in Princeton, but also before and after, a lot of progress was made. In particular, the formalisation of classical homotopy-theoretical theorems was pushed forward. The formalised part of homotopy theory includes the calculation of some homotopy groups of spheres, the van Kampen theorem, the Freudenthal suspension theorem, a restricted form of Whitehead's theorem, the Blakers-Massey theorem, and others, mostly reported in [Uni13].

1.2 A brief introduction to truncation levels and operations

One important aspect that plays a role for nearly everything done in HoTT are the *truncation levels* of types. These are, in effect, an internalised version of the property UIP_n that a single specific type can satisfy. We say that a type X is (-2)-truncated, or *contractible*, if we know a point $x_0 : X$, its *centre*, and we know that every other point is equal to this point. Of course, we here refer to propositional equality.

[2] The 86th edition of the Peripatetic Seminar on Sheaves and Logic, Institut Élie Cartan, Nancy

A type is (-1)-truncated, or *propositional*, if for any two of its inhabitants the path space is contractible. An equivalent way to express this is to say that any two inhabitants are equal, which often is expressed by saying that it has at most one inhabitant. In general, a type is $(n+1)$-truncated if all its path spaces are n-truncated, for $n \geq -2$. It is easy to prove that a type satisfies UIP if and only if it is 0-truncated, and such types are called *sets*. In general, an n-truncated type is also called an n-type.

The notion of n-types, or n-truncatedness, comes from topology and related areas. A primary example are topological spaces: a space is called a *homotopy n-type* if all homotopy groups above degree n are trivial. Similarly, an ω-groupoid can be called n-truncated if it is an n-groupoid. It was Voevodsky who realised that this concept can be expressed in type theory [Voe10a]. Voevodsky's terminology differs slightly from ours, by speaking of *h-levels* (*homotopy levels*) and starting to count at 0. In contrast, we use the terminology that is introduced in our main reference [Uni13], that is, *truncation levels* which start at -2, which matches the traditional numbering of topology. Thus, contractible types can be said to be of truncation level -2 or of h-level 0, and in general, the statement that a type is an n-type (or n-truncated) is identical to the statement that it is of (or has) h-level $(n+2)$; the only difference is terminological. Several fundamental results on h-levels are also due to Voevodsky [Voe10a; Voe10b; Voe13b]. These include the fact that h-levels can be used to characterise functions, in particular ("weak") equivalences. In addition, he has proved that his univalence axiom implies weak function extensionality (dependent function spaces preserve h-levels), and that weak function extensionality implies "naive" function extensionality (which has always been subject to discussions in intensional type theory). Voevodsky also shows that "naive" function extensionality implies strong function extensionality, stating that the canonical map (which, for any functions f and g, shows that $f = g$ implies $f(x) = g(x)$ for all x) is an equivalence.

The truncation level of a type tells us something about its higher homotopical structure. An n-type does not have any interesting structure above level n. It turns out that, in HoTT, we can directly talk about the k-th *loop space* of a type, and thereby indirectly about its k-th *homotopy group*. If $k > n$, the k-th loop space and homotopy group of an n-type are trivial.

An interesting detail is hidden in the previous paragraph: we said that an n-type does not have any interesting structure *above* level n. Indeed, the reason is that for any number $m > n$, any n-type is also an m-type. From a topologist's point of view, this might be surprising at first sight. For example, let us pick $m = 0$ and $n = -1$. As we just said, a type is (-1)-connected if any two of its inhabitants are equal. Topologically, this looks as if it was *path-connected*. However, a path-connected space does not necessarily have the additional property of being *simply connected*, which does not seem to match our claim that a (-1)-truncated type is also 0-truncated. The solution to this lies in the observation that everything we express type-theoretically is automatically stated in a *continuous* way. And indeed, given a space, if we know that any two of its points are connected by a path,

and this function which assigns a path to any two point is continuous in a way that is straightforward to define, then that type is simply connected. Similarly, one can convince oneself that all higher homotopy groups must be trivial as well.

Of particular interest is often the property of being propositional, i.e. (-1)-truncated. As stated above, it means that a type has at most one element. Such a type is often called *proof irrelevant*, although one has to be careful as this notion is slightly ambiguous. In any case, propositional types (or simply *propositions*) correspond to what is called a "proposition" in traditional mathematics, where one does usually not distinguish between different proofs. For example, it is a very subtle question to ask whether there "is an element" in some given type. From the traditional *propositions as types*-point of view, the corresponding type-theoretic statement would simply be the type itself. This means, for a positive answer, one would have to provide an inhabitant of the type, which seems to be more than one was asked for. Similarly, if one asks whether there exists an element of a type fulfilling some given predicate, this would be translated as a a Σ-type. However, one can argue that a Σ-type is more than a simple "exists", as an element also provides a concrete "choice". As a consequence, the set-theoretic axiom of choice becomes under this translation the so-called "type theoretic axiom of choice" which is a tautology (Lemma 2.2.12). However, for propositional types, this mismatch disappears. There is no non-trivial "choice" involved, as one cannot distinguish between two different inhabitants, and giving an inhabitant is therefore appropriate if one is asked to prove that there is an element in a type. Motivated by these considerations, *squash types* (the NuPRL book [Con+86]), and similar, *bracket types* (Awodey and Bauer [AB04]) were introduced in different versions of type theory. These allow to "turn a type into a proposition", namely the proposition that the type is inhabited. Homotopy type theory calls this concept *propositional truncation* or (-1)*-truncation*. More generally, given any number $n \geq -1$, HoTT offers an operation to trivialise ("cut off") its higher structure. These operations are useful when one is not interested in equalities on higher levels and wants to keep them simple. For example, the homotopy groups are defined to be the 0-truncation of the corresponding loop spaces. This is not only a matter of convenience: it can happen that it is impossible to develop a certain theory about some "raw" types while it is possible, and totally sufficient, to develop the same theory for an appropriate truncation of those.

In this thesis, we will present several original results about the truncation properties of types.

1.3 Overview over Our Results

Let us first describe the contents of this thesis in some detail.

The name of Chapter 2 (*Overview over Homotopy Type Theory and Preliminaries*) is self-explanatory. We mostly work in the formal system (or a fragment thereof) that is presented in our main reference, the textbook on homotopy type

theory [Uni13]. We give a very brief introduction to the system, but our presentation is certainly non-exhaustive. We refer a beginner to [Uni13] for a much better introduction. An experienced reader will surely want to skip most of the chapter, possibly apart from our explanation of the proving strategy that we call *Equivalence Reasoning* (Section 2.2.5), and the short clarification regarding terminology in Section 2.4.

In Chapter 3 (*Truncation Level Criteria*), we recall Hedberg's Theorem which says that any type with decidable equality is a set. We give several versions of assumptions that are weaker than decidable equality and still sufficient, deriving a variety of conditions which are equivalent to saying that a type is a set, or *locally* a set (in the sense that all path spaces starting from a fixed base point are propositional). In the second part of that chapter, we formulate the corresponding principles in a way that allow us to use them together with higher truncation levels, and we formulate our *Generalised Local Hedberg Argument* (GLHA). This statement is straightforward to prove. We still consider it a nice result as it has the potential to overcome technical difficulties when trying to prove that a type, in particular a higher inductive type, is n-truncated. We will see a justification for that claim much later in the (first) proof of Lemma 8.10.10, where this argument plays a crucial role.

Chapter 4 (*Anonymous Existence*) deals with weakly constant endofunctions, where we say that a function is weakly constant if it maps any two points to equal points. We show that the type of fixed points of such a map, properly defined, is propositional. We can conclude that a type has a weakly constant endofunction if and only if it is stable with respect to the projection map of the truncation, i.e. if and only if it has *split support*. This allows us to define a new propositional notion of anonymous existence which we call *populatedness*. Four different forms of expressing the inhabitance of a type, namely usual pure inhabitance, truncation, populatedness, and double negation, are carefully defined and statements about their relationships are proved.

We devote Chapter 5 (*Weakly Constant Functions*) to the question whether it is possible to factor a weakly constant function $f : X \to Y$ through the propositional truncation. In the previous chapter we had seen that this is always possible if X and Y are the same type. We give some intuition why it should *not* be expected to be possible in the general case. We then show why it can be done if Y is a set, which also serves as an appetizer for Chapter 8. Finally, we show that a weakly constant function $f : X \to Y$ can be factored if X is the sum of two propositions, implying that the truncation of the sum of two propositions has the universal property of the *join* (which is usually defined as a higher inductive type) even in a weaker theory without higher inductive types.

Chapter 6 (*On the Computation Rule of the Propositional Truncation*) shows a couple of possibly surprising consequences of the judgmental β-rule of the propositional truncation. Not only does it imply that $\|\mathbf{2}\|$ is the interval, which is known to be enough to conclude function extensionality, it also allows us to factor a function *judgmentally* through its propositional truncation, assuming that we know

how to factor it in any propositional way. The most counter-intuitive construction is the term myst which, together with univalence, allows us to seemingly reverse the projection map of the truncation for a non-trivial class of types, including the natural numbers: we have $(\mathsf{myst} \circ |-|) =_{\mathbb{N}\to\mathbb{N}} \mathsf{id}_{\mathbb{N}}$, something that is only possible because $\circ$ is here the dependent function composition operator.

In Chapter 7 (*Higher Homotopies in a Hierarchy of Univalent Universes*) we prove that, in MLTT with a hierarchy $\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots$ of univalent universes, the universe $\mathcal{U}_n$ is not an n-type. At the same time, we construct (for any fixed n) a type that is "strictly" an $(n+1)$-type, i.e. is $(n+1)$-truncated and not n-truncated, without using higher inductive types. This had been an open problem of the special year program at the Institute for Advanced Study in 2012/2013, where I have originally presented the solution. Our construction shows that in particular $\mathcal{U}_n$ is such a strict $(n+1)$-type if we restrict it to n-types. In the presentation of the proof, we develop some fairly simple theory of pointed types. Important ingredients are the observations that the loop space operator Ω commutes in some sense with (pointed versions of) Π and Σ. This leads to (among others) our *local-global looping principle*, which says that a loop in a universe with a type X as basepoint corresponds to a family of loops with basepoints in X. These lemmata also make it possible to complete an alternative approach (due to Finster, Lumsdaine, and Voevodsky) that was discussed at the special year program and which was (to the best of my knowledge) until now not known to work. However, the results that are obtainable from this approach are necessarily weaker compared to those of our own approach. We also show how we can control the connectedness properties of a type with a rather straightforward and technical construction. In total, this allows us to present a type of which only and exactly the n-th homotopy group is non-trivial, constructed without using higher inductive types.

Chapter 8 (*The General Universal Properties of Truncations*) establishes a strong connection between constancy and the propositional truncation, something that is to some extend already foreseen in Chapter 5. The usual universal property tells us that $\|A\| \to B$ is equivalent to $A \to B$, but with the condition that B needs to be propositional. This can make it hard to define a map $\|A\| \to B$ if the latter condition is not met. Weakening the assumption on B, we derive an equivalence of $\|A\| \to B$ and a type of *coherently constant* functions, depending on the truncation level of B. For the general case in which we do not know anything about B, we need the theory to support Reedy ω^{op}-limits in the sense of Shulman [Shu15] ("infinite Σ-types") to formulate the type of constant functions with an infinite tower of coherence conditions. Intuitively, this type corresponds to the type of natural transformations between two presheaves over the index category Δ_+^{op}, in other words, two semi-simplicial types. We can then prove that this type is equivalent to $\|A\| \to B$. In fact, our construction can be seen as a "universal form" of the usual approach of defining a function $\|A\| \to B$ by finding a propositional type Q "in the middle", i.e. such that $A \to Q$ and $Q \to B$ (see [Uni13, Chapter 3.9]). We do all of this without making use of higher inductive types, even though we will explain that these give an alternative way to derive the finite special cases, corresponding

to the construction of the Rezk completion [AKS15]. However, we do not think that the general result can be obtained with this alternative approach unless one makes very strong assumptions on the theory (intuitively, higher inductive types with an infinite number of constructors).

The last part of this thesis is Chapter 9 (*Future Directions and Concluding Remarks*). It contains several results and discussions which are loosely related to the problem of formalising structures such as weak ω-groupoids in HoTT, which (in the theory we consider) is likely to be impossible. Most of the work in this chapter gives rise to potential future research projects. We start by discussing the challenge of defining semi-simplicial types. More generally, we describe the "Reedy"-approach of defining a functor from an inverse category into a universe. It then turns out that it is beneficial if associativity in the index category is strict, and we show that this can be achieved in the case of Δ_+, which is needed for semi-simplicial types. Further, we think that it should be possible to derive elimination principles for higher truncations, similar to those proved in Chapter 8. However, in this thesis, we only derive a special case and show that functions from the n-truncation of a type A into some $(n+1)$-type B correspond to functions $A \to B$ which are weakly constant on the $(n+1)$-st path spaces. We give two different proofs for this, one of which uses higher inductive types and is somewhat related to the Rezk completion [AKS15]. The next topic is on what we call *Yoneda groupoids*, an attempt to construct a class of weak ω-groupoids using that the universe already has such a structure. While this works, it does not seem to be particularly helpful as not many interesting ω-groupoids can actually be defined in that way. Further, we analyse an idea by Altenkirch that we call *set-based representation* of groupoids. The question is whether one can translate between 1-types and groupoids that are represented by giving their set of points and a family of sets of morphisms. We prove that it is not possible to do what Altenkirch had originally hoped for, but we show how a slightly weaker construction can in some cases be achieved.

After outlining these potentially interesting projects of future research, we make some additional notes on related work and summarise the contents of the thesis.

I consider my two main contributions to the field of homotopy type theory to be

- Theorems 7.4.7 and 7.4.8, our results that the univalent universe $\mathcal{U}_n$ in MLTT is not an n-type, and $\mathcal{U}_n^n$ (which is $\mathcal{U}_n$ restricted to n-types) is a strict $n+1$-type.

- Theorem 8.8.5, the *General universal property of the propositional truncation*, which says that the type $(\|A\| \to B)$ is equivalent to the type of coherently constant functions from A to B, written $(A \xrightarrow{\omega} B)$, in any type theoretic fibration category with Reedy ω^{op}-limits. Related is Theorem 8.9.6 which shows the finite version of the statement in standard MLTT with propositional truncations: if B is n-truncated, then the type $(\|A\| \to B)$

is equivalent to the type of functions $(A \to B)$ which are constant with n coherence conditions.

Apart from these, selected original results that I prove in this thesis are the following, in order of occurrence:

- Theorem 3.2.1, the *Generalised Local Hedberg Argument*: a very simple, but powerful statements that helps to analyse the truncation property of a type, in particular a higher inductive type.
- Main Lemma 4.1.1, the *Fixed Point Lemma* which says that the type of fixed points of a weakly constant endofunction is propositional, and its immediate consequence Theorem 4.1.4, proving that a type has a weakly constant endofunction if and only if it has split support.
- Theorem 5.2.6, which proves that the truncation of the sum of two propositions has the universal property of the *join*, even in a theory without higher inductive types.
- Theorem 6.4.6, the *Myst Puzzle* which seems to yield an inverse that cannot exist, using the computational properties of the truncation in a clever way.
- Main Lemma 7.4.2, our *Local-Global Looping Principle*, expressing that an $(n+2)$-loop in the universe with base point X is the same as a family of $(n+1)$-loops in X.
- Theorem 7.5.4, an alternative construction of a strict $(n+1)$-type of which we know the n-th loop space explicitly.
- Theorem 7.7.1, the construction of a type that has exactly one non-trivial homotopy group on level n from univalence alone.
- Main Lemma 8.5.1, showing the (intuitive but technically tedious) fact that the projection from the n-dimensional tetrahedron, built out of paths of level 0 to n, to any of its horns is an equivalence.
- Theorem 8.10.2, which shows that the function space $(\|A\|_n \to B)$ for any $(n+1)$-type B is equivalent to the type of functions from A to B which are weakly constant on the n-th loop space.
- Proposition 9.2.1, a short observation that the category of finite sets and (strictly) increasing functions can be implemented such that the categorical laws hold strictly, provided that we have η for Σ-types.
- Theorem 9.4.7, a simple but nice statement that types with braided loop spaces are *reduced set-based representable*. This means such a type can be "split" in a set of points and, for every point, a type representing its loop space.

1.4 Computer-Verified Formalisations

This thesis is supplemented by an electronic appendix containing formalisations of all results that are marked with the symbol Ⓐ in *Agda* [Nor07].[3] We want to use this section to give some details.

One important virtue of MLTT and HoTT are that they provide possible foundations of mathematics which can be implemented directly, allowing the computer-supported and machine-checked development of proofs. Agda is the implementation of an MLTT-style type theory that we use to approximate the theory that we are working.

Not all of our results can be formalised in this way. We call a statement *internal* if it can be expressed and proved in the formal system itself. Some of our results are purely internal, especially those of Chapters 3 to 5. Similarly, all basic lemmata listed in Chapter 2 are internal. Almost all of these internal results are formalised in the electronic appendix.

In contrast, the results of Chapter 6 are of meta-theoretic nature, at least in the form in which we have presented them. This means that they are statements about the type theory "from the outside". In the theory itself, we cannot talk about these results, and consequently, we cannot formalise them in the style of internal results. However, Agda can still help us to check whether some equality holds judgmentally by testing whether refl makes it type-check. This allows us to include these results in the electronic appendix as well; for details, see the beginning of Chapter 6.

The main result of Chapter 8, and parts of the contents of Chapter 9, are meta-theoretic as well. We would be very happy if we could express them completely in the theory we work in; however, we strongly believe that this is impossible. Sometimes, it is possible to construct a *family* of internal results; for example, for every natural number n, we can construct the type of n-truncated semi-simplicial types (see Chapter 9). These "families of internal constructions" are often uniform enough to be generated mechanically, in the sense that it is possible to write a program (in any language) that takes an element of the indexing type and produces the Agda code of the corresponding internal statement. In the case of semi-simplicial types, we have written and experimented with such a short program in Haskell [Kra14a]; for some more details, see Proposition 9.2.1 and the discussion proceeding it. What we can *not* do (or believe to be impossible) is constructing a function in the theory which maps any natural number n to the type of n-truncated semi-simplicial types.

Chapter 7 has a special status. Technically, it also contains a family of internal results, indexed over $\mathbb{N}$. However, Agda allows us to quantify over universe levels, and even to eliminate into the type of universe levels. This is not possible in homotopy type theory, but it allows us to formalise the results of Chapter 7 which

[3] Ⓐ stands for *electronic Appendix*, and can alternatively be read as *Agda*.

would not be possible in this form otherwise. Details can be found in Remark 7.1.2. We do not take advantage of this possibility in any other chapter.

A further potentially controversial feature which Agda formalisations typically use is the judgmental η-rule (or *uniqueness* principle) for Σ-types that Agda implements; i.e. $(\mathsf{fst}(x), \mathsf{snd}(x))$ is for Agda judgmentally equal to x if x is an inhabitant of $\Sigma\,(a : A)\,.\,B(a)$. This does not seem to be crucial in any way for our formalisation, i.e. very minor modifications would be sufficient to make the formalisation type-check if we had a version of Agda without the judgmental η-rule for Σ-types. The corresponding propositional equality can be proved easily, and it seems to be a rather arbitrary choice of [Uni13] to not include the judgmental rule. Indeed, in Appendix A.2.5, the authors state: "Notice that we don't postulate a judgmental uniqueness principle for Σ-types, even though we could have". Unlike Agda, Coq does not implement these judgmental rules.

One further difference between the theory we work in and the theory Agda implements concerns the treatment of universes (see Section 2.1.4). While HoTT universes are *cumulative*, i.e. $A : \mathcal{U}$ and $\mathcal{U} : \mathcal{U}'$ imply $A : \mathcal{U}'$, Agda requires explicit *lifting*. A sour consequence of this is the following: the univalence axiom (see Section 2.3.1) implies $(A = B) =_{\mathcal{U}_{k+1}} (A \simeq B)$ for types $A, B : \mathcal{U}_k$. Note however that this cannot be stated in Agda, the reason being that $A = B$ lives in $\mathcal{U}_{k+1}$, while $A \simeq B$ lives in $\mathcal{U}_k$. We can still make this statement by first lifting $A \simeq B$ to the universe $\mathcal{U}_{k+1}$. Fortunately, this difference does not affect us, apart from the formalisation of Chapter 7. In that part, we strive to represent (pointed) type equality by (pointed) equivalence wherever possible in the formalisation so that we can avoid manifold instances of lifting which would make the formalisation unreadable.[4] In a theory with proper cumulativity of universes, both versions work equally well.

The formalisations in the electronic appendix type-check with Agda 2.4.2, which in particular uses the implementation of *without-K* by Cockx, Devriese, and Piessens [CDP14]. We make use of the community's Agda library [Hagda]; however, for compatibility and convenience, we include all the relevant files in the electronic appendix so that no further material is needed to check the results. We also include a browser-viewable version, produced with Agda's *html* feature. We have given our best to produce a formalisation that is as readable as possible, hopefully even for non-experts. The reader who is not familiar with Agda, or who simply does not have an Agda installation at hand, is invited to look at the html

[4] Another possibility to resolve all issues related to non-matching universe levels would have been Agda's optional flag *type-in-type*, which, for example, Licata's Agda library [Lic12] uses. This flag allows the judgment $\mathcal{U}_i : \mathcal{U}_i$, which enables the user to work completely in the lowest universe $\mathcal{U}_0$. The price is, of course, that one works in an inconsistent theory. In many cases it seems to be clear that the *type-in-type* flag is only used for convenience, and the proofs could be translated into Agda without this option turned on. However, it seems questionable whether a result that is formalised in such a setting can really be considered "formally verified", and all statements that involve any form of impredicativity become necessarily highly suspicious. We therefore refrain from turning on *type-in-type*.

version of the formalisation. It allows to read the completely hyperlinked code without any specialised tools; all that is necessary is a web browser.

1.5 Declaration of Authorship and Previous Publications

I want to stress that many of the results have been found in collaboration with fellow researchers. I have published Section 3.1 and Chapter 4 together with Martín Escardó, Thierry Coquand, and Thorsten Altenkirch as **Generalizations of Hedberg's Theorem** [KECA13] at *Typed Lambda Calculi and Applications* (TLCA) 2013.

As a contribution to TLCA's special issue, we have submitted the largely extended article **Notions of Anonymous Existence in Martin-Löf Type Theory** [KECA14] to *Logical Methods in Computer Science* (LMCS), which additionally includes most of Chapters 5 and 6 (except some minor additions such as Example 5.2.5).

The main contents of Chapter 7, authored together with Christian Sattler, have been published as **Higher Homotopies in a Hierarchy of Univalent Universes** [KS15] in *Transactions on Computational Logic* (TOCL).

Finally, the main results of Chapter 8 (the contents up to Section 8.8) are to appear in the *TYPES'14* post-proceedings as **The General Universal Property of the Propositional Truncation** [Kra14b].

The remaining work in this thesis has not been published, but I want to say that early attempts to define semi-simplicial types (described in Section 9.2) have been done mostly together with Nuo Li. Further, several contents and discussion especially in Sections 8.10 and 9.2 are joint work with (or come from discussions with) Paolo Capriotti. This is in particular true for Theorem 8.10.2, for which we give two proofs. The second one is based on an argument found by Andrea Vezzosi (see Section 8.10.2). Of course, details and acknowledgements of contributions from researchers different from myself will always be given in the relevant parts of the thesis.

During my time as a PhD student, I have obviously benefited much from countless discussions with numerous people, especially with my supervisor Thorsten Altenkirch, Paolo Capriotti, Martín Escardó, Ambrus Kaposi, Nuo Li, and Christian Sattler, but also other current or former members of the *functional programming lab* in Nottingham, participants of the *univalent foundations special year program* of the Institute for Advanced Study in Princeton 2012/13, members of the *semantics of proofs and certified mathematics* thematic trimester at the Institut Henri Poincaré, and researchers at various other events. Many of these discussions have certainly influenced various results in this thesis more than I am aware of.

Parts of the Agda formalisations in the electronic appendix are based on formalisations that originally served to supplement the publications described above, and these parts are thus joint work as well.

Chapter 2

Overview over Homotopy Type Theory and Preliminaries

The first part of this chapter serves as a rough overview over the formal system that is often referred to by *Martin-Löf type theory*. We keep it very concise as an excellent extensive introduction of exactly the material that we want to present already exists in our standard reference [Uni13, Chapter 1]. We are aware that our presentation can not serve as an introduction to type theory for a newcomer; our main aim here is to clarify which theory we work in. Should there remain any ambiguities, the appendix of the mentioned reference can be consulted as it contains a formal presentation of the specific type theory we use. In particular, we do not elaborate on *contexts*, *substitutions*, *typing judgments*, *formation*, *introduction* and *elimination rule* of types, and similar notions. Instead, we refer to Hofmann's introduction [Hof97]. We also do not give details about the assumed *computation rules*, but the ones we use are standard and can, again, be found in [Uni13] if required. Following the terminology of that reference, we call the non-dependent elimination principle of a type its *recursion* principle, and the dependent one its *induction* principle.

Although shortly introduced as a concept of MLTT, we devote the second part of this chapter to propositional equality. We explain the groupoidal structure that the equality type carries, together with many related constructions. The crucial notions of truncation levels and loop spaces are discussed in slightly higher detail. We also explain what we call *equivalence reasoning*, a very simple (and obvious) proving technique that is far more powerful that one might think. In this thesis, this technique will be used numerous times.

In the third part of the current chapter, we introduce univalence, higher inductive types, and in particular truncation operations.

Finally, in Section 2.4, we explain how we treat some notions that, in the context of HoTT, could potentially be ambiguous.

2.1 Martin-Löf Type Theory

The most basic and most important statement in type theory is as simple as

$$a : A, \tag{2.1}$$

meaning that the *term* a is of *type* A. We give a brief introduction to the basic types and type formers of MLTT.

Let us shortly repeat possible semantics that we mentioned in Section 1.1. As [PW12] recalls, (2.1) was understood as "a is an element of the set A" by Russel [Rus03], and as "a is a solution to the problem A" by Kolmogorov [Kol32], later refined to "a is a proof of the proposition A". The latter is known as the famous Curry-Howard isomorphism [How80]. This interpretation allows us to say that we have *proved* a statement if we actually have *constructed* an inhabitant of a type. We will discuss below what the type formers that we introduce stand for under this view.

2.1.1 The Unit Type

The *unit type*, written $\mathbf{1}$, could be viewed as the most basic type: it has exactly one inhabitant $\star : \mathbf{1}$. If we have to find an inhabitant of a type and we can show that this type is *equivalent* or *isomorphic* (the concrete meaning of which will be introduced below) to $\mathbf{1}$, we are done, as we always have the inhabitant $\star$. On the other hand, having an element of $\mathbf{1}$ as an assumption is a particularly useless information. The unit type is neutral (in some appropriate sense) in many situations. As trivial as it may sound, showing that a type is equivalent (see Section 2.2.2) to $\mathbf{1}$ is a very powerful proving technique, as we will see plenty of times. Under the propositions-as-types view, the unit type corresponds to the statement that is always true, independent of any other assumptions.

2.1.2 The Empty Type

Written $\mathbf{0}$, the *empty type* is from some point of view the opposite of the unit type: under the propositions-as-types interpretation, $\mathbf{0}$ stands for the statement that is always false. Having an inhabitant of the empty type can be understood as having found a *contradiction*, from which we can construct an inhabitant of any given type (including $\mathbf{0}$). This principle is known as *ex falso quodlibet*, "from a falsehood, anything follows", or as $\mathbf{0}$-*elimination*. Technically, this principle is the empty type's *non-dependent* elimination, i.e. its *recursion* principle, but the dependent one hardly ever occurs naturally and can be derived (using that $\mathbf{0}$ is propositional in the sense of the definition given in Section 2.3).

2.1.3 Function Types

Given types A and B, there is the type of *functions* from A to B, written $A \to B$. $\to$ can, under the propositions-as-types view, be read as *implies*. Thus, if we have to construct an element of $A \to B$, we may say that we have to show that A implies B. If we have a term t, depending on a variable a of type A, we get a term $\lambda a.t : A \to B$ ("introduction rule" for the function type). On the other hand, given $f : A \to B$ and some $a : A$, we get $f(a) : B$ ("elimination rule").

Going back to our explanation of $\mathbf{0}$, we can now express what we meant in the informal description above. For any type A, the recursion principle of $\mathbf{0}$ gives us a term $\mathbf{0} \to A$.

If we want to apply the introduction rule, but do not want to give an explicit name to the bound variable, we may write $\lambda\_.t$ instead of $\lambda a.t$; this can sometimes improve readability. Instead of $A \to \mathbf{0}$, we write $\neg A$, "not A".

2.1.4 Universes

In (our version of) MLTT, types can be seen as terms, living in some *universe*. At the same time, we want to view a universe as a type again, and it should therefore live in a universe itself. However, if a universe lived in itself, we could derive a contradiction (*Russel's Paradox*). MLTT therefore uses a hierarchy of universes,

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \dots, \tag{2.2}$$

where every universe $\mathcal{U}_n$ lives in the next universe $\mathcal{U}_{n+1}$. We do *not* assume that there is a universe $\mathcal{U}_\infty$. Our universes are cumulative in the sense that if we have $A : \mathcal{U}_k$ and $\mathcal{U}_k : \mathcal{U}_m$, we also have $A : \mathcal{U}_m$. Most of the time, we will write $\mathcal{U}$ for the universe that we are talking about, symbolising that we use a generic universe, or, for simplicity, just the lowest universe $\mathcal{U}_0$. In principle, $\mathcal{U}$ could stand for any "type of types" which is closed under all type formers, not only for the primitives of the type theory.

We can now introduce the notion of a *dependent type*, or *type family*, which is just a term $B : A \to \mathcal{U}$ for some type A. Note that this notion can also be considered in a theory without universes, but, having universes at hand, it becomes considerably simpler.

2.1.5 Dependent Functions

The type former for *dependent functions*, written Π, is a generalisation of the type former $\to$. Given a type $A : \mathcal{U}$ and a family $B : A \to \mathcal{U}$, we have the type $\Pi_{a:A} B(a) : \mathcal{U}$. An inhabitant is a function f such that $f(a) : B(a)$ for all $a : A$. The elimination and introduction rules correspond to those of $\to$.

We sometimes write $\Pi_A B$ instead of $\Pi_{a:A} B(a)$, taking care that we only do so if it does not cause confusion and improves readability. Under the propositions-as-types view, $\Pi_{a:A} B(a)$ may be understood as "for all a, the statement $B(a)$ holds",

and therefore, we also write $\forall(a : A).\, B(a)$ or even $\forall a.\, B(a)$, depending on the context, if we think that it improves the readability.[1]

2.1.6 Products or Pairs

Given types A and B, we write $A \times B$ for their *(cartesian) product*. This type is also called the type of *pairs* arising from A and B. In order to construct an element of that type, we can (using the product's *introduction rule*) separately find an inhabitant of A and an inhabitant of B. At the same time, the induction principle of this type former tells us that, if we are given $x : A \times B$, we can treat x as if it was such a pair. In particular, we get $\mathsf{fst}(x) : A$ and $\mathsf{snd}(x) : B$.

Under the propositions-as-types view, the product corresponds to a conjunction. We also observe that $\mathbf{1}$ can be understood as a product with zero components (the nullary product type).

2.1.7 Dependent Pairs

Dependent pairs generalise pair types in a similar way as dependent functions generalise function types. The difference is that the second component may depend on the first. Let A be a type and $B : A \to \mathcal{U}$ be a type family. Then, we write $\Sigma\,(a : A)\,.\,B(a)$ for the corresponding dependent pair type. Concerning the *introduction rule*, we can construct an element of $\Sigma\,(a : A)\,.\,B(a)$ by giving $a : A$ together with $b : B(a)$. Turning this around, the induction principle tells us that, given $x : \Sigma\,(a : A)\,.\,B(a)$, we may assume that x is a pair of an $a : A$ and a $b : B(a)$.

Note that we do *not* call these types "dependent products". This would be highly ambiguous as that name is also used for what we call "dependent function types". We also abstain from using the description "dependent sums". While the latter would not be problematic, we think that "dependent pairs" is the most accurate naming. A potential explanation for the name clashes follows from the discussion in Section 2.1.12.

The symbol Σ can be understood as a strong existential quantifier. We can read $\Sigma\,(a : A)\,.\,B(a)$ as "there is an $a : A$ such that $B(a)$".

Note that, notationally, we treat Π and Σ very differently. While we follow the notation of [Uni13] for Π and write $\Pi_{a:A} B(a)$, we do *not* follow [Uni13] with respect to the notation of Σ, as we do not want to write $\Sigma_{a:A} B(a)$. This is because we view Σ as the dependent version of $\times$ (again, see Section 2.1.12 for an alternative view), with two parts of the same value. In particular, we will often consider nested Σ-types with more than two components, which we want to write in the form

$$\Sigma\,(a : A)\,.\,\Sigma\,(b : B(a))\,.\,\Sigma\,(c : C(a,b))\,.\,D(a,b,c), \tag{2.3}$$

and it would be very bad with respect to the intuition if we wrote all apart from the very last component as subscripts. In some cases, it will be helpful to have a

[1] Caveat: $\forall$ does **not** indicate propositional truncation!

name for the last component of a Σ-type, in which case we may write

$$\Sigma\,(a:A)\,.\,\Sigma\,(b:B(a))\,.\,\Sigma\,(c:C(a,b))\,.\,(d:D(a,b,c)) \tag{2.4}$$

2.1.8 Coproducts

For types A and B, their *coproduct* $A+B$ (sometimes called their *disjoint union* or *sum*) is the type of elements which either come from A or from B. More precisely, we have two functions

$$\mathsf{inl} : A \to A + B \tag{2.5}$$
$$\mathsf{inr} : B \to A + B, \tag{2.6}$$

where we keep the type information, that inl and inr should technically be annotated with, implicit. On the other hand, if we are given an element of a coproduct, we may always do the two cases separately. We may regard $\mathbf{0}$ as the coproduct with zero components, which both explains its elimination principle and the absence of its introduction rules.

Under the propositions-as-types view, the coproduct can be understood as a disjunction.

2.1.9 Booleans

$\mathbf{2}$, the type of booleans, has exactly two inhabitants, written $0_{\mathbf{2}}$ and $1_{\mathbf{2}}$. It could be defined as $\mathbf{1}+\mathbf{1}$ but we choose to give it a separate name because of its frequent usage. If we are given $x : \mathbf{2}$, we may always assume the two cases that x is $0_{\mathbf{2}}$ or x is $1_{\mathbf{2}}$ separately.

2.1.10 Natural Numbers

The natural numbers type $\mathbb{N} : \mathcal{U}$ is probably the simplest type with an infinite number of pairwise distinguishable inhabitants. As usual, we present it as the inductive type that is generated by $0 : \mathbb{N}$ and $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$. Its recursion principle is powerful enough to define all the standard function such as addition (which we write as + if there is no risk of confusing it with the coproduct of types), and its induction principle is probably the most famous version of induction: for a given family $P : \mathbb{N} \to \mathcal{U}$, it is enough to construct $p_0 : P(0)$ and a term $p_s : \Pi_{n:\mathbb{N}} P(n) \to P(\mathsf{succ}(n))$ in order to construct an inhabitant of $\Pi_{\mathbb{N}} P$.

In standard implementations of MLTT, exactly one of the two expressions $n+1$ and $1+n$ is judgmentally equal to $\mathsf{succ}(n)$, depending on whether addition is defined by recursion on the first or the second argument. We prefer to use the latter version as it allows us to replace the somewhat clumsy expression $\mathsf{succ}(n)$ by $n+1$, and the latter looks slightly more natural than $1+n$ from the point of view of traditional mathematical style.

2.1.11 Identity Types

In MLTT, we can express that two inhabitants of the same type are *propositionally equal*, a concept that is not to be confused with *judgmental* equality. For any given type $A : \mathcal{U}$, we have a type family

$$\mathsf{Id}_A : A \times A \to \mathcal{U}, \tag{2.7}$$

and we call an inhabitant of $\mathsf{Id}_A(a_1, a_2)$ a *proof of equality* (or a *path*, as explained in the section about HoTT below) between a_1 and a_2. Instead of $\mathsf{Id}_A(a_1, a_2)$, we write $a_1 =_A a_2$ and, if the type A can be inferred or plays no role, often just $a_1 = a_2$, which corresponds to the terminology of many authors in the area of HoTT. Instead of $\neg(a_1 = a_2)$ (which is already a short-hand notation for $a_1 = a_2 \to \mathbf{0}$), we may write $a_1 \neq a_2$.

The *equality* or *identity* type in MLTT is an inductive type which has only one constructor (for every type). Given $A : \mathcal{U}$ and a point $a : A$, we get $\mathsf{refl}_a : a = a$. If a can be inferred, we allow ourselves to write $\mathsf{refl} : a = a$. Its dependent elimination (or induction) principle reflects the fact that there is only one constructor. However, as the equality type Id_A is parametrised *twice* over the type A, while the constructor refl takes only *one* argument, the situation is very different from the situation of other inductive types such as $\mathbb{N}$. In fact, the equality type is the simplest (and certainly by far the most interesting) case in which this situation occurs.

The dependent elimination principle for equality, traditionally called J (although we will follow [Uni13] and call it *path induction* which matches the rest of our terminology better and, we hope, is intuitively clearer), says that whenever we have a type A and a predicate

$$P : (\Sigma\,(a_1, a_2 : A)\,.\,a_1 = a_2) \to \mathcal{U}, \tag{2.8}$$

it is enough to construct

$$\Pi_{a:A} P(a, a, \mathsf{refl}_a) \tag{2.9}$$

in order to get an inhabitant of

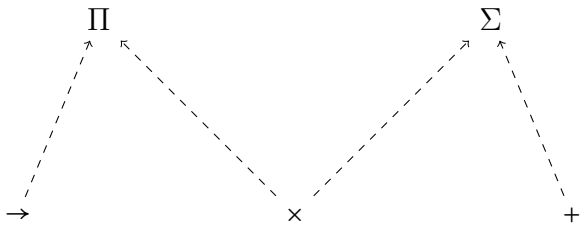
$$\Pi_{a_1,a_2:A} \Pi_{p:a_1=a_2} P(a_1, a_2, p). \tag{2.10}$$

We will elaborate more extensively on some basic constructions of the identity type Section 2.2 below.

2.1.12 Generalisations Graph

As we have mentioned above, the type $\mathbf{2}$ can be constructed as $\mathbf{1} + \mathbf{1}$. What is potentially more interesting is that some type formers can be expressed in terms of others. Assuming that $\mathbf{2}$ is given, the complete graph for the type formers discussed above looks as follows, where † -↠ ‡ means that † can be defined using

‡. However, note that the judgmental properties of the derived type former can, depending on the precise formulation of the theory, be weaker than those that we expect:



It is clear that "→" is a special case of Π. To see that "×" is also a special case, assume types A and B are given. Define $C : \mathbf{2} \to \mathcal{U}$ by $C(0_{\mathbf{2}}) :\equiv A$ and $C(1_{\mathbf{2}}) :\equiv B$ and note that $A \times B$ is equivalent to $\Pi_{\mathbf{2}} C$.

Then again, it is immediate that "×" is a special case "of Σ", where we only need to take the second component to be the constant type family. Regarding "+", given A and B, define C as above and observe that $A + B$ is equivalent to $\Sigma\,(x : \mathbf{2})\,.\,C(x)$.

2.2 Constructions with Propositional Equality

In addition to the possible semantics of type theory that we repeated at the beginning of the above Section 2.1, we can interpret the statement

$$a : A \tag{2.11}$$

as "A is (some kind of) a topological space, and a is a point in this space". As said in Section 1.1, this possibility was described first by Awodey and Warren [AW09] as well as Voevodsky [Voe10a]. An inhabitant of $a_1 = a_2$ can then be seen as a *path* from the *point* a_1 to the *point* a_2, and for "paths on higher levels", it makes sense to speak of *homotopies*.[2]

2.2.1 Basic Functions

If $B : A \to \mathcal{U}$ is a type family, we interpret it as a *fibration*. That is, we think of B being a bigger space "over" the smaller space A, with a map from the bigger to the smaller space that has the topological property of a fibration (see [Hat01]). If a is a point in A then $B(a) : \mathcal{U}$ corresponds exactly to the inverse image of a under this fibration.

Having this interpretation in mind, it is easy to understand what path induction really does. Suppose we have a type family that depends on a triple

[2] Of course, a path in topology is nothing else than a homotopy between two functions which have the one-point space as their domain, and a homotopy in general may be called a path in some appropriate function space.

$(a_1, a_2, p) : \Sigma\,(a_1, a_2 : A)\,.\,a_1 = a_2$, and assume we are given such a triple. In the semantics we sketched, this triple is a pair of points and a path in between. Just by "shrinking" the path and "pulling" the second endpoint a_2, we can *continuously* transform it to the triple corresponding to $(a_1, a_1, \mathsf{refl}_{a_1})$.

What we have just described, namely that we can leave one endpoint fix and move only the second point, motivates a very useful alternative formulation of path induction. It was originally given by Paulin-Mohring [PM93], who observed that an eliminator for equality can be stated *locally* at a fixed point $a_0 : A$. We call the corresponding principle *based path induction*: given $a_0 : A$ and a predicate

$$Q : (\Sigma\,(a : A)\,.\,a_0 = a) \to \mathcal{U}, \tag{2.12}$$

it is enough to construct an inhabitant of $Q(a_0, \mathsf{refl}_{a_0})$ in order to get an inhabitant of

$$\Pi_{a:A}\Pi_{p:a_0=a}Q(a, p). \tag{2.13}$$

Path induction and based path induction can be derived from each other, as shown by Altenkirch and Goguen (see [Uni13, Chapter 1.12.2]). More often than not, the latter is slightly easier to apply.

There is a very useful special case of path induction. Assume again that $B : A \to \mathcal{U}$ is a family indexed over A. We then claim

$$\Pi_{a_1,a_2:A}(a_1 = a_2) \to B(a_1) \to B(a_2), \tag{2.14}$$

and the proof of this *substitution* property is immediate by path induction: the type family we use is $P(a_1, a_2, p) :\equiv B(a_1) \to B(a_2)$ which is independent of the actual path. Following the notation of [Uni13], we write $\mathsf{transport}^B$ for the derived term, and we usually omit the two points a_1 and a_2. Thus, for $p : a_1 = a_2$ and $b_1 : B(a_1)$, we would write $\mathsf{transport}^B(p, b_1) : B(a_2)$. If B is clear from the context, we also write $p_* : B(a_1) \to B(a_2)$ and $p_*(b_1) : B(a_2)$, respectively. Homotopically, $\mathsf{transport}^B$ does not more than using the property that we regard the type family B as a fibration. $b_1 : B(a_1)$ lies "over" a_1 and we have a path p in the base space, so we can construct $\mathsf{transport}^B(p, b_1) : B(a_2)$.

As Hofmann and Streicher have described [HS96], the identity type carries the structure of a *groupoid*, i.e. a category in which every morphism is an isomorphism. For $p : a_1 = a_2$, we have an *inverse* $p^{-1} : a_2 = a_1$, and, if we additionally have $q : a_2 = a_3$, we can *compose* p and q to get $p \cdot q : a_1 = a_3$, with refl acting as the neutral element. The corresponding groupoid laws between them, such as *associativity*, hold again up to propositional equality.[3] For any two paths we have again a path space, giving types the structure of weak ω-groupoids [Lum10; BG11].

As [Uni13, Chapter 2.2] emphasises, functions play the role of functors under this view. If we have a function $f : A \to B$ and points $a_1, a_2 : A$, we can define a function

$$\mathsf{ap}_f : (a_1 =_A a_2) \to (f(a_1) =_B f(a_2)) \tag{2.15}$$

[3] Depending on the exact definitions, some laws will typically hold judgmentally.

by path induction. This also exists if $B : A \to \mathcal{U}$ is a type family and $f : \Pi_A B$ a dependent function. We then get

$$\mathsf{apd}_f : \Pi_{p:a_1=a_2} p_*(f(a_1)) =_{B(a_2)} f(a_2). \tag{2.16}$$

Note that the codomain of apd_f involves transporting along p. The reason is that it has to be the type of paths "lying over" p. In [Uni13, Chapter 6.2], the terminology of a *type of paths lying over a given path* is introduced, and for $a_1, a_2 : A$, $p : a_1 = a_2$ and $b_1 : B(a_1)$, $b_2 : B(a_2)$, they write

$$\left(b_1 =_p^B b_2\right) :\equiv \left(p_*(b_1) =_{B(a_2)} b_2\right), \tag{2.17}$$

which we will adopt in some situations when it increases the readability. Note that the type B gets moved to the upper right corner of the equality sign to make space for the path p. Using this terminology, (2.16) becomes

$$\mathsf{apd}_f : \Pi_{p:a_1=a_2} f(a_1) =_p^B f(a_2). \tag{2.18}$$

The functions ap_f, apd_f and $\mathsf{transport}^R$ behave functorial in the appropriate sense. For details, we refer to [Uni13, Lemma 2.2.2 and all Lemmata in Chapter 2.3]. The following result is also very useful. It tells us how transportation of a path along another path works:

Ⓐ **Lemma 2.2.1** ([Uni13, Theorem 2.11.3], [KECA13])**.** *Let $A, B : \mathcal{U}$ be two types. Assume $h, k : A \to B$ are two functions, $a_1, a_2 : A$ points and $t : a_1 =_A a_2$ as well as $p : h(a_1) =_B k(a_1)$ paths. Then, transporting p along t can be expressed as a composition of paths:*

$$t_*(p) = (\mathsf{ap}_h t)^{-1} \cdot p \cdot \mathsf{ap}_k t. \tag{2.19}$$

Proof. This is immediate by path induction on t, using the functoriality of all involved functions. □

Even if the latter proof is trivial, the statement is essential. For example, in the proof of Main Lemma 4.1.1, we need a special case in which a_1 and a_2 are (judgmentally) the same. However, this special version cannot be proved directly.

2.2.2 Type Equivalences

Given types A and B, let us mimic the traditional mathematical definition of what it means if two spaces are homotopy equivalent. It turns out that it corresponds exactly to the definition of an *isomorphism* between A and B in MLTT, that is a 4-tuple with four components

$$f : A \to B \tag{2.20}$$
$$g : B \to A \tag{2.21}$$
$$\eta : \Pi_{a:A} g(f(a)) = a \tag{2.22}$$
$$\varepsilon : \Pi_{b:B} f(g(b)) = b. \tag{2.23}$$

In particular, we could say that f is an *isomorphism* if we are able to "fill in" the other three components. However, this type is not very well-behaved in the sense that it is generally not propositional. We therefore add a fifth component which makes sure that the components η and ε "fit together".

Given a function f (as in (2.20)), we define the type stating that f is an *equivalence*, written $\mathsf{isequiv}(f)$, to be the (nested) dependent pair type with four components. The first three of those components are (2.21 - 2.23), and the last is

$$\tau : \Pi_{a:A} \mathsf{ap}_f(\eta a) = \epsilon(f(a)). \tag{2.24}$$

In particular, we say that A and B are equivalent, written $A \simeq B$, if there is some f that is an equivalence:

$$(A \simeq B) :\equiv (\Sigma (f : A \to B) . \mathsf{isequiv}(f)). \tag{2.25}$$

The crucial property of this definition is that for any f between any two types, the type $\mathsf{isequiv}(f)$ is propositional. An important result is that f is an equivalence if and only if it is an isomorphism in the weaker sense above, that is, if and only if we can find the first three components: an "inverse" g and two proofs that they are really mutually inverse [Uni13, Theorem 4.2.3].

We give two examples for equivalences: first, for any type A, we have

$$\mathsf{id\text{-}equiv} :\equiv (\mathsf{id}_A, \mathsf{e}_{\mathsf{id}}) : A \simeq A, \tag{2.26}$$

where e_{id} is the canonical (and, up to propositional equality, unique) proof that the identity function is an equivalence. Second, we have a function $\mathsf{swap} : \mathbf{2} \to \mathbf{2}$ defined by $\mathsf{swap}(0_\mathbf{2}) :\equiv 1_\mathbf{2}$ and $\mathsf{swap}(1_\mathbf{2}) :\equiv 0_\mathbf{2}$. It is easy to see that swap is self-inverse and to construct $\mathsf{e}_{\mathsf{swap}} : \mathsf{isequiv}(\mathsf{swap})$, yielding one of the simples non-identity equivalences:

$$\mathsf{swap\text{-}equiv} :\equiv (\mathsf{swap}, \mathsf{e}_{\mathsf{swap}}) : \mathbf{2} \simeq \mathbf{2}. \tag{2.27}$$

Similar to how equality induces a (higher) groupoid structure on any type, equivalence induces a groupoid structure on the universe.

A thorough analysis of the different ways that equivalences can be defined is given in [Uni13, Chapter 4]. They call the above definition *half-adjoint equivalence* [Uni13, Chapter 4.2]. There are other possible definitions of equivalences that are well-behaved as well (in the sense that those definitions lead to propositional types). Having multiple such notions between which we can switch provides technical advantages, but we refer to the mentioned reference for a deeper going introduction.

2.2.3 Truncation Levels

If a type A has a certain truncation level n, or is an n-type, or is n-truncated (all notions used synonymously), that means its higher homotopical structure (above the given level n) is trivial. We start with the lowest case: A is *contractible* if

there is a point $a_0 : A$ (sometimes called the *centre*) such that all points are equal to a,

$$\mathsf{isContr}(A) :\equiv \Sigma\,(a : A)\,.\,\Pi_{b:A}\,a = b. \tag{2.28}$$

For $n \geq -2$, the statement that A is an *n-type* is defined as follows. For $n \equiv -2$, we take $\mathsf{isContr}(A)$ as the definition. Otherwise, the meaning is that all path spaces over elements of A are of truncation level one lower,

$$\mathsf{is}\text{-}(-2)\text{-}\mathsf{type}(A) \quad :\equiv \mathsf{isContr}(A) \tag{2.29}$$

$$\mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(A) :\equiv \Pi_{x,y:A}\,\mathsf{is}\text{-}n\text{-}\mathsf{type}(x =_A y). \tag{2.30}$$

For $n \equiv -1$ and $n \equiv 0$, there are very common synonyms to "n-type" and "n-truncated". By a standard lemma, A is a (-1)-type if and only if all its inhabitants are equal, i.e. if $\Pi_{a,b:A}\,a = b$ is inhabited. Such a type is called a *proposition* and has the property of being *propositional*. This implies that a type is contractible if and only if it is both propositional and inhabited. Further, a 0-type is a type with unique identity proofs; those types are called *sets*. For $n \equiv -2, -1, 0$, instead of $\mathsf{is}\text{-}n\text{-}\mathsf{type}(A)$, we will therefore write $\mathsf{isContr}(A), \mathsf{isProp}(A), \mathsf{isSet}(A)$, respectively. As the hierarchy of truncation levels starts by convention with -2 (in order to match the notion in homotopy theory), it is convenient to introduce a type $\mathbb{N}_{-2}$ of numbers starting with -2.

There are some very basic standard lemmata that we want to list here. The proofs can be found in the referenced literature. The second was already mentioned in Section 1.1.

Ⓐ **Lemma 2.2.2** (Truncation levels are upwards closed [Uni13, Theorem 7.1.7])**.** *For any type A and numbers $n \geq m \geq -2$, we have*

$$\mathsf{is}\text{-}m\text{-}\mathsf{type}(A) \to \mathsf{is}\text{-}n\text{-}\mathsf{type}(A). \tag{2.31}$$

□

Further, Σ preserves truncation level. A similar statement about Π does however require function extensionality (Lemma 2.2.6).

Ⓐ **Lemma 2.2.3** (Σ preserves truncation level [Uni13, Theorem 7.1.8])**.** *For a type B that may depend on some type A, if A and all $B(a)$ with $a : A$ are n-types, then so is $\Sigma\,(a : A)\,.\,B(a)$. In particular, the product of two n-types is an n-type.* □

If we have some $n \geq -2$ and a universe $\mathcal{U}$, we can consider its "subuniverse" of n-types:

Ⓐ **Definition 2.2.4** ($\mathcal{U}^n$)**.** For $n \geq -2$ and a universe $\mathcal{U}$, we write

$$\mathcal{U}^n :\equiv \Sigma\,(X : \mathcal{U})\,.\,\mathsf{is}\text{-}n\text{-}\mathsf{type}(X) \tag{2.32}$$

for the type of n-types in $\mathcal{U}$.

Note that $\mathcal{U}^n$ is not a universe in the sense that it is a basic component of the theory, it is simply a defined type (living in any universe that $\mathcal{U}$ lives in). However, it is in an appropriate sense closed under Σ (Lemma 2.2.3) and Π (Lemma 2.2.6). In [Uni13, Chapter 7.1], $\mathcal{U}^n$ is written n–Type. We do not use this notation as it fails to refer to the universe $\mathcal{U}$, which especially in Chapter 7 will be important for us.

If we are given $K : \mathcal{U}^n$, we keep application of the first projection implicit whenever we want to refer to the underlying type. That is, we talk of the *type* K, ignoring the second component (the proof that the first component is n-truncated). Therefore, we write $\Sigma\,(x : K)\,.\,M(x)$ instead of $\Sigma\,(x : \mathsf{fst}(K))\,.\,M(x)$.

Definition 2.2.4, combined with the hierarchy of universes as introduced in Section 2.1.4, allows us to consider a *two-dimensional* hierarchy. For any $m, n \geq 0$, we have a universe $\mathcal{U}_n^m$: the index n refers to the *size*, while m refers to the *truncation level*. Lemma 2.2.2 tells us that the collection of universes $\mathcal{U}_n^m$ is cumulative with respect to m, and therefore cumulative with respect to both indices.

2.2.4 Function Extensionality

Given $A : \mathcal{U}, B : A \to \mathcal{U}$ and two dependent functions $f, g : \Pi_{a:A} B(a)$, it is reasonable to think of them as being equal if they take equal values everywhere. However, in plain MLTT, the implication

$$\big(\Pi_{a:A} f(a) =_{B(a)} g(a)\big) \to f =_{\Pi_A B} g \tag{2.33}$$

is not derivable. As is was considered feasible, it was often added as an axiom, which we call *Naive Function Extensionality*. The resulting loss of canonicity (see Section 2.3) was discussed and solved by Altenkirch using a setoid model of MLTT [Alt99], requiring a setting that guarantees uniqueness of identity proofs.

In this thesis, we generally *do* assume naive function extensionality:

Axiom 2.2.5. *Given dependent functions $f, g : \Pi_{a:A} B(a)$ as before, if f and g are pointwise equal, then f and g are equal; that is, we postulate an inhabitant (typically called* funext*) of* (2.33)*.*

Sometimes we may consider MLTT without function extensionality and note that it is not required for some statement, in particular if we want to prove that another assumption implies function extensionality.

As we are not only interested in whether a path space is inhabited or not, but in the structure of the path space, we could require that *all* equalities $f = g$ "come from" a pointwise equality. More precisely, we could ask for the function

$$\mathsf{happly} : (f =_{\Pi_A B} g) \to \Pi_{a:A} f(a) =_{B(a)} g(a), \tag{2.34}$$

defined using apd (or directly by path induction), to be an equivalence, resulting in

$$\big(f =_{\Pi_A B} g\big) \;\simeq\; \big(\Pi_{a:A} f(a) =_{B(a)} g(a)\big). \tag{2.35}$$

We call this principle *Strong Function Extensionality* (see [Uni13, Chapter 2.9]).

Yet another very related principle is that "a family of contractible types is contractible". The statement

$$\big(\Pi_{a:A}\mathsf{isContr}(B(a))\big) \to \mathsf{isContr}\big(\Pi_A B\big) \tag{2.36}$$

is called *Weak Function Extensionality* [Uni13, Definition 4.9.1].

It turns out that these three versions of function extensionality are pairwise logically equivalent, i.e. imply each other. The proof that weak implies strong function extensionality is given in [Uni13, Theorem 4.9.5], the other parts are simple. In HoTT, these principles are a consequence of the univalence axiom (see Section 2.3.1 below), as proved in [Uni13, Chapter 4.9], and thus in many cases do not need to be treated as axioms themselves.

An easy consequence of the formulation (2.36) is the following:

Ⓐ **Lemma 2.2.6** (Π preserves truncation level [Uni13, Theorem 7.1.9])**.** *For any types A and B (where B depends on A), if $B(a)$ is an n-type for every $a : A$, then $\Pi_A B$ is also an n-type.* □

We also have the following very useful property (relying on function extensionality):

Ⓐ **Lemma 2.2.7** (Level properties are propositional [Uni13, Theorem 7.1.10])**.** *For any type A and any $n \geq -2$, the type $\mathsf{is}\text{-}n\text{-}\mathsf{type}(A)$ is propositional.* □

2.2.5 Equivalence Reasoning

What we call *equivalence reasoning* is a trivial but powerful proving technique. The name is inspired by *equational reasoning* which describes an analogously working well-known principle, and which is a term that is often used in the Haskell community (I do not know where it has its origins). We will see later that, with the univalence axiom (Section 2.3.1), equivalence reasoning *is* essentially equational reasoning for types.

Assume we want to prove that two types A and B are equivalent. The most straightforward way is certainly to give functions $f : A \to B$ and $g : B \to A$ and prove that they are inverses of each other, and this also seems to be the strategy that is used most often by far. However, I believe that a (conceptually and in terms of readability) better style is to split the equivalence into small steps. In the same way as one proves an equality by "transforming" an expression step by step, we can prove $A \simeq B$ by finding a chain

$$A \simeq C_1 \simeq C_2 \simeq \ldots \simeq C_k \simeq B \tag{2.37}$$

in such a way that every step is easily seen to be a type equivalence, for example because it is an instance of a more general well-known equivalence.

I want to emphasise that this principle *is* completely trivial *but* it is surprising how far one gets with only a couple of "building blocks". It seems that most (or all?) equivalences that one would want to prove (or that have already been proved before in the "two functions"-style) can be derived like this in a natural way, and most of the time it leads to very clean, understandable proofs. In particular, it makes it unnecessary to rely on judgmental computation rules to simplify expressions. For example, it will be unnecessary to assume the judgmental computation rule of the propositional truncation. Equivalence reasoning is thus as simple as powerful and we will see it frequently in this thesis. Explicit examples of equivalence reasoning can be found in particular in Chapter 8: see Propositions 8.1.2 and 8.1.3, but also Theorem 8.8.5. However, it is only a byproduct that I demonstrate how powerful equivalence reasoning is, and I will (usually) not mention explicitly that I am using it. More theoretical considerations and a streamlined presentation of the strategy are potential future work. I have first learned from Paolo Capriotti and Christian Sattler how useful this proving principle is. Only then, I have recognised it as a principle that is worth paying attention to, and reformulated some of my earlier proofs. Formalised proofs in Agda benefit a lot from equivalence reasoning when it comes to readability, which we exploit heavily in the electronic appendix of this thesis.

Important "building blocks" (i.e. basic equivalences) are given in the lemmata below. Let us begin with the fact that constructions which include families of contractible types can often be simplified:

Ⓐ **Lemma 2.2.8** (Neutral contractible families)**.** *If B is a type depending on A and $B(a)$ is contractible for all a, then the equivalences*

(i) $\Sigma\,(a : A)\,.\,B(a) \simeq A$ *[Uni13, Lemma 3.11.9 (i)]*

(ii) $\Pi_{a:A} B(a) \simeq \mathbf{1}$

hold. □

An analogous statement holds if the base type (resp. the "exponent") is equivalent to $\mathbf{1}$:

Ⓐ **Lemma 2.2.9** (Neutral contractible base/exponent)**.** *Let A be a contractible type with center a_0, and let B be a family indexed over A. We then have the equivalences*

(i) $\Sigma\,(a : A)\,.\,B(a) \simeq B(a_0)$ *[Uni13, Lemma 3.11.9 (ii), Exercise 3.20]*

(ii) $\Pi_{a:A} B(a) \simeq B(a_0)$. □

A further standard lemma tells us that an equality between pairs corresponds to a pair of equalities. We record it here for later reference:

Ⓐ **Lemma 2.2.10** (Equality of pairs is pair of equalities [Uni13, Theorem 2.7.2])**.** *If* (x_1, y_1) *and* (x_2, y_2) *are both of type* $\Sigma\,(x : X)\,.\,Y(x)$*, then*

$$(x_1, y_1) = (x_2, y_2) \;\simeq\; \Sigma\,(u : x_1 = x_2)\,.\,u_*(y_1) = y_2. \tag{2.38}$$

In the case of a non-dependent product $X \times Y$*, this equivalence simplifies to*

$$(x_1, y_1) = (x_2, y_2) \;\simeq\; (x_1 = x_2) \times (y_1 = y_2). \tag{2.39}$$

□

Moreover, we have the following basic property:

Ⓐ **Lemma 2.2.11** (Equivalences preserve path spaces [Uni13, Theorem 2.11.1])**.** *If* $f : A \to B$ *is an equivalence and* $a_1, a_2 : A$ *are two points, then*

$$\big(a_1 =_A a_2\big) \;\simeq\; \big(f(a_1) =_B f(a_2)\big). \tag{2.40}$$

□

The following statement is often called the *[type theoretic] axiom of choice*, as it corresponds to the axiom of choice from set theory under the "propositions as types" view. However, it is by no means an axiom but a simple tautology in type theory. Moreover, it is not the correct formulation of choice when one has the homotopical interpretation in mind.

Ⓐ **Lemma 2.2.12** (Distributivity law for Σ and Π / type theoretic axiom of choice)**.** *If* A *is a type and* $B : A \to \mathcal{U}$ *as well as* $C : (\Sigma\,(a : A)\,.\,B(a)) \to \mathcal{U}$ *type families, then the equivalence*

$$(\Pi_{a:A}\Sigma\,(b : B(a))\,.\,C(a, b)) \;\simeq\; (\Sigma\,(g : \Pi_A B)\,.\,\Pi_{a:A} C(a, g(a))) \tag{2.41}$$

holds. More precisely, the canonical functions in both directions are equivalences: from left to right $f \mapsto (\mathsf{fst} \circ f, \mathsf{snd} \circ f)$*; and from right to left* $(f, g) \mapsto \lambda a.(f(a), g(a))$*.*

□

Ⓐ **Remark 2.2.13.** As a side note, the two different compositions of the canonical functions in Lemma 2.2.12 are even judgmentally equal to the two identity functions, if the theory supports the judgmental η-law for dependent pair types (in addition to the judgmental η-law for dependent function types). We do not assume this η-law, but Agda does, which allows us to check the claim explicitly in the electronic appendix.

One very important concept which needs to be recorded are *singletons*. If A is a type with a point $a_0 : A$, we say *singleton* for a type of the form

$$\Sigma\,(a : A)\,.\,a = a_0 \tag{2.42}$$

or

$$\Sigma\,(a : A)\,.\,a_0 = a. \tag{2.43}$$

The following lemma is a statement that we will make use of numerous times, usually in combination with one of the other lemmata above:

Ⓐ **Lemma 2.2.14** (contractible singletons [Uni13, Lemma 3.11.8])**.** *All types of the form* (2.42) *or* (2.43) *are contractible.* □

A caveat is required here. Our main reference uses the term *singleton* for any type that is contractible [Uni13, Definition 3.11.1], which differs slightly from our usage. When we talk of singletons, we explicitly mean types that are given by expressions of the form (2.42) or (2.43). As these type expressions turn up frequently, it is helpful to be able to refer to them directly. Lemma 2.2.14 shows that our terminology is not in direct conflict with the terminology of [Uni13], we simply use the term more sparely.

Another "basic equivalence" is the correspondence between type families and "fibrations". This however requires the univalence axiom, so we state it below as Lemma 2.3.1.

2.2.6 Pointed Types

If A is a type and $a : A$ one of its elements, then (A, a) is called a *pointed type* with *underlying type* A and *basepoint* a. Let us write $\mathcal{U}_\bullet$ for the type of pointed types with underlying type living in $\mathcal{U}$ [Uni13, Definition 2.1.7], that is,

$$\mathcal{U}_\bullet :\equiv \Sigma\,(A : \mathcal{U})\,.\,A. \tag{2.44}$$

We call $\mathcal{U}_\bullet$ the *universe of pointed types* as this matches the intuition. Note, however, that it is really just a defined type, rather than a primitive of the theory as the universes $\mathcal{U}_k$ are. If (A, a) and (B, b) are pointed types, a *pointed function* consists of a map $f : A \to B$ and a proof of $f(a) = b$, showing that the basepoint is preserved. If additionally f is an equivalence, we speak of a *pointed equivalence*. Further, we call a pointed type n-truncated (or an n-type, or say that it has truncation level n) if its underlying type has that property.

The following simple definition will be fairly useful later:

Ⓐ **Definition 2.2.15** (Pointed family, see [Uni13, Definition 5.8.1])**.** For a pointed type $\mathfrak{A} \equiv (A, a)$, a *pointed family* is a type family $P : A \to \mathcal{U}$ where the type over the basepoint is again pointed:

$$\mathrm{Fam}^\bullet_{\mathfrak{A}} :\equiv \Sigma\,(P : A \to \mathcal{U})\,.\,P(a). \tag{2.45}$$

Extending the notion of truncatedness from types to families, we say that the pointed family (P, p) is n-truncated if P is a family of n-types.

Remark 2.2.16. The definition of a pointed family is identical to that of a *pointed predicate* [Uni13, Definition 5.8.1]. However, we want the reader to think of actual families, and *predicates* are usually understood as "logical" (propositional) properties. Note that a pointed type can always be seen as a pointed family over the trivial pointed type $(\mathbf{1}, \star)$.

2.2.7 Loop Spaces

Let $(A, a) : \mathcal{U}_\bullet$ be a pointed type. Its *loop space* [Uni13, Chapter 2.1] the pointed type

$$\Omega(A, a) :\equiv ((a =_A a), \mathsf{refl}_a) : \mathcal{U}_\bullet, \tag{2.46}$$

the elements of which are called *loops*. As Ω is thus an endomorphism on $\mathcal{U}_\bullet$, it can be composed with itself. This gives us the *n-fold iterated loop space*

$$\Omega^0(A, a) \equiv (A, a) \tag{2.47}$$
$$\Omega^{n+1}(A, a) \equiv \Omega^n(\Omega(A, a)). \tag{2.48}$$

To gain intuition for $\Omega^{n+1}(A, a)$, we can unfold the definition. This shows immediately that the underlying type is $\mathsf{refl}_a^n = \mathsf{refl}_a^n$, while the point is the canonical inhabitant of the underlying type, namely refl_a^{n+1}.

It is well-known that, in order to say something about the higher homotopical structure of a type, it is enough to look at its loop spaces. We give a sketch of a proof that, we think, is more direct than the one in the reference we cite.

Ⓐ **Lemma 2.2.17** ([Uni13, Theorem 7.2.9])**.** *For every $n \geq -1$, a type A is an n-type if and only if $\Omega^{n+1}(A, a)$ is contractible for all $a : A$.* □

Proof sketch. The statement is clear for $n \equiv -1$. Assume $n \geq 0$. By definition, A is an n-type if and only if, for all $a, b : A$, the type $a = b$ is an $(n-1)$-type. By the induction hypothesis, this is (for all a, b) the case if and only if $\Omega^n(a = b, p)$ is contractible for all $p : a = b$. By path induction on p, this is equivalent to requiring $\Omega^n(a = a, \mathsf{refl}_a)$ to be contractible for all a in A. □

Even if not explicitly mentioned, it is straightforward to see that this can be stated with two indices in the following form:

Lemma 2.2.18. *Given $A : \mathcal{U}$ and $m > k \geq -1$. Then, A is an $(m + k)$-type if and only if $\Omega^m(A, a)$ is a k-type for all $a : A$.* □

From the second version, we can recover the original form by putting $m :\equiv n+1$ and $k :\equiv -1$. The apparent mismatch is made up for by the fact that any propositional pointed type is indeed contractible.

We will develop some further properties of $\mathcal{U}_\bullet$ and the interactions between Ω and certain type formers in Section 7.3.

2.2.8 Classical Principles

A principle that we do not assume to hold in general, but which is sometimes interesting to consider, is the *law of excluded middle*, formulated only for propositions.

Ⓐ **Definition 2.2.19** (Law of excluded middle, see [Uni13, Chapter 3.4])**.** For a universe $\mathcal{U}$, we say that the law of excluded middle holds if the type

$$\mathsf{LEM}_{-1} :\equiv \Pi_{P:\mathcal{U}} \mathsf{isProp}(P) \to (P + \neg P) \tag{2.49}$$

is inhabited. Further, we say that we have excluded middle with choice if

$$\mathsf{LEM}_{\infty} :\equiv \Pi_{A:\mathcal{U}} (A + \neg A) \tag{2.50}$$

is inhabited.

We do not assume either of them; quite the contrary, we sometimes use that, if from some assumption we can prove one of them, then that assumption cannot be derivable. Note that LEM_{∞} contradicts the univalence axiom (see Section 2.3.1 below), while LEM_{-1} does not, and we want to emphasise that LEM_{-1} is the "correct" assumption if we want to reason classically in HoTT.

2.3 Homotopy Type Theory

Compared to the fundamental formulation of MLTT, HoTT essentially only introduces two new principles (at least in the presentation we give): univalence and higher inductive types.

2.3.1 Univalence

Given types $A, B : \mathcal{U}$ as above, there is a canonical map

$$\mathsf{idtoeqv} : (A =_{\mathcal{U}} B) \to (A \simeq B), \tag{2.51}$$

defined by path induction. Voevodsky's *univalence axiom* says that this function is an equivalence itself. More precisely, we say that the universe $\mathcal{U}$ is *univalent* if, for any of its types A and B, the correspondingly defined function $\mathsf{idtoeqv}$ is an equivalence. In HoTT, we assume that all our universes $\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \ldots$ are univalent.

It is a well-known result by Voevodsky the univalence axiom implies function extensionality [Uni13, Chapter 4.9].

Adding the univalence axiom to the theory destroys *canonicity*: there are closed terms of type $\mathbb{N}$ that are not in normal form. It is conjectured that some form of canonicity relative to propositional equality can be recovered. While many formalisations of mathematical theorems have already been performed, many of them could have been easier if there had been an appropriate computation rule for the univalence axiom. As discussed in the introduction, one of the most important open problems in HoTT is currently to find proper computation rules for the univalence axiom. We do not further discuss this problem in this thesis, however, we touch it in Chapter 9.

If we combine the univalence axiom with Lemma 2.2.10 we see that, for pointed types X and Y, the type of pointed equivalences between them is equivalent to the type of equalities $X =_{\mathcal{U}_\bullet} Y$.

Univalence allows us to make the connection between type families and "fibrations" in type theory concrete. As *every* function is a "fibration", this becomes very simple:

Lemma 2.3.1 (type families and fibrations [Uni13, Theorem 4.8.3])**.** *For any type A of a univalent universe $\mathcal{U}$, we have the type equivalence*

$$A \to \mathcal{U} \;\simeq\; \Sigma\,(B : \mathcal{U})\,.\,(B \to A). \tag{2.52}$$

Proof. Given $f : A \to \mathcal{U}$, we get the type $\Sigma\,(a : A)\,.\,f(A)$, together with the first projection. From any type B with a function $g : B \to A$, we can get a map $A \to \mathcal{U}$ by mapping any point to the corresponding fibre of g, i.e. we get the function $\lambda a.\Sigma\,(b : B)\,.\,(b = f(a))$. It is standard to check that both maps are inverses to each other. In fact, each of the two ways of composing the maps is easily seen to be equal to the identity map if we use Lemma 2.2.14, as each way essentially adds a singleton. □

2.3.2 Higher Inductive Types

The other major new addition that HoTT makes are *Higher Inductive Types* (HITs). They are discussed extensively in [Uni13, Chapter 6]. In case of ordinary inductive types, we give constructors to construct points of that type. For HITs, we can also have constructors to generate a new path inside the corresponding type. We choose to present a simple but yet interesting example, the *cirlce* $\mathbb{S}^1$, generated by one constructor

$$\mathsf{base} : \mathbb{S}^1 \tag{2.53}$$

and one constructor

$$\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}. \tag{2.54}$$

Its recursion principle says: if we are given a type B with a point $b_0 : B$ and a loop around that point, i.e. $l : b_0 = b_0$, we get a function $f : \mathbb{S}^1 \to B$ satisfying $f(\mathsf{base}) \equiv b_0$ and $\mathsf{ap}_f(\mathsf{loop}) =_{b_0 = b_0} l$. Note that the first equality holds judgmentally, the second only propositionally. The induction principle is somewhat more involved. Suppose we have a type family $B : \mathbb{S}^1 \to \mathcal{U}$ (for example defined using the just presented recursion rule). We then definitely need a point $b_0 : B(\mathsf{base})$. However, a loop $b_0 =_{B(\mathsf{base})} b_0$ is not the right thing, as that would be a path in the fibre over b_0. Instead, we want a path over loop, that is we ask for

$$l : b_0 =^B_{\mathsf{loop}} b_0. \tag{2.55}$$

Then, we get a function $f : \Pi_{\mathbb{S}^1} B$ with $f(\mathsf{base}) \equiv b_0$ and $\mathsf{apd}_f(\mathsf{loop}) = l$.

2.3.3 Truncations

Truncations (or *truncation operators*), for which we have already given intuition in Section 1.2, can be viewed as a special sort of higher inductive types. This is how at least the higher truncations are introduced in the standard reference [Uni13, Chapter 7.3]. In our setting of intensional MLTT, we are especially interested in *propositional truncations*, or (-1)-*truncations* [Uni13, Chapter 3.7]. If (a version of) MLTT has propositional truncations, it means that for any type X, there is a propositional type $\|X\|$, intuitively 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.

When we talk about a general "type theory" in the following definition or in any of the later chapters, we always mean (a version of) intensional Martin-Löf type theory which has at least the components of Section 2.1.

Ⓐ **Definition 2.3.2.** We say that a type theory has *weak propositional truncations* [for a universe $\mathcal{U}$] if there is a function $\|-\| : \mathcal{U} \to \mathcal{U}$ such that, for every $X : \mathcal{U}$, the type $\|X\| : \mathcal{U}$ represents the proposition that X is inhabited. More precisely, we have[4]

1. a function (or "constructor") $|-| : X \to \|X\|$
2. a proof $\mathsf{h_{tr}} : \mathsf{isProp}(\|X\|)$
3. a "recursion principle" $\mathsf{rec_{tr}} : \forall(P : \mathcal{U}^{-1}).\, (X \to P) \to \|X\| \to P$.

Definition 2.3.2 axiomatizes the propositional truncation. It characterises $\|X\|$ in the sense of the following *universal property*, which is especially useful when we apply the equivalence reasoning style:

Lemma 2.3.3 (Universal property of the propositional truncation [Uni13, Lemma 7.3.3])**.** *If a theory has function extensionality and weak propositional truncations, then, for any type X and any proposition Y, the canonical map*

$$(\|X\| \to Y) \to (X \to Y), \tag{2.56}$$

defined by $g \mapsto g \circ |-|$, is an equivalence.

Proof. Function extensionality implies that the domain and codomain type of the function are both propositional. It is therefore sufficient that the recursion principle gives a function in the other direction. □

[4]Recall the convention that we keep the first projection implicit when we have $P : \mathcal{U}^n$, so that we may write $x : P$ instead of $x : \mathsf{fst}(P)$.

The axioms of Definition 2.3.2 imply that, in a sense that we do not make precise here, the operator $\|-\|$ is the reflector for the category of propositions, viewed as a subcategory of the category of types. Therefore, it could also be justified to call it the *propositional reflection*.

Adopting the terminology of [Uni13, Chapter 3.10], we say that X is *merely* inhabited if $\|X\|$ is inhabited. We may also say that (the statement) X *merely* holds. However, we try to always be precise by giving the type expression to support the statement that is given in "natural language".

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

The *non-dependent eliminator* (or *recursion principle*) $\mathsf{rec}_{\mathsf{tr}}$ implies the *dependent* one (the *induction principle*):

Ⓐ **Lemma 2.3.4** (see [Uni13, Exercise 3.17])**.** *The weak propositional truncation admits the following induction principle: Given a type X, a family $P : \|X\| \to \mathcal{U}$ with a proof $h : \forall(z : \|X\|).\, \mathsf{isProp}(P(z))$, a term $k : \forall(x : X).\, P(|x|)$ gives rise to an inhabitant of $\forall(z : \|X\|).\, P(z)$. We call this term (for an implicitly given type X)*

$$\mathsf{ind}_{\mathsf{tr}} : \Pi_{P:\|X\|\to\mathcal{U}^{-1}} \left(\Pi_{x:X} P(|x|)\right) \to \left(\Pi_{z:\|X\|} P(z)\right). \tag{2.57}$$

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

Ⓐ **Remark 2.3.5.** In the standard reference [Uni13], the propositional truncation is introduced with the judgmental computation rule

$$\mathsf{rec}_{\mathsf{tr}}(P, h, f, |x|) \equiv_{\beta} f(x). \tag{2.58}$$

This fits into the general pattern that β-rules of inductive types hold judgmentally. In this case, one might also require the theory to have the induction principle $\mathsf{ind}_{\mathsf{tr}}$ with a judgmental computation rule,

$$\mathsf{ind}_{\mathsf{tr}}(P, h, f, |x|) \equiv_{\beta} f(x). \tag{2.59}$$

Note that we did *not* require this rule to hold in Definition 2.3.2, and that is why we added the attribute "weak". The equivalence reasoning style (Section 2.2.5) makes it unnecessary to depend on computational rules, so we simply do not need it for our proofs (not even for convenience). However, in Chapter 6, we will discuss a couple of consequences of the computation rule which do definitely not hold without it. A practical advantage of not assuming (2.58) is that the truncation can be implemented in existing proof assistants more easily. Of course, this "computation" (or β-) rule holds propositionally as both sides of the equation inhabit the same proposition.

Recall that there is a type expression that is equivalent to propositional truncation, its *impredicative encoding*:

Ⓐ **Proposition 2.3.6.** *For any given $X : \mathcal{U}$, we have*

$$\|X\| \leftrightarrow \forall(P : \mathcal{U}).\, \mathsf{isProp}P \to (X \to P) \to P. \tag{2.60}$$

Due to function extensionality, the logical equivalence is even an equivalence.

A potential problem with the expression on the right-hand side is that it is not living in 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 2.3.2 are satisfied (including even the computation rule (2.58)), at least under the assumption of function extensionality. Voevodsky (see the Coq library [Voe10b; Voe13b]) uses *resizing rules* to get rid of the problem.

Proof. The direction "→" of the statement is not more than a rearrangement of the assumptions of property (3). For the other direction, we only need to instantiate P with $\|X\|$ and observe that the properties (1) and (2) in the definition of $\|X\|$ are exactly what is needed. For the last claim of the statement, we observe that the right-hand side is propositional under function extensionality. □

Making use of the propositional truncation, we can formulate the axiom of choice properly. Compared with Lemma 2.2.12, the following definition involve no choice.

Definition 2.3.7 (Axiom of choice [Uni13, Chapter 3.8])**.** Let a set X, a family of sets $A : X \to \mathcal{U}^0$, and a family of propositions $P : (\Sigma\,(x : X)\,.\,A(x)) \to \mathcal{U}^{-1}$ be given. The *axiom of choice* is (for any such X, A, P) an inhabitant of the type

$$\mathsf{AC}_{-1} :\equiv (\Pi_{x:X} \|\Sigma\,(a : A(x))\,.\,P(a,x)\|) \to \|\Sigma\,(g : \Pi_{x:X} A(x))\,.\,\Pi_{x:X} P(x, g(x))\|. \tag{2.61}$$

A useful equivalent formulation ([Uni13, Lemma 3.8.2]) is the following: The axiom of choice holds if and only if, for any set X and family of sets $Y : X \to \mathcal{U}^0$, the type

$$(\Pi_{x:X} \|Y(x)\|) \to \|\Pi_{x:X} Y(x)\| \tag{2.62}$$

is inhabited.

Note that we do *not* assume the axiom of choice. It is simply a type that may be inhabited, but does not need to be, and is only called an "axiom" to make the correspondence to the set theoretic principle clear.

Although the propositional (or (-1)-) truncation is certainly the most important incarnation, homotopy type theory allows us to truncate a type at any level $n \geq -1$.[5] The n-truncation can be introduced in the following way.

[5] Technically, there is no reason to disregard the case (-2), although the (-2)-truncation of any type will be contractible. One could argue that $\|-\|_{-2}$ might still be interesting because of its computational behaviour. However, if I am not mistaken, the computational behaviour of $\|A\|_{-2}$ that one would expect would not be very different from that of $\|1 + A\|_{-1}$ anyway.

Definition 2.3.8 (General truncation, [Uni13, Chapter 7.3])**.** We say that a type theory has *weak general truncations* [for a universe $\mathcal{U}$] if, for any number $n \geq -1$ and for any type $X : \mathcal{U}$, we have a type $\|X\|_n : \mathcal{U}$ together with

1. a function ("constructor") $|-|_n : X \to \|X\|_n$, where $|-|_n(a)$ is written $\|a\|_n$;
2. a proof $t : \mathsf{is}\text{-}n\text{-}\mathsf{type}(\|X\|_n)$;
3. an induction principle
$$\mathsf{ind}_{\mathsf{tr}} : \Pi_{P:\|X\|_n \to \mathcal{U}^n} \left(\Pi_{x:X} P(|x|_n)\right) \to \left(\Pi_{z:\|X\|_n} P(z)\right). \tag{2.63}$$
4. which satisfies
$$\Pi_{P:\|X\|_n \to \mathcal{U}^n} \Pi_{f:\Pi_{x:X} P(|x|_n)} \mathsf{ind}_{\mathsf{tr}}(P, f, |n|_n) =_{P(|x|_n)} f(x). \tag{2.64}$$

Remark 2.3.9. Of course, we do not want to distinguish between $\|-\|$ and $\|-\|_{-1}$. We will use the first whenever propositional truncation is the only truncation that we consider, and the second otherwise.

As before, one can (but by default, we do not) require item 4 to hold judgmentally. This is done in the standard reference [Uni13, Chapter 7.3]. In other words, $\mathsf{ind}_{\mathsf{tr}}(n, A, P)$ is "judgmentally a section" of the canonical map
$$(\Pi_{x:|n|_A a} P(x)) \to (\Pi_{a:A} P(|n|_A a)). \tag{2.65}$$
As before, our arguments in this thesis would not benefit from the judgmental computation rule as we can avoid all "computational overhead" by using the equivalence reasoning style. Instead, we make crucial use of the following property, for which we omit the proof. It is analogous to (and generalises) Lemma 2.3.3.

Lemma 2.3.10 (Universal property of the general truncation, [Uni13, Lemma 7.3.3])**.** *If a theory has function extensionality and weak general truncations, then, for all n, A and any n-type B, the canonical map*
$$(\|A\|_n \to B) \to (A \to B) \tag{2.66}$$
is an equivalence. □

Remark 2.3.11. For inductive types,

- the induction principle
- the recursion principle plus a uniqueness (i.e. η) rule
- the universal property

can be derived from each other. This has been stated and proved by Awodey, Gambino, and Sojakova [AGS12] for *homotopical W-types*, that is "ordinary" inductive types (without judgmental computation rules) in homotopy type theory. Sojakova [Soj15] has proved the same statement for specific classes of higher inductive types. The logical equivalence of the three points holds (if carefully formulated) in particular for truncations.

2.4 A Word on Ambiguity Avoidance and Readability

In homotopy type theory, certain "key words" are reserved. We try to be very careful with the denomination of statements. The usage of the terms *definition*, *remark*, *example* and *corollary* is self-explanatory. We state a result as a *lemma* if it is of technical nature and useful for the proceeding constructions. We follow the standard convention and call a result a *proposition* if it is neither a main result nor has the "auxiliary" character of a lemma. However, a caveat is required: this does not necessarily mean that the corresponding statement is a propositional type in the sense of HoTT. The terms *principle* and *claim* will not be used very often, and if they are used, their meaning will be clear.

In contrast to other authors, we strive to minimise the usage of the term *theorem* and we only use it for results that we either consider main contributions of this thesis, or (even though they might be very easy to prove) core insights that potentially have powerful applications. If such a statement is of fairly technical nature, we call it a *main lemma*; however, to immediately revoke the last sentence, we admit that the difference is fairly non-objective, as well as the pairwise difference between propositions, lemmata, theorems, and main lemmata.

One might argue that the word *property* should also be reserved in the same way as *proposition*. However, we do not consider this term critical, and we will say *property* to attributes that might not be propositional. If we want to emphasise that they are, we call them *propositional properties*.

Being an *equivalence* is a precisely defined term that can only refer to a function $f : X \to Y$, and begin *equivalent* can only refer to two types (or more than two, in which case it has to be read as "pairwise equivalent"). Often in mathematics, a statement will start with the sentence *"The following are equivalent"*, a very useful expression that is tempting to use. We do *not* use this expression if it interferes with the HoTT meaning of *equivalence*. Sometimes, the statements (or types) of which we want to prove that they pairwise imply each other are propositional, and in that case, we do not hesitate to use the above expression as it is fully justified. If the statements are not all propositional, but still equivalent as types, the usage of that expression would cause no conflict, but in that case we believe that it is worth to emphasise that fact. However, more often than not, the statements in question are *not* equivalent as types, but they simply imply each other. In this case, what we write is *"The following are logically equivalent"*.

In the terminology of the textbook [Uni13], the word *merely* refers to the propositional truncation, and we use it in this sense (see the explanations in Section 2.3.3).

One further potentially dangerous issue regarding notation is that we have to deal with many equality-like concepts: the internal propositional equality of type theory, internal equivalence of types, judgmental equality of type theoretic expressions, as well as isomorphism of objects in a category that we sometimes

consider in the meta-theory, isomorphism or equivalence of categories, and strict equality of morphisms. Our convention is that *internal* concepts are written using "two-line" symbols, coinciding with the notation of [Uni13]: we write $a = b$ for the internal equality type $\mathsf{Id}(a, b)$, and $A \simeq B$ for the type of equivalences between A and B. Non-internal concepts are denoted (if at all) using "three-line" symbols: we write $a \equiv b$ if a and b denote two judgmentally equal expressions, and we use $\equiv$ for other cases of *strict* equality in the meta-theory. $x \cong y$ means we want to express that x and y are isomorphic objects of a category (in the meta-theory). Equality of morphisms of a category is sometimes expressed with $\equiv$, but usually by saying that some diagram commutes, and if we say that some diagram commutes, we always mean that it commutes *strictly*, not only up to homotopy. Other notions of equality are written out.

Chapter 3

Truncation Level Criteria

An important property of a type that one would often like to know is whether it is of some truncation level. For example, it is crucial to know the higher homotopical structure of a type X if we want to construct a function from a certain higher inductive type H into X, especially if H is a truncation itself. Moreover, if X is n-truncated, we immediately get that all higher loop spaces and homotopy groups (starting from the ones at level $(n+1)$) are trivial. On the other hand, even though there is no clear distinction between the two topics, truncation levels are also important for a programmer who does not care about the formalisation of mathematics. For example, for a *set* (a 0-truncated type), Streicher's K eliminator is justified, making it (essentially) possible to replace all equality proofs by reflexivity. In an appropriately designed programming language, this then allows a more powerful form of pattern matching.[1] From a more mathematical perspective in the context of HoTT, sets have been studied in depth by Rijke and Spitters [RS14].

Section 3.1 only deals with the criteria for whether a type is a set. In the beginning, we only consider basic MLTT, and add weak propositional truncations after a short motivation. In the same way, we mention explicitly if function extensionality is needed to prove a result. We have published most of its contents together with Escardó, Coquand and Altenkirch ([KECA13], [KECA14]). Those results are also contained in the HoTT standard reference [Uni13, Chapter 7.2]. The (straightforward but useful) generalisations to higher truncations, as presented in Section 3.2, have not been published before.

3.1 Hedberg's Theorem Revisited

A classic such criterion for the truncation level of a type was given by Hedberg:

[1] In Agda, such a powerful form of pattern matching is possible for all types by default, which contradicts the Univalence Axiom. The *without-K* flag can be used to prevent this.

Ⓐ **Proposition 3.1.1** (Hedberg [Hed98])**.** *Every type with decidable equality is a set,*

$$\mathsf{discrete}X \to \mathsf{isSet}X. \tag{3.1}$$

Here, a type X is said to have *decidable equality*, written $\mathsf{discrete}X$ (which we have referred to in the introduction, page 4), if the equality of any two points is decidable,

$$\mathsf{discrete}X :\equiv \forall(x_1\, x_2 : X).\, (x_1 = x_2) + \neg(x_1 = x_2). \tag{3.2}$$

Before we give the proof, we introduce several useful notions.

Ⓐ **Definition 3.1.2.** Given two types X, Y and a function $f : X \to Y$, we say that f is *weakly constant* if it maps any inputs to equal values,

$$\mathsf{const}_f :\equiv \forall(x_1\, x_2 : X).\, f(x_1) = f(x_2). \tag{3.3}$$

We further say that a type with a weakly constant endofunction is *collapsilbe*,

$$\mathsf{coll}X :\equiv \Sigma\,(f : X \to X)\,.\,\mathsf{const}_f, \tag{3.4}$$

and it is *path-collapsible* if all its path spaces are collapsible,

$$\mathsf{pathColl}X :\equiv \forall(x_1\, x_2 : X).\, \mathsf{coll}(x_1 = x_2). \tag{3.5}$$

For convenience, we drop the attribute "weakly" when talking about weakly constant function, and call a function just *constant* if it satisfies (3.3). We will see later (especially in Chapters 5 and 8) why this notion of constancy is not entirely satisfying.

Hedberg's original proof of Proposition 3.1.1 consists of two steps which we present in the following two lemmata.

Ⓐ **Lemma 3.1.3.** *If a type has decidable equality, it is path-collapsible,*

$$\mathsf{discrete}X \to \mathsf{pathColl}X. \tag{3.6}$$

Proof. Given inhabitants x_1 and x_2 of X, we get by assumption either an inhabitant of $x_1 = x_2$ or an inhabitant of $\neg(x_1 = x_2)$. In the first case, we construct the required constant function $(x_1 = x_2) \to (x_1 = x_2)$ by mapping everything to this given path. In the second case, the identity function is trivially constant. □

Ⓐ **Lemma 3.1.4.** *If a type is path-collapsible, it is a set,*

$$\mathsf{pathColl}X \to \mathsf{isSet}X. \tag{3.7}$$

Proof. Assume f is a parametrised constant endofunction on the path spaces, meaning that, for any $x_1, x_2 : X$, we have a constant function

$$f_{x_1,x_2} : (x_1 = x_2) \to (x_1 = x_2). \tag{3.8}$$

Let p be a path from x_1 to x_2. We claim that

$$p = (f_{x_1,x_1}(\mathsf{refl}_{x_1}))^{-1} \cdot f_{x_1,x_2}(p). \tag{3.9}$$

By path induction, we only have to give a proof if the triple (x_1, x_1, p) is in fact $(x_1, x_1, \mathsf{refl}_{x_1})$, which is one of the groupoid laws that propositional equality satisfies. Using the fact f_{x_1,x_2} is constant, the right-hand side of the above equality is independent of p, and in particular, p is equal to any other path of the same type. □

Hedberg's proof is the concatenation of the two lemmata. We want to present a slightly more direct (but essentially equivalent) proof [Hcoq], [Kra12]:

Second proof of Proposition 3.1.1. Assume that, for any $x_1, x_2 : X$, we have

$$\mathsf{dec}_{x_1,x_2} : (x_1 = x_2) + \neg(x_1 = x_2). \tag{3.10}$$

Given any x_1, x_2 and $p : x_1 = x_2$, we know that $\mathsf{decidable} x_1 x_2$ is of the form $\mathsf{inl}\, q$. Further, we know that $\mathsf{decidable} x_1 x_1$ is of the form $\mathsf{inl}\, r$. We claim

$$p = r^{-1} \cdot q, \tag{3.11}$$

which is trivial by path induction because we have $q \equiv r$ in case of $x_1 \equiv x_2$. Again, the argument is that the right-hand side of (3.11) is independent of p. □

Let us analyse the ingredients of the original proof. Lemma 3.1.3 uses the rather strong assumption of decidable equality. In contrast, the assumption of Lemma 3.1.4 is equivalent its conclusion, so that there is no space for a strengthening. We include a proof of this simple claim in Proposition 3.1.10 below and concentrate on weakening the assumption of Lemma 3.1.4. Let us first introduce the notions of *stability* and *separatedness*.

Ⓐ **Definition 3.1.5.** For any type X, define

$$\mathsf{stable} X :\equiv \neg\neg X \to X, \tag{3.12}$$
$$\mathsf{separated} X :\equiv \forall(x, y : X).\, \mathsf{stable}(x = y). \tag{3.13}$$

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

Ⓐ **Proposition 3.1.6** ([Uni13, Corollary 7.2.3])**.** *Any separated type is a set if function extensionality holds,*

$$\mathsf{separated} X \to \mathsf{isSet} X. \tag{3.14}$$

Proof. There is, for any $x, y : X$, a canonical map $(x = y) \to \neg\neg(x = y)$. 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 3.1.4. □

We remark that full function extensionality is actually not needed here. Instead, a weaker version that only works with the empty type is sufficient.

The core of Proposition 3.1.6 is that the condition $\mathsf{separated}X$ allows us to conclude $x = y$ from a proposition, where function extensionality is exactly used to make sure that $\neg\neg(x = y)$ is propositional. In fact, *any* proposition with the appropriate property allows us to replicate the argument. The statement in the following form is sometimes attributed to Egbert Rijke:

Ⓐ **Lemma 3.1.7** ([Uni13, Theorem 7.2.2])**.** *Let R be a reflexive propositional relation on a type X, that is*

$$R : X \times X \to \mathcal{U} \tag{3.15}$$
$$\forall x_1\, x_2.\, \mathsf{isProp}\,(R(x_1, x_2)) \tag{3.16}$$
$$\forall x.\, R(x, x). \tag{3.17}$$

Suppose further that R implies identity in the sense of

$$\forall x_1\, x_2.\, R(x_1, x_2) \to x_1 = x_2. \tag{3.18}$$

Then X is a set and $R(x_1, x_2)$ is equivalent to $x_1 = x_2$ for all $x_1, x_2 : X$.

Proof. The argument is essentially the same as for Proposition 3.1.6. For any two points, we get a map

$$(x_1 = x_2) \to R(x_1, x_2), \tag{3.19}$$

using path induction with the fact that R is reflexive. Composition with the map (3.18) yields a witness that X is path-collapsible. □

Remark 3.1.8. Our original publication [KECA13] included Lemma 3.1.7 only for the fixed choice of $R(x_1, x_2) :\equiv \|x_1 = x_2\|$.

Comparing Proposition 3.1.6 and Lemma 3.1.7, we might say that

$$R(x_1, x_2) :\equiv \neg\neg(x_1 = x_2) \tag{3.20}$$

is an attempt to find a reflexive and (assuming function extensionality) propositional relation that is in some sense universal. However, it is not completely satisfactory: it is in general not the case that $\neg\neg(x_1 = x_2)$ allows us to conclude $R(x_1, x_2)$ for any other reflexive propositional relation R. The right choice requires the theory to have weak propositional truncation. In that case, we set $R(x_1, x_2) :\equiv \|x_1 = x_2\|$. Whenever we have another reflexive propositional relation

Q, it is easy to see that $\|x_1 = x_2\|$ will imply $Q(x_1, x_2)$: using the recursion principle, we may assume $x_1 = x_2$, and by path induction, all that is needed to show is $Q(x_1, x_1)$, which is given by the fact that Q is reflexive.

Completely analogously to the notions of stability and separatedness, we can therefore define what it means for a type to have *split support* or to be *h-separated*:

Ⓐ **Definition 3.1.9.** For any type X, define

$$\mathsf{splitSup}X :\equiv \|X\| \to X, \tag{3.21}$$

$$\mathsf{hSeparated}X :\equiv \forall(x, y : X).\, \mathsf{splitSup}(x = y). \tag{3.22}$$

We observe that $\mathsf{hSeparated}X$ is a strictly weaker condition than $\mathsf{separated}X$. Not only can we conclude $\mathsf{isSet}X$ from $\mathsf{hSeparated}X$, but the converse holds as well. We summarize the discussion up to now in the following statement.

Ⓐ **Proposition 3.1.10.** *For a type X in MLTT, the following properties are equivalent:*

1. *X is a set*
2. *X is path-collapsible*
3. *X has a reflexive propositional relation that implies identity*
4. *X is h-separated (obviously, this point requires the theory to support propositional truncation).*

Proof. "$(2) \Rightarrow (1)$" is Lemma 3.1.4. For "$(1) \Rightarrow (3)$", we may define the required relation as $R(x_1, x_2) :\equiv (x_1 = x_2)$ which is propositional by the assumption. The argument of "$(3) \Rightarrow (2)$" was implicitly used in the proof of Lemma 3.1.7: assume that the relation R is given and construct an endomap on $(x_1 = x_2)$ which factors over $R(x_1, x_2)$. This map is constant because R is propositional.

Finally, the equivalence of (3) and (4) follows directly from the fact that $R(x_1, x_2) :\equiv \|x_1 = x_2\|$ has the universality property discussed before. □

We observe that using propositional truncation in some cases makes it unnecessary to appeal to functional extensionality: in Proposition 3.1.6, we have given a proof for the simple statement that separated types are sets in the context of function extensionality. This is not provable in plain MLTT. Let us now drop function extensionality and assume instead that propositional truncation is available. Every separated type is h-separated - more generally, we have

$$(\neg\neg X \to X) \to (\|X\| \to X) \tag{3.23}$$

for any type X -, and every h-separated space is a set. Notice that the mere availability of propositional truncation suffices to solve a gap that function extensionality would usually fill.

We want to mention that there is a slightly stronger version of the Hedberg's Theorem which applies to types where equality might only be decidable *locally*. In fact, nearly everything we stated or proved so far can be done locally, and thus made stronger. In the proof of Lemma 3.1.3, 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 3.1.4, we observe:

Ⓐ **Lemma 3.1.11** (Local form of Lemma 3.1.4)**.** *A locally path-collapsible type locally satisfies UIP. This means, for a pointed type* (X, x_0)*, we have*

$$(\forall x.\, \mathsf{coll}(x_0 = x)) \to \forall x.\, \mathsf{isProp}(x_0 = x). \tag{3.24}$$

Proof. The proof is identical to the one of Lemma 3.1.4, with the only difference that we need to apply based path induction instead of path induction. □

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

Ⓐ **Proposition 3.1.12** ([Pal12],[Kra12]; Local form of Proposition 3.1.1)**.** *A locally discrete type is locally a set: for any pointed type* (X, x_0)*,*

$$(\forall x.\, \mathsf{decidable}(x_0 = x)) \to \forall x.\, \mathsf{isProp}(x_0 = x). \tag{3.25}$$

□

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

Ⓐ **Proposition 3.1.13** (Local form of Proposition 3.1.6)**.** *Under the assumption of function extensionality, a locally separated type locally is a set: given a pointed type* (X, x_0)*,*

$$(\forall x.\, \mathsf{stable}(x_0 = x)) \to \forall x.\, \mathsf{isProp}(x_0 = x). \tag{3.26}$$

□

Similarly, the local forms of the characterizations of Proposition 3.1.10 are still equivalent. The following statement is not included in the electronic appendix; however, it is essentially a special case of Theorem 3.2.1 in the next subsection which *is* formalised and can be found in the appendix.

Proposition 3.1.14 (Local form of Proposition 3.1.10)**.** *For a pointed type* (X, x_0) *in MLTT, the following are equivalent:*

1. *for all* $x : X$*, the type* $x_0 = x$ *is propositional*
2. *for all* $x : X$*, the type* $x_0 = x$ *is collapsible*
3. *there is family of propositions* $Q : X \to \mathcal{U}^{-1}$ *which implies identity, that is*

$$\forall x.\, Q(x) \to x_0 = x \tag{3.27}$$

together with a point $q_0 : Q(x_0)$*.*

4. *for all $x : X$, the type $x_0 = x$ has split support (requiring that the theory supports propositional truncation).* □

3.2 Generalisations to Higher Levels

One way of generalising Proposition 3.1.14 is to formulate the criteria in a way that allows them to be applied if we want to show that a type has truncation level n, for some $n > -2$, rather than just truncation level 0. We call the following statement the *Generalised Local Hedberg Argument* (GLHA):

Ⓐ **Theorem 3.2.1.** *In MLTT, let X be a type, $x_0 : X$ a point, and $n \geq -1$ an integer. The following are logical equivalent (note that in general only the first two points are propositional types):*

1. *$\Omega^{n+1}(X, x_0)$ is contractible*
2. *there is a pointed family of $(n-1)$-types that implies local identity, that is a family $Q : X \to \mathcal{U}^{n-1}$ with $q_0 : Q(x_0)$ such that*

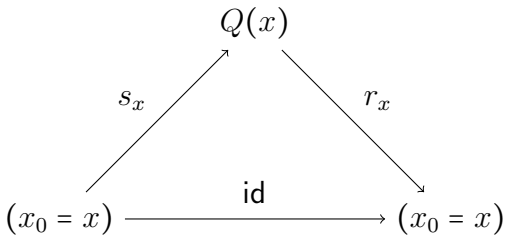
$$f : \forall x.\, Q(x) \to x_0 = x \tag{3.28}$$

3. *$\forall (x : X).\, \mathsf{is}\text{-}(n-1)\text{-}\mathsf{type}(x_0 = x)$*
4. *assuming that the theory has (weak) general truncations, we have*

$$\forall x.\, \|x_0 = x\|_{n-1} \to x_0 = x. \tag{3.29}$$

Further, X is n-truncated if and only if these statements hold for all $x_0 : X$.

Proof. We first prove the logical equivalence of the first three points, without referring to truncations. For $(1) \Rightarrow (2)$, observe that the condition $\mathsf{isContr}(\Omega^{n+1}(X, x_0))$ implies that $x_0 = x$ is $(n-1)$-truncated. To show $(2) \Rightarrow (3)$, let $x : X$ be any point. Consider the triangle



where the function $s_x : x_0 = x \to Q(x)$ is defined by path induction, $s_{x_0} :\equiv q_0$, and $r_x : Q(x) \to x_0 = x$ is defined as $r_x(q) :\equiv f(q_0)^{-1} \cdot f(q)$. By path induction, the diagram commutes, making $x_0 = x$ a retract of the $(n-1)$-type $Q(x)$. By [Uni13,

Theorem 7.1.4] (a retract of an m-type is an m-type), $x_0 = x$ is then $(n-1)$-truncated as well. The implication $(3) \Rightarrow (1)$ is easy and standard. It follows by induction on n.

Concerning (4), we can see $(1) \Rightarrow (4)$ in the same way as we saw $(1) \Rightarrow (2)$, and $(4) \Rightarrow (2)$ is trivial as we only need to choose $Q(x) :\equiv \|x_0 = x\|_{n-1}$.

Moreover, (1) for all $x_0 : X$ is well-known (Lemma 2.2.17) to be equivalent to the statement that X is n-truncated. □

Remark 3.2.2. The proof of Theorem 3.2.1 is entirely trivial. However, I think the statement itself is still very valuable and worth keeping in mind as a technical tool. I have presented it as such a tool at the first *HoTT-Day* in Leeds (February 2014). One concrete case where I found it useful is when one needs to prove the truncation level of a higher inductive type, which can turn out to be somewhat involved if one tries to apply the definition directly. Such an application will be presented in the "HIT proof" of Theorem 8.10.2.

Steve Awodey has pointed out to me that Theorem 3.2.1 can be seen as a solution to an open problem that was discussed during the special year program in Princeton (2012/13), the question of how Hedberg's argument can be generalised to higher levels. This open problem does not seem to be documented.

Chapter 4

Anonymous Existence

We have already discussed shortly that $\|X\|$ is a way of saying that X is inhabited without giving an explicit element. Instead, an element of $\|X\|$ can be understood as an *anonymous* inhabitant of X. Similarly, we can read a proof of $\neg\neg X$ as a form of *weak existence*. Both $\|X\|$ and $\neg\neg X$ have the property of being propositional (under function extensionality). In this Chapter, we will define a new notion of anonymous existence, namely $\langle\langle X \rangle\rangle$ which says that every constant endofunction on X has a fixed point, and we will see that this is also a propositional property of X. We will prove that it lies in between of $\neg\neg X$ and $\|X\|$. In particular, $\langle\langle X \rangle\rangle$ is strictly stronger than $\neg\neg X$ and, at the same time, a *definable* property (while $\|X\|$ is a notion that we need to add to the theory unless we want to use impredicativity).

The theory that we work in is basic MLTT, with the additional principles of function extensionality, propositional truncation and (in the last part) univalence explicitly mentioned whenever they are used.

We start by generalising the results of Section 3.1 in a different direction than we did in Section 3.2, which will inspire the notion of populatedness.

The main results of this chapter have been published before in [KECA14] and partially in [KECA13].

4.1 Collapsible Types have Split Support

If we unfold the definitions in the statements of Proposition 3.1.10, they all involve the path spaces over some type X. Recall from Definition 3.1.2 that we call a function $f : X \to Y$ *constant* (dropping the attribute *weakly* for convenience) if

$$\mathsf{const}_f : \forall (x_1\, x_2 : X).\, f(x_1) = f(x_2). \tag{4.1}$$

Recall from the same definition that we say that a type X is *collapsible* if it has a constant endomap,

$$\mathsf{coll}(X) :\equiv \Sigma\, (f : X \to X)\, .\, \mathsf{const}_f. \tag{4.2}$$

The logically equivalent statements on X as given in Proposition 3.1.10 are:

1. $\forall(x_1\, x_2 : X).\, \mathsf{isProp}(x_1 = x_2)$
2. $\forall(x_1\, x_2 : X).\, \mathsf{coll}(x_1 = x_2)$
3. there is
$$R : X \times X \to \mathcal{U}^{-1} \tag{4.3}$$
such that
$$\forall(x : X).\, R(x, x) \tag{4.4}$$
and
$$\forall(x_1\, x_2 : X).\, R(x_1, x_2) \to (x_1 = x_2) \tag{4.5}$$
4. $\forall(x_1\, x_2 : X).\, \|x_1 = x_2\| \to (x_1 = x_2)$.

As we have shown, these statements are logically equivalent. We think it is a natural question to ask whether the properties of path spaces are required. The possibilities that path spaces offer are very powerful and we have used them heavily. Let us try to formulate the above properties for an arbitrary type Y instead of path types. In (1), (2) and (4), we just replace $(x_1 = x_2)$ by Y to see what happens. In case of (3), this is not possible and we need to be slightly more careful: it can be read as saying that $R(x_1, x_2)$ is logically equivalent to $(x_1 = x_2)$, and we get:

1. $\mathsf{isProp}(Y)$
2. $\mathsf{coll}(Y)$
3. there is $R : \mathcal{U}^{-1}$ such that $Y \longleftrightarrow R$
4. $\|Y\| \to Y$.

Here, we notice immediately that (1) is significantly stronger than the other three properties. It says that Y is propositional, which trivially implies the other three statements. In a theory with propositional truncation, the logical equivalence of (3) and (4) is obvious, and they clearly imply collapsibility of Y as the composition of the maps $Y \to R$ and $R \to Y$ is a constant endofunction on Y (where R is taken to be $\|Y\|$ in (4).

However, the strength of (2) is less clear. Is a constant endofunction on Y sufficient to get from $\|Y\|$ to Y? More generally, for types Y and Z, is a constant function from Y to Z enough to derive $\|Y\| \to Z$? Somewhat surprisingly, the answer to the first question is positive, while the answer to the second one is very likely to be negative in the light of Chapter 8. In particular, Theorem 8.8.5 states that a constant function does factor through the propositional truncation if the constancy proof satisfies an infinite tower of coherence conditions. We will also discuss the second question in Section 5.1, while the positive answer for the first question is a consequence of the following crucial fixed point lemma.

Ⓐ **Main Lemma 4.1.1** (Fixed Point Lemma)**.** *Given, for some type X, an endofunction $f : X \to X$, we define the type of its fixed points as*

$$\mathsf{fix} f :\equiv \Sigma\,(x : X)\,.\,x = f(x). \tag{4.6}$$

If f is constant, then $\mathsf{fix} f$ is propositional.

Before we can give the proof, we need to formulate the following observation that we consider a key insight for the Fixed Point Lemma:

Ⓐ **Lemma 4.1.2.** *Let X and Y be two types. If $f : X \to Y$ is constant and $x_1, x_2 : X$ are points, then $\mathsf{ap}_f : (x_1 =_X x_2) \to (f(x_1) =_Y f(x_2))$ is constant. In particular, ap_f maps every loop around x (that is, path from x to x) to $\mathsf{refl}_{f(x)}$.*

Proof. If c is the proof of const_f, then ap_f maps a path $p : x_1 = x_2$ to

$$c(x_1, x_1)^{-1} \cdot c(x_1, x_2). \tag{4.7}$$

This is easily seen to be correct for (x, x, refl_x), which is enough to apply path induction. As the expression is independent of p, the function ap_f is constant. The second part follows from the fact that ap_f maps refl_x to $\mathsf{refl}_{f(x)}$. □

With this lemma at hand, we give a proof of the Fixed Point Lemma:

Proof of Main Lemma 4.1.1. Assume $f : X \to X$ is a function and $c : \mathsf{const}_f$ is a proof that it is constant. For any two pairs (x, p) and $(y, q) : \mathsf{fix} f$, we need to construct a path connection them. Figure 4.1 illustrates the situation.

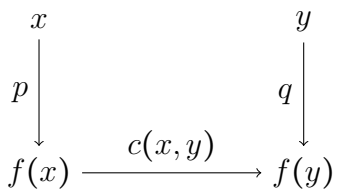


Figure 4.1: Two elements (x, p) and (y, q) of $\mathsf{fix} f$

First, we simplify the situation by showing that we can assume $x \equiv y$. The composition $p \cdot c(x, y) \cdot q^{-1}$ shows $x = y$. By Lemma 2.2.10, a path between pairs is a pair of paths. If we take the trivial paths as the second component, we get that (x, p) and

$$\big((y, \mathsf{transport}^{\lambda z \to z = f(z)}(p \cdot c(c, y) \cdot q^{-1}, p)\big) \tag{4.8}$$

are equal. Let us write r for the second component of (4.8); then, we need to prove $(y, r) = (y, q)$, which is the claimed simplification.

Again, such a path can be constructed from a pair of paths for the two components. Let us assume that we use some path $t : y = y$ for the first component. We then have to show that $t_*(r)$ equals q.

By Lemma 2.2.1, with the identity for h and f for k, the path $t_*(r)$ is equal to $t^{-1} \cdot r \cdot \mathsf{ap}_f t$. With Lemma 4.1.2, that term can be further simplified to $t^{-1} \cdot r$. What we have to prove is now just $t^{-1} \cdot r = q$, so let us just choose t to be $r \cdot q^{-1}$, thereby making it into a straightforward application of the standard lemmata. □

A more elegant but possibly less revealing proof of the Fixed Point Lemma was given by Christian Sattler. It uses the principle of *equivalence reasoning* that we have described in Section 2.2.5. The electronic appendix contains a formalisation of both the proof we gave above and Sattler's argument.

Second Proof of Main Lemma 4.1.1 (Sattler). Given $f : X \to X$ and $c : \mathsf{const}_f$ as before, assume $(x_0, p_0) : \mathsf{fix} f$. For any $x : X$, we have an equivalence of types,

$$f(x) = x \;\simeq\; f(x_0) = x, \tag{4.9}$$

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

$$\Sigma\,(x : X)\,.\, f(x) = x \;\simeq\; \Sigma\,(x : X)\,.\, f(x_0) = x. \tag{4.10}$$

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

We will exploit Main Lemma 4.1.1 in different ways. For the following corollary note that, given an endomap f on X with constancy proof c, we have a canonical projection

$$\mathsf{fst} : \mathsf{fix} f \to X \tag{4.11}$$

and a function

$$\epsilon : X \to \mathsf{fix} f \tag{4.12}$$

$$\epsilon(x) :\equiv (f(x)\,,\, c(x, f(x)))\,. \tag{4.13}$$

Ⓐ **Corollary 4.1.3.** *In basic MLTT, for a type X with a constant endofunction f, the type $\mathsf{fix} f$ is a proposition that is logically equivalent to X. In particular, $\mathsf{fix} f$ has all the properties that $\|X\|$ has, i.e. satisfies the axioms of Definition 2.3.2:*

- *the function $X \to \mathsf{fix} f$ is given by* (4.12)
- $\mathsf{isProp}(\mathsf{fix} f)$ *is shown in Main Lemma 4.1.1*
- *the recursion principle is given by composition with* (4.11)*.*

Therefore, the weak propositional truncation of collapsible types is actually definable. *If $\|-\|$ is part of the theory, $\|X\|$ and $\mathsf{fix} f$ are equivalent.* □

This has the following direct consequence:

Ⓐ **Theorem 4.1.4.** *A type X is collapsible, i.e. has a constant endomap, if and only if it has split support in the sense that $\|X\| \to X$.* □

We want to add the remark that $\mathsf{coll}X$ is actually still more than required to get from $\|X\|$ to X. The following statement (together with Theorem 4.1.4) shows that is is enough to have $f : X \to X$ which is *merely* constant:

Ⓐ **Proposition 4.1.5.** *For a type X, the following are logically equivalent:*

1. *X is collapsible*
2. *X has an endofunction f with a proof $\|\mathsf{const}_f\|$.*

The first direction is trivial, but its reversibility is interesting. We do *not* think that $\|\mathsf{const}_f\|$ implies const_f.

Proof of the non-trivial direction of Proposition 4.1.5. Assume f is an endofunction on X. From Main Lemma 4.1.1, we know that

$$\mathsf{const}_f \to \mathsf{isProp}(\mathsf{fix}f). \tag{4.14}$$

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

$$\|\mathsf{const}_f\| \to \mathsf{isProp}(\mathsf{fix}f). \tag{4.15}$$

Previously, we have constructed a map

$$\mathsf{const}_f \to \|X\| \to \mathsf{fix}f. \tag{4.16}$$

Let us write this implication as

$$\|X\| \to \mathsf{const}_f \to \mathsf{fix}f. \tag{4.17}$$

This trivially implies

$$\|X\| \times \|\mathsf{const}_f\| \to \mathsf{const}_f \to \mathsf{fix}f. \tag{4.18}$$

We assume $\|X\| \times \|\mathsf{const}_f\|$. From (4.15), we conclude that $\mathsf{fix}f$ is a proposition. Therefore, we may apply the recursion principle of the truncation and get

$$\|X\| \times \|\mathsf{const}_f\| \to \|\mathsf{const}_f\| \to \mathsf{fix}f, \tag{4.19}$$

which, of course, gives us

$$\|X\| \to \mathsf{fix}f \tag{4.20}$$

if we assume $\|\mathsf{const}_f\|$. Composing $|–|$ with (4.20) and with the first projection, we get a constant function $g : X \to X$. □

Note that, in the above proof, we could have used the induction principle (Lemma 2.3.4) instead of the "trick" of duplicating the assumption $\|\mathsf{const}_f\|$.

Further, 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:

$$\|\forall x.\, f(x) = g(x)\|. \tag{4.21}$$

The detailed formalised proof can be found in the electronic appendix.

4.2 Populatedness

The results on constant endofunctions enable us to define a notion of *anonymous existence*, similar to but weaker than propositional truncation. It crucially depends on the Fixed Point Lemma (Main Lemma 4.1.1). Let us start by discussing another perspective of what we have explained in Section 4.1.

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

$$\|X\| \to (\|X\| \to X) \to X. \tag{4.22}$$

By Theorem 4.1.4, this is equivalent to

$$\|X\| \to \mathsf{coll}X \to X, \tag{4.23}$$

and hence

$$\mathsf{coll}X \to \|X\| \to X, \tag{4.24}$$

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 (4.22), it is tempting 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 4.3.2, we give a proof that it is indeed weaker.

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

Ⓐ **Definition 4.2.1** (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:

$$\langle\langle X \rangle\rangle :\equiv \forall (f : X \to X).\, \mathsf{const}_f \to \mathsf{fix}f, \tag{4.25}$$

where $\mathsf{fix}f$ is the type of fixed points, defined as in (4.1.1).

This definition allows us to comment on the question risen above. If $\langle\langle X \rangle\rangle$ is inhabited and X is collapsible, 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 (4.24) would be sufficient as well. Therefore,

$$\langle\langle X \rangle\rangle \to (\|X\| \to X) \to X. \tag{4.26}$$

At this point, we have to ask ourselves whether (4.26) is an improvement over (4.24). But indeed, we have the following property:

Ⓐ **Proposition 4.2.2.** *Any merely inhabited type is populated. That is, for any type X, we have*

$$\|X\| \to \langle\langle X \rangle\rangle. \tag{4.27}$$

Proof. Assume f is a constant endofunction on X. The claim follows directly from Corollary 4.1.3. □

We will see later (Section 4.3.2) that $\langle\langle X \rangle\rangle$ is in fact strictly weaker than $\|X\|$. Note that from (4.26), Theorem 4.1.4 and Proposition 4.2.2 we immediately get the following:

Ⓐ **Corollary 4.2.3.** *For any type X, the following statements are logically equivalent:*

1. *X is collapsible,*
$$\Sigma\,(f : X \to X)\,.\,\mathsf{const}_f \tag{4.28}$$
2. *X has split support,*
$$\|X\| \to X \tag{4.29}$$
3. *X is inhabited if it is populated,*
$$\langle\langle X \rangle\rangle \to X. \tag{4.30}$$

In particular, if X is a proposition, (2) is always satisfied and we may conclude $\langle\langle X \rangle\rangle \to X$. □

In the presence of propositional truncation, we give an alternative characterisation of populatedness.

Ⓐ **Lemma 4.2.4.** *In MLTT with propositional truncation, a type X 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 implies X. In formula, the following types are logically equivalent:*

1. $\langle\langle X \rangle\rangle$
2. $\big\|\|X\| \to X\big\| \to \|X\|$
3. $(\|X\| \to X) \to X$.

Proof. We have already discussed $(1) \Rightarrow (3)$ above, see (4.26). $(3) \Rightarrow (2)$ follows from the functoriality of the truncation operator. For $(2) \Rightarrow (1)$, assume we have a constant endofunction f on X. This implies $\|X\| \to X$, thus $\big\|\|X\| \to X\big\|$ and, by assumption, $\|X\|$. But $\|X\|$ is enough to construct a fixed point of f by Corollary 4.1.3. □

One more characterisation of populatedness, and a strong parallel to mere inhabitance, is given by the following statement.

Ⓐ **Proposition 4.2.5.** *In MLTT, any given type X is populated if and only if any proposition that is logically equivalent to it holds,*

$$\langle\langle X \rangle\rangle \;\longleftrightarrow\; \forall (P : \mathcal{U}).\, \mathsf{isProp} P \to (P \to X) \to (X \to P) \to P. \tag{4.31}$$

Note that the only difference to the type expression in Proposition 2.3.6 is that we only quantify over *sub-propositions* of X, i.e. over those that satisfy $P \to X$, while we quantify over all propositions in the case of $\|X\|$. This again shows that $\|X\|$ is at least as strong as $\langle\langle X \rangle\rangle$.

Proof. Let us first prove the direction "$\to$". Assume a proposition P is given, together with functions $X \to P$ and $P \to X$. Composition of these gives us a constant endomap on X. But then $\langle\langle X \rangle\rangle$ makes sure that this constant endomap has a fixed point, and thus an inhabitant of X. Using $X \to P$ again, we get P.

For the direction "$\leftarrow$", assume we have a constant endomap f. We need to construct an inhabitant of $\mathsf{fix} f$. In the expression on the right-hand side, choose P to be $\mathsf{fix} f$, and everything follows from Corollary 4.1.3. □

The similarities between $\|X\|$ and $\langle\langle X \rangle\rangle$ do not stop here. The following statement, together with the direction "$\to$" of the statement that we have just proved, is worth to be compared to the definition of $\|X\|$ (that is, Definition 2.3.2):

Ⓐ **Proposition 4.2.6.** *For any type X, the type $\langle\langle X \rangle\rangle$ has the following properties:*

1. $X \to \langle\langle X \rangle\rangle$
2. $\mathsf{isProp}(\langle\langle X \rangle\rangle)$ *(if function extensionality holds).*

Proof. The first point follows immediately from the (stronger) statement of Proposition 4.2.2. For the second, we use once more that $\mathsf{fix} f$ is a proposition and that, by function extensionality, truncation levels are closed under Π (Lemma 2.2.6). □

4.3 Comparison of Notions of Existence

Let us now compare the various notions of inhabitance we have encountered. We have, for any type X, the following chain of implications:

$$X \longrightarrow \|X\| \longrightarrow \langle\langle X \rangle\rangle \longrightarrow \neg\neg X. \tag{4.32}$$

The first implication is trivial and the second is given by Proposition 4.2.2. 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

$$\langle\langle X \rangle\rangle \to \neg\neg X\,, \tag{4.33}$$

let us assume $f : \neg X$. But then, f can be composed with the canonical function from the empty type into X, yielding a constant endofunction 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_{-1}, all implications of the chain (4.32) except the first can be reversed as it is easy to show

$$\forall(X:\mathcal{U}).\,(\|X\| + \neg\|X\|) \to \neg\neg X \to \|X\|. \tag{4.34}$$

Of course, if we even assume LEM_{∞}, then the four conditions in the chain (4.32) become logically equivalent by the same argument: (4.34) remains true if we remove the truncation operators.

Constructively, none of the implications of (4.32) should be reversible. To make that precise, we use *taboos*, showing that the provability of a statement would imply the provability of another, better understood statement, that is known to be not derivable. As a second technique, we use models. In this section, we present the following discussions:

1. If the first implication could be reversed, i.e. we had $\forall(X:\mathcal{U}).\,\|X\| \to X$, it would immediately follow from Proposition 3.1.10 that all types are sets, an inconsistent assumption in HoTT. We will show the less trivial consequence that all equalities would be decidable, even in a minimalistic version of MLTT. In the same minimalistic setting, we show that a form of choice that does not belong to intuitionistic type theory would be implied.

   Moreover, we observe that $\|X\| \to X$ can be read as "the map $|-| : X \to \|X\|$ is a split epimorphism" (where the latter notion requires to be read with care), and we show that already the weaker assumption that it is an epimorphism implies that all types are sets.

2. General reversibility of the second arrow is logically equivalent to a certain weak version of the axiom of choice.

3. If the last implication could be reversed, LEM_{-1} (and thereby AC_{-1}) would be derivable. Assuming function extensionality, it is in fact the case that LEM_{-1} can be shown to be logically equivalent to the type $\forall(X:\mathcal{U}).\,\neg\neg X \to \langle\langle X\rangle\rangle$.

These results, and in particular the lack of a more suspicious consequence in the second case, again yield some evidence that the differences between $\|X\|$ and $\langle\langle X\rangle\rangle$ are fairly subtle, and that mere inhabitance and populatedness are closer to each other than any two of the other notions of existence that we consider.

4.3.1 Inhabited and Merely Inhabited

Let us discuss certain consequences of the assumption

$$\forall(X:\mathcal{U}).\,\|X\| \to X. \tag{4.35}$$

First, using Theorem 4.1.4, this assumption is clearly equivalent to

$$\forall(X:\mathcal{U}).\,\mathsf{coll}X, \tag{4.36}$$

or, in words, "every type has a constant endofunction." Note that latter formulation means that, for any type X, we have a proposition that is logically equivalent to X, namely the type of fixed points of the given constant endofunction (Corollary 4.1.3). This is the case even in a type theory without propositional truncation.

From a constructive type of view, (4.36) is an interesting statement. 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. Intuitively, (4.36) seems to be related to the assumption that every type is either empty or inhabited, but it does not tell us in which case we are.

Of course, (4.36) is an inconsistent in HoTT as it implies that all types are sets. Already without univalence it is very easily seen to imply the axiom of choice (Definition 2.3.7). If we have univalence for propositions and set quotients, this allows us to use Diaconescu's proof of LEM_{-1} ([Dia75], see [Uni13, Theorem 10.1.14]).

Let us instead consider a very minimalistic type theory without univalence, without function extensionality (and without truncations). We do not think that LEM_∞ can be derived in this minimalistic setting. However, what we can conclude is the ∞-version of excluded middle for all path spaces, i.e. that all types are discrete, see Lemma 4.3.1 and Proposition 4.3.2 below.

Ⓐ **Lemma 4.3.1.** *In basic MLTT without extensionality, without truncation, and even without a universe, let A be a type and $a_0, a_1 : A$ two points. If for all $x : A$ the type $(a_0 = x) + (a_1 = x)$ is collapsible, then $a_0 = a_1$ is decidable.*

Before giving the proof, we state an immediate corollary (which does involve a type universe):

Ⓐ **Proposition 4.3.2.** *In basic MLTT, if every type has a constant endofunction then every type has decidable equality,*

$$(\forall (X : \mathcal{U}).\, \mathsf{coll} X) \to \forall (X : \mathcal{U}).\, \mathsf{discrete} X. \tag{4.37}$$

□

Proof of Lemma 4.3.1. For (technical and conceptual) convenience, we regard the elements a_0, a_1 as a single map

$$a : \mathbf{2} \to A \tag{4.38}$$

and we use

$$E_x :\equiv \Sigma\, (i : \mathbf{2})\,.\ a_i = x \tag{4.39}$$

in place of the type $(a_0 = x) + (a_1 = x)$. This is justified by the fact that the property of being collapsible is clearly closed under type equivalence. In a theory with propositional truncation, the *image* of a can be defined to be $\Sigma\, (x : A)\,.\, \|E_x\|$

[Uni13, 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

$$E :\equiv \Sigma\,(x : A)\,.\,\mathsf{fix} f_x, \tag{4.40}$$

which can be unfolded to

$$\Sigma\,(x : A)\,.\,\Sigma\,((i,p) : E_x)\,.\,f_x(i,p) = (i,p), \tag{4.41}$$

as the image of a. It is essentially the observation that we can define this image that allows us to mimic Diaconescu's argument. Clearly, a induces a map

$$r : \mathbf{2} \to E \tag{4.42}$$
$$r(i) :\equiv (a_i, \epsilon(i, \mathsf{refl}_{a_i})). \tag{4.43}$$

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

$$r(i) = r(j) \leftrightarrow a_i = a_j. \tag{4.44}$$

The type E can be understood as the quotient of 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 [Uni13, Definition 7.6.3], the axiom of choice would be necessary to find a section of r (see [Uni13, Theorem 10.1.14]). In our situation, this section is given by a simple projection,

$$s : E \to \mathbf{2} \tag{4.45}$$
$$s(x, ((i,p),q)) :\equiv i. \tag{4.46}$$

It is easy to see that s is indeed a section of r in the sense of $\forall (e : E).\, r(s(e)) = e$. Given $(x, ((i,p),q)) : E$, applying first s, then r leads to $(a_i, \epsilon(i, \mathsf{refl}_{a_i}))$. 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 $e_0, e_1 : E$,

$$e_0 = e_1 \leftrightarrow s(e_0) = s(e_1). \tag{4.47}$$

Combining (4.44) and (4.47) yields

$$a_i = a_j \leftrightarrow s(r(i)) = s(r(j)), \tag{4.48}$$

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

In the same minimalistic setting, we want to show that (4.36) further implies a form of choice that does not pertain to intuitionistic type theory. In order to formulate and prove this, we need a few definitions.

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

$$\forall (x\,y : X).\, \mathsf{isProp}(R(x,y)). \tag{4.49}$$

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

$$R_x :\equiv \Sigma\,(y : X)\,.\,R(x, y). \tag{4.50}$$

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

$$\forall(x : X).\,\mathsf{isProp}R_x. \tag{4.51}$$

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

$$\forall(x : X).\,R_x \longleftrightarrow S_x, \tag{4.52}$$

and that S is a *subrelation* of R if

$$\forall(x\,y : X).\,S(x, y) \to R(x, y). \tag{4.53}$$

Ⓐ **Proposition 4.3.3.** *If all types are collapsible, then every binary relation has a functional, propositionally valued subrelation with the same domain.*

Proof. Assume that $R : X \times X \to U$ is given. For $x : X$, let $k_x : R_x \to R_x$ be the constant map given by the assumption (4.36) that all types are collapsible. Define further

$$S(x, y) :\equiv \Sigma\,(a : R(x, y))\,.\,(y, a) = k_x(y, a). \tag{4.54}$$

Then S is a subrelation of R by construction. We observe that S_x is equivalent to $\mathsf{fix}(k_x)$ and therefore propositional (by Main Lemma 4.1.1), proving that S is functional. Together with Corollary 4.1.3, this further implies

$$R_x \longleftrightarrow \mathsf{fix}k_x \longleftrightarrow S_x, \tag{4.55}$$

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 $(y, s) =_{S_x} (y, s')$. This type corresponds to a dependent pair type with components

$$p : y =_X y \tag{4.56}$$

$$q : p_*(s) =_{S(x,y)} s'. \tag{4.57}$$

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

Instead of (4.36), let us now consider the assumption in the logically equivalent formulation (4.35). Note that a map $h : \|X\| \to X$ is automatically a section of $|{-}| : X \to \|X\|$ in the sense of

$$\forall(z : \|X\|).\,|h(z)| = z \tag{4.58}$$

as any two inhabitants of $\|X\|$ are equal. Therefore, we may read (4.35) as:

$$\text{For any type } X\text{, the map } |{-}| : X \to \|X\| \text{ is a } \textit{split epimorphism}. \tag{4.59}$$

We want to consider a weaker assumption, namely

$$\text{For any type } X, \text{ the map } |{-}| : X \to \|X\| \text{ is an } \textit{epimorphism}, \tag{4.60}$$

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

$$(\forall u.\, f(e\,u) = g(e\,u)) \to \forall v.\, f\,v = g\,v. \tag{4.61}$$

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

$$f \circ e = g \circ e \to f = g. \tag{4.62}$$

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 propositional equality is not only an ordinary category, but rather an $(\omega, 1)$-category. The definition (4.61) makes sense in the sub-universe of sets [Uni13, 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 as it can be argued that LEM_∞ is a problematic version of the principle of excluded middle (see [Uni13, Chapter 3]). Despite of this, we use the notion as we think that it helps providing an intuitive meaning to the plain type expression (4.61).

Ⓐ **Lemma 4.3.4.** *Let Y be a type. If the map $|{-}| : (y_1 = y_2) \to \|y_1 = y_2\|$ 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

$$f, g : \|y_1 = y_2\| \to Y \tag{4.63}$$

by

$$f(q) :\equiv y_1, \tag{4.64}$$
$$g(q) :\equiv y_2, \tag{4.65}$$

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

With these concrete choices, our assumption (4.61) with $e \equiv |{-}|$ becomes

$$(y_1 = y_2 \to y_1 = y_2) \to (\|y_1 = y_2\| \to y_1 = y_2) \tag{4.66}$$

which, of course, implies

$$\|y_1 = y_2\| \to y_1 = y_2. \tag{4.67}$$

The statement of the lemma then follows from Proposition 3.1.10. □

In the following statement, we include the theorem about decidable equality from above again to directly compare it with the just established result:

Ⓐ **Proposition 4.3.5.** *In basic MLTT with weak propositional truncation,*

1. *if $|-| : X \to \|X\|$ is a split epimorphism for every X, then all types have decidable equality*
2. *if $|-| : X \to \|X\|$ is an epimorphism for every X, then all types are sets.*

Proof. The first part is a reformulation of Proposition 4.3.2, while the second part is a corollary of Lemma 4.3.4. □

4.3.2 Merely Inhabited and Populated

Assume that the second implication can be reversed, meaning that we have

$$\forall(X : \mathcal{U}).\, \langle\langle X \rangle\rangle \to \|X\|. \tag{4.68}$$

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 expect every type to have split support. However, a weaker version of this is provable:

Ⓐ **Lemma 4.3.6.** *For every type X, the statement that it has split support is populated,*

$$\forall(X : \mathcal{U}).\, \langle\langle \|X\| \to X \rangle\rangle. \tag{4.69}$$

To demonstrate the different possibilities that the logically equivalent formulations of populatedness offer, we want to give three different proofs. We have formalised all three proofs in the electronic appendix. The first one uses Definition 4.2.1:

First proof. Assume we are given a constant endofunction f on $\|X\| \to X$. We need to construct a fixed point of f, or correspondingly, any inhabitant of $\|X\| \to X$. By Theorem 4.1.4, a constant function $g : X \to X$ is enough for this. Given $x : X$, we may apply f on the function that is everywhere x, yielding an inhabitant of $\|X\| \to X$. Applying it on $|x|$ 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. □

Alternatively, we can use the logically equivalent formulation of populatedness, proved in Proposition 4.2.5:

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

$$P \leftrightarrow (\|X\| \to X). \tag{4.70}$$

Applying Proposition 4.2.5, it suffices to show P. The logical equivalence above immediately provides an inhabitant of $X \to P$, and, by the rules of the propositional truncation, therefore $\|X\| \to P$. Assume $\|X\|$. We get P, thus $\|X\| \to X$ with the above equivalence, and therefore X (using the assumed $\|X\|$ again). This shows $\|X\| \to X$, and consequently, P. □

For yet another and possibly the most elegant proof, we may use that $\langle\!\langle - \rangle\!\rangle$ can be written in terms of $\|-\|$.

Third proof. Using Lemma 4.2.4 (direction $(3) \Rightarrow (1)$), the statement that needs to be shown becomes

$$\big(\big\|\|X\| \to X\big\| \to \|X\| \to X\big) \to (\|X\| \to X)\,, \tag{4.71}$$

which is immediate. □

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

Ⓐ **Proposition 4.3.7.** *In MLTT with weak propositional truncation, the following are logically equivalent:*

1. *every populated type is merely inhabited,*

$$\forall(X : \mathcal{U}).\, \langle\!\langle X \rangle\!\rangle \to \|X\| \tag{4.72}$$

2. *every type merely has split support,*

$$\forall(X : \mathcal{U}).\, \big\|\|X\| \to X\big\| \tag{4.73}$$

3. *every proposition is* projective *in the following sense:*

$$\forall(P : \mathcal{U}).\, \mathsf{isProp}P \to \forall(Y : P \to \mathcal{U}).\, (\Pi_{p:P} \|Y(p)\|) \to \|\Pi_P Y\| \tag{4.74}$$

   (note that this is the axiom of choice AC_{-1} *(Definition 2.3.7) for propositions, without the requirement that* Y *is a family of sets)*

4. $\langle\!\langle - \rangle\!\rangle : \mathcal{U} \to \mathcal{U}$ *is* functorial *in the sense that*

$$\forall(X\,Y : \mathcal{U}).\, (X \to Y) \to (\langle\!\langle X \rangle\!\rangle \to \langle\!\langle Y \rangle\!\rangle), \tag{4.75}$$

   where the terminology functorial *is justified at least in the presence of function extensionality which implies that* $\langle\!\langle X \rangle\!\rangle \to \langle\!\langle Y \rangle\!\rangle$ *is propositional, ensuring* $\langle\!\langle g \circ f \rangle\!\rangle = \langle\!\langle g \rangle\!\rangle \circ \langle\!\langle f \rangle\!\rangle$.

Proof. We show all the implications that we know nice arguments for, and those are sufficient (and even more than necessary) to prove the theorem.

The equivalence of the first two points follows easily from what we already know. $(1) \Rightarrow (2)$ is an application of Lemma 4.3.6, while $(2) \Rightarrow (1)$ follows easily from Lemma 4.2.4.

Regarding the equivalence of the first and the last point, $(1) \Rightarrow (4)$ is immediate by functoriality of $\|-\|$. Turned around, if (4) holds, the map $|-|$ gives rise to a function $\langle\!\langle X \rangle\!\rangle \to \langle\!\langle \|X\| \rangle\!\rangle$, and for any propositional P, the types P and $\langle\!\langle P \rangle\!\rangle$ are equivalent.

Let us now show $(1) \Rightarrow (3)$. Let P be some proposition and $Y : P \to \mathcal{U}$ some family of types. If we assume (1), it is then enough to prove

$$\Pi_{p:P} \|Y(p)\| \to \langle\langle \Pi_P Y \rangle\rangle. \tag{4.76}$$

By Lemma 4.2.4, it is enough to show

$$\Pi_{p:P} \|Y(p)\| \to (\|\Pi_P Y\| \to \Pi_P Y) \to \Pi_P Y. \tag{4.77}$$

We can reorder the assumptions of this type. In particular, we may move the very last assumed point in P to the beginning and thus transform the type (4.77) into

$$\Pi_{p_0:P}\big(\Pi_{p:P} \|Y(p)\| \to (\|\Pi_P Y\| \to \Pi_P Y) \to Y(p_0)\big). \tag{4.78}$$

Recall the principle of the *neutral contractible exponent*, Lemma 2.2.9. It allows us to replace $\Pi_P Y$ by $Y(p_0)$ and $\Pi_{p:P} \|Y(p)\|$ by $\|Y(p_0)\|$, and (4.78) becomes the trivially inhabited type

$$\Pi_{p_0:P}\big(\|Y(p_0)\| \to (\|Y(p_0)\| \to Y(p_0)) \to Y(p_0)\big). \tag{4.79}$$

$(3) \Rightarrow (2)$ can be seen easily by taking P to be $\|X\|$ and Y to be constantly X. □

We conjecture that Proposition 4.3.7 can be used to show that

$$\forall (X : \mathcal{U}).\, \langle\langle X \rangle\rangle \to \|X\| \tag{4.80}$$

is not derivable in MLTT with weak propositional truncation. Consider the third of the four statements that we prove to be logically equivalent. When $Y(p)$ is a set with exactly two elements for every $p : P$, this amounts to *the world's simplest axiom of choice* [FŠ82]. There are details that have yet to be checked, but we think that this principle fails in some toposes that model MLTT.

4.3.3 Populated and Non-Empty

If we can reverse the last implication of the chain, we have

$$\forall (X : \mathcal{U}).\, \neg\neg X \to \langle\langle X \rangle\rangle. \tag{4.81}$$

To show that this is not derivable, we show that it would imply LEM_{-1}, a constructive taboo.

Ⓐ **Proposition 4.3.8.** *With function extensionality, the following implication holds:*

$$(\forall (X : \mathcal{U}).\, \neg\neg X \to \langle\langle X \rangle\rangle) \;\to\; \mathsf{LEM}_{-1}. \tag{4.82}$$

Proof. Assume P is a proposition. Then, so is the type $P + \neg P$: assume we have two inhabitants. Each of them can either come from an inhabitant of P or from an inhabitant of $\neg P$, giving us four cases. If both inhabitants come from P, they are equal as P is propositional. The same argument works if both come from $\neg P$, as $\neg P$ is propositional under function extensionality. In the remaining two cases, we have a proof of both P and $\neg P$, yielding a contradiction. Therefore, the identity function on $P + \neg P$ is constant.

It is also straightforward to construct a proof of $\neg\neg\,(P + \neg P)$. By the assumption, this means that $P + \neg P$ is populated, implying by definition that the identity function has a fixed point. But finding a fixed point of the identity function is the same as proving $P + \neg P$. □

It is well-known that LEM_{-1} implies $\neg\neg P \to P$ for any proposition. Using Proposition 4.2.6, this shows the other direction of Proposition 4.3.8. Thus, we have derived:

Ⓐ **Corollary 4.3.9.** *Under the assumption of function extensionality,* LEM_{-1} *holds if and only if all nonempty types are populated.* □

Chapter 5

Weakly Constant Functions

In this chapter, we will see why the attribute *weak* in the definition of constancy, as given in Definition 3.1.2, is justified. Therefore, we do not always drop the attribute any more, although weak constancy is still the only notion of constancy that we consider.

In Theorem 4.1.4 we have seen that a weakly constant function $f : X \to X$ implies that X has split support. On the other hand, what we have done is actually slightly more: the constructed map $\overline{f} : \|X\| \to X$ has the property that

$$\overline{f} \circ |-| : X \to X \tag{5.1}$$

is pointwise equal to f.

It seems a natural question to ask whether the fact that f is an endofunction is required: given a weakly constant function $f : X \to Y$, can it be factored in this sense through $\|X\|$?

Some of our results need function extensionality, in which cases we try to make that fact clear. Most contents of this rather short chapter are part of our publication [KECA14].

5.1 The Limitations of Weak Constancy

Let us start by giving a precise definition.

Ⓐ **Definition 5.1.1.** Given a function $f : X \to Y$ between two types, we say that f *factors* (propositionally) through a type Z if there are functions $f_1 : X \to Z$ and $f_2 : Z \to Y$ such that

$$\Pi_{x:X}\ f_2(f_1(x)) =_Y f(x). \tag{5.2}$$

In particular, we say that f factors (propositionally) through $\|X\|$ if there is a function $\overline{f} : \|X\| \to Y$ such that

$$\Pi_{x:X}\ \overline{f}(|x|) =_Y f(x). \tag{5.3}$$

We only consider propositional factorisation in this chapter, and we will drop the attribute *propositionally* and just say that a function *factors*. We will later see that, assuming judgmental computation for $\|-\|$, we can always construct a judgmental factorisation from a propositional one (see Section 6.3).

We could indeed assume that a constant function $f : X \to Y$ factors through $\|X\|$ if we expect $\|X\|$ to be a *quotient* of X in the more "traditional" sense. Before the development of HoTT, the quotient X/R of X by a relation $R : X \times X \to \mathcal{U}$ was defined to have an eliminator that allows to construct a function $\overline{f} : (X/R) \to Y$ whenever a map $f : X \to Y$ with the property

$$\forall (x, y : X).\, R(x, y) \to f(x) = f(y) \tag{5.4}$$

is given (see [Hof95; AAL11; Li15]). If we want to view $\|X\|$ as X divided by the chaotic relation that relates each pair of elements of X, this elimination principle amounts exactly to the extension of a constant function $X \to Y$ to a function $\|X\| \to Y$. This is indeed the case under the assumption of unique identity proofs as we will see later (Proposition 5.2.3).

However, the homotopical view suggests that it is unreasonable to expect such an extension property in the general case precisely because we have no way of knowing what happens on the (higher) path spaces. Consider the case that X is the coproduct of three propositions, $X \equiv P + Q + R$. Let us write $\mathsf{in}_1 : P \to X, \mathsf{in}_2 : Q \to X, \mathsf{in}_3 : R \to X$ for the three embeddings. Assume that, for some function $f : X \to Y$, we have three "potential paths"

$$c_{12} : \Pi_{p:P} \Pi_{q:Q} f(\mathsf{in}_1 p) = f(\mathsf{in}_2 q), \tag{5.5}$$
$$c_{23} : \Pi_{q:Q} \Pi_{r:R} f(\mathsf{in}_2 q) = f(\mathsf{in}_3 r), \tag{5.6}$$
$$c_{13} : \Pi_{p:P} \Pi_{r:R} f(\mathsf{in}_1 p) = f(\mathsf{in}_3 r). \tag{5.7}$$

A priori, we do not know which of P, Q and R are inhabited so we do not know which of these paths actually exists. Exploiting that P, Q and R are propositional, it is straightforward to construct a proof that f is constant out of this data. Further, we get by the fact that $\|X\|$ is propositional the proofs

$$h_{12} : \Pi_{p:P} \Pi_{q:Q} |\mathsf{in}_1 p| = |\mathsf{in}_2 q|, \tag{5.8}$$
$$h_{23} : \Pi_{q:Q} \Pi_{r:R} |\mathsf{in}_2 q| = |\mathsf{in}_3 r|, \tag{5.9}$$
$$h_{13} : \Pi_{p:P} \Pi_{r:R} |\mathsf{in}_1 p| = |\mathsf{in}_3 r|. \tag{5.10}$$

Let us now assume that there is a way to factor any generic constant function through the propositional truncation.

In our situation, we then get $\overline{f} : \|X\| \to Y$. If we are further given inhabitants $p : P$, $q : Q$ and $r : R$, our situation is pictured in Figure 5.1. In that figure, the arrows are the equality proofs, where we omit the arguments of c_{ij} and h_{ij}. The three unlabelled lines which "connect" the outer and the inner triangle are given by the fact that f and $f \circ \mathsf{ap}_{\_}$ are pointwise equal.

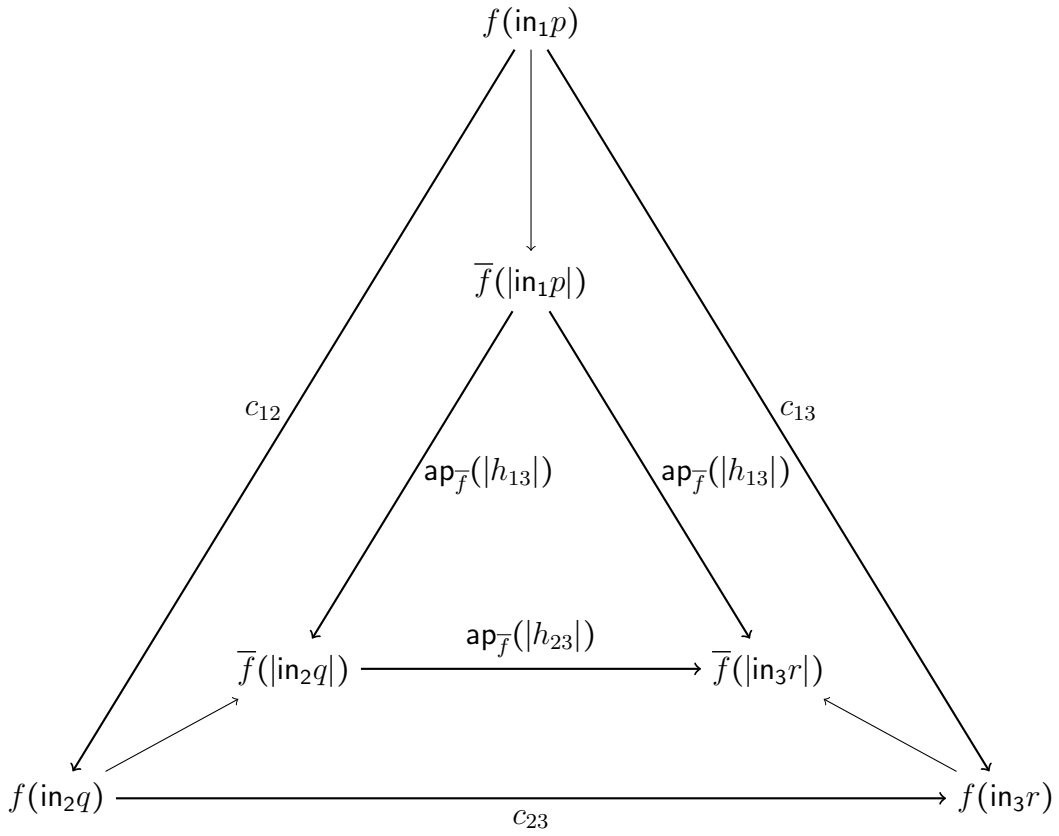


Figure 5.1: Factorising a non-coherently constant function

Further, note that $h_{12} \cdot h_{23} = h_{13}$ is automatically satisfied since $\|X\|$ is propositional. Looking at the smallest triangle in the above diagram we can conclude that it commutes due to the usual functoriality of ap (see [Uni13, Lemma 2.2.2]), which means there is a proof of

$$\mathsf{ap}_{\overline{f}}(h_{12}) \cdot \mathsf{ap}_{\overline{f}}(h_{23}) = \mathsf{ap}_{\overline{f}}(h_{31}). \tag{5.11}$$

The large triangle will in general not commute as identity proofs are not necessarily unique. If we only regard those parts of the diagram that do not mention the element r, we get a quadrangle in the top-left part. A similar observation holds for p and q so that we have three such quadrangles.

Let us go back one step. Assume that we are given the function f with the c_{ij} and (only) two points $p : P$ and $q : Q$. The path type $f(\mathsf{in}_1 p) = f(\mathsf{in}_2 q)$ is inhabited by $c_{12}(p, q)$. However, we should not expect to be able to construct an inhabitant that is not propositionally equal to this one. If we regard P and Q as two copies of the unit type, the only paths that we are able to construct are built out of refl and the c_{ij}. We further argue that the terms c_{23} and c_{13} can not be used for a generic R, as we can not know whether they actually provide any data

(R could be empty). Therefore, if there is a way to construct the factorisation of f, we expect it to not give us a "new" proof of $f(\mathsf{in}_1 p) = f(\mathsf{in}_2 q)$, i.e. we expect the quadrangle to commute, and the other quadrangles by analogous arguments. However, this contradicts the observation that the large triangle will in general not commute.

While this is not a rigorous argument, it hopefully provides some intuition. It seems that such a factorisation would need to make some form of choice on the path spaces if the constancy proof does not satisfy certain coherence laws. In general, we cannot make such a non-canonical choice. We do not know whether the assumption that every constant function factors through the propositional truncation makes it possible to derive a clearer version of choice, such as LEM_{-1} or the axiom of choice AC_{-1} (see Definition 2.3.7). This question is still open (see also the concluding remarks in the final chapter). A meta-theoretic proof sketch that the factorisation is not possible was described to us by Shulman in an online discussion [Hmail].

Remark 5.1.2. In later parts of this thesis, there are two more results that are directly related to the concrete problem of factoring a constant function $f : (P + Q + R) \to Y$ through the propositional truncation:

1. In Chapter 8, we will prove that *coherently* constant functions $X \to Y$ correspond to functions $\|X\| \to Y$. What we are given in the situation above is only a weakly constant function, and what is lacking is exactly its coherence.

2. In Theorem 5.2.6, we will see that the factorisation of a weakly constant function $X \to Y$ is possible if X is the sum of only *two* propositions.

The connection between (1) and (2), and the case $X \equiv P+Q+R$ considered above, is the following. In the case of (2), the general coherence conditions of (1) do not hold automatically, but the proof of weak constancy can be replaced by a coherent proof. This is essentially the idea of Theorem 5.2.6, see also the analysis at the end of Chapter 5. In the case that X is the sum of three propositions, such a coherent replacement is not possible in general. However, one single coherence condition (the one given in Proposition 8.1.2) would be enough to construct such a coherent replacement, and consequently the factorisation.

5.2 Factorisation for Special Cases

Even though we cannot factor weakly constant functions in general, we can do it in some interesting special cases.

Constructing a function out of the propositional truncation of a type is somewhat tricky. A well-known [Uni13, Chapter 3.9] strategy for defining a map $\|X\| \to Y$ is to construct a proposition P together with functions $X \to P$ and $P \to Y$. 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 5.2.1.** *Let X, Y be two types. Assume P is a type such that $P \to Y$. If X implies that P is contractible, then $\|X\|$ implies Y. In particular, if $f : X \to Y$ is a function that factors through P, then f factors through $\|X\|$.*

Let us shortly 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 $P \to Y$ clearly gives rise to a map $\|X\| \times P \to Y$, and the map $X \to \|X\| \times P$ is given by $|–|$ and the fact that P is contractible under the assumption X. □

There are several situations in which this principle can be applied. The following statement does not need it as it is mostly a restatement of our previous result from Section 4.1.

Ⓐ **Proposition 5.2.2.** *A weakly constant function $f : X \to Y$ factors through $\|X\|$ in any one of the following cases, of which the equivalent* (3) *and* (4) *generalise all others:*

1. *X is empty, i.e. $X \to \mathbf{0}$*
2. *X is inhabited, i.e. $\mathbf{1} \to X$*
3. *X has split support, i.e. $\|X\| \to X$*
4. *X is collapsible, i.e. has a weakly constant endofunction*
5. *we have any function $g : Y \to X$.*

Proof. (1) and (2) both imply (3). Further, (5) implies (4) as the composition $g \circ f$ is a constant endofunction on X. The equivalence of (3) and (4) is Theorem 4.1.4. Thus, it is sufficient to prove the statement for (3), so assume $s : \|X\| \to X$. The required conclusion is then immediate as f is pointwise equal to the composition of $|–| : X \to \|X\|$ and $f \circ s$. □

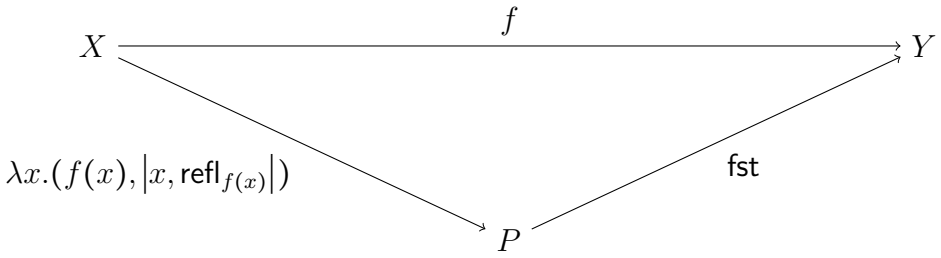
Our next statement implies what we mentioned at the beginning of Section 5.1: under the assumption of unique identity proofs, the factorisation is always possible. This will be hugely generalised in Chapter 8.

Ⓐ **Proposition 5.2.3.** *Let X, Y be again two types and $f : X \to Y$ a constant function. If Y is a set, then f factors through $\|X\|$.*

Proof. We use Principle 5.2.1 (or actually rather the comment preceding it, the strengthened version is not needed here) and define

$$P :\equiv \Sigma\,(y : Y)\,.\,\|\Sigma\,(x : X)\,.\,f(x) =_Y y\|. \tag{5.12}$$

We can see that f factors through P from the following diagram:



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 $y_1 = y_2$. □

In the setting of Proposition 5.2.3, we can say even more:

Proposition 5.2.4. *Assuming function extensionality, the function*

$$\tau : (\Sigma\,(f : X \to Y)\,.\,\mathsf{const}_f) \to (\|X\| \to Y), \tag{5.13}$$

as constructed in Proposition 5.2.3, is an equivalence.

Proof. We claim that the inverse of τ is given by

$$\sigma : (\|X\| \to Y) \to (\Sigma\,(f : X \to Y)\,.\,\mathsf{const}_f) \tag{5.14}$$

$$\sigma(g) :\equiv \left(g \circ |-|\ ,\ \lambda(x_1, x_2 : A).\mathsf{ap}_g(h_{|x_1|,|x_2|})\right), \tag{5.15}$$

where $h_{|x_1|,|x_2|} : |x_1| = |x_2|$ comes from the axiom that $\|X\|$ is propositional. Assume $f : A \to B$ and $c : \mathsf{const}_f$. We first need to show $(f, c) = \sigma(\tau(f, c))$. By the assumed function extensionality, the second components live in propositional types, and equality of the first components is exactly the statement that we have factored f through $\|X\|$, i.e. Proposition 5.2.3.

The missing part is to show that $\tau(\sigma(g)) = g$ for any $g : \|X\| \to Y$, i.e. showing that, if we factor $g \circ |-|$ through $\|X\|$, we get g. But if we factor $g \circ |-|$ (with the canonical constancy proof) through $\|X\|$, we get a function which, when applied on $|x|$ for any $x : X$, takes $g(|x|)$ as value. This is enough by function extensionality and the induction principle of $\|-\|$, using once more that equality in Y is propositional. □

Proposition 5.2.4 is the first non-trivial special case of the "general universal property of the propositional truncation" that we present in Chapter 8. We will see a different proof for the same statement in Proposition 8.1.2.

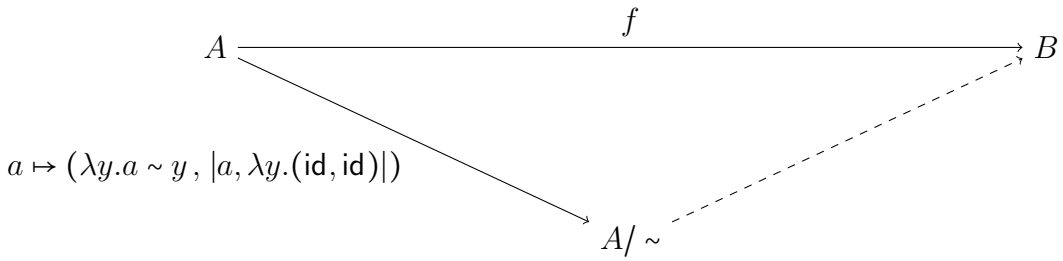
Let us mention the following application of Proposition 5.2.3:

Example 5.2.5 (Elimination rule for set-quotients)**.** The operation of quotienting by an equivalence relation was considered long before the development of HoTT, in extensional (see [Con+86]) as well as in intenstional type theories [Hof95]. Formulated in the theory discussed here (see [Uni13, Chapter 6.10]), we can consider a set A and a (propositional) equivalence relation $\sim : A \times A \to \mathcal{U}^{-1}$. One can regard a (propositionally-valued) family $P : A \to \mathcal{U}^{-1}$ as an *equivalence class* if

$$\mathsf{isEquivClass}(P) :\equiv \left\| \Sigma\,(x : A)\,.\,\forall (y : A).\,(x \sim y) \leftrightarrow P(y) \right\|. \tag{5.16}$$

Then, one may define the quotient $A/\sim$ as $\Sigma\,(P : A \to \mathcal{U}^{-1})\,.\,\mathsf{isEquivClass}(P)$. This type-theoretic version of the well-known construction via equivalence classes is due to Voevodsky [Voe10b; Voe13b, file hSet.v] and requires the univalence axiom at least for propositional types. Using Proposition 5.2.3, we can derive the correct elimination principle. What we do is essentially Voevodsky's construction presented as an application of our lemma.

Let us assume that B is some set and $f : A \to B$ a function that "respects" the relation $\sim$, in the sense that $\forall (x\,y : A).\,x \sim y \to f(x) = f(y)$. The canonical elimination property of the quotient that one would expect is then that f can be extended to a function $(A/\sim) \to B$:



This is indeed the case: assume $P : A \to \mathcal{U}^{-1}$. There is a canonical function

$$\left(\Sigma\,(x : A)\,.\,\forall (y : A).\,(x \sim y) \leftrightarrow P(y)\right) \;\to\; B, \tag{5.17}$$

given by composing the first projection with f. Using that f respects $\sim$, it is easy to prove that the function (5.17) is weakly constant, and by Proposition 5.2.3 we thus get $\mathsf{isEquivClass}(P) \to B$. Summarised, we have $(A/\sim) \to B$ as required. □

For more on quotients in intensional type theories, we refer to the PhD thesis of Li [Li15].

Our last example of a special case in which the factorisation 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.

Ⓐ **Theorem 5.2.6.** *Assume that function extensionality holds. If $f : X \to Y$ is weakly constant and X is the coproduct of two propositional types, then f factors through $\|X\|$.*

The following proof greatly benefits from the equivalence reasoning style, shortening it significantly compared to the argument that I had originally used. This simplification was suggested by Sattler in a private communication with me in July 2013.

Proof of Theorem 5.2.6. Assume $X \equiv Q + R$, where Q and R are propositional types. Define P to be the following Σ-type with four components:

$$\begin{aligned} P :\equiv \Sigma\,(y : Y)& \\ \Sigma\,(s : \Pi_{q:Q}\; y = f(\mathsf{inl}\, q))& \\ \Sigma\,(t : \Pi_{r:R}\; y = f(\mathsf{inr}\, r))& \\ \big(\Pi_{q:Q}\Pi_{r:R}\; s(q)^{-1} \cdot t(r) &= c(\mathsf{inl}\, q\,,\, \mathsf{inr}\, r)\big)\,. \end{aligned} \tag{5.18}$$

The proof is done in the equivalence reasoning style. In order to apply Principle 5.2.1 we need to construct a function $P \to Y$ and a proof that X implies that P is contractible.

The function $P \to Y$ 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 $\mathsf{inl}\, q_0$ with $q_0 : Q$. While the "naive" approach of finding a point that is equal to any other given inhabitant *can* be used to show that P is contractible, the following construction of a chain of equivalences yields a much nicer proof.

Let us first use the property of *neutral contractible exponents* (Lemma 2.2.9): instead of quantifying over all elements of Q, it suffices to only consider q_0. Applying this twice shows that P is equivalent to the following type:

$$\begin{aligned} &\Sigma\,(y : Y) \\ &\Sigma\,(s : y = f(\mathsf{inl}\, q_0)) \\ &\Sigma\,(t : \Pi_{r:R}\; y = f(\mathsf{inr}\, r)) \\ &\quad \big(\Pi_{r:R}\; s^{-1} \cdot t(r) = c(\mathsf{inl}\, q_0\,,\, \mathsf{inr}\, r)\big)\,. \end{aligned} \tag{5.19}$$

The first two components together form a singleton, showing that this part is contractible with the canonical inhabitant $(f(\mathsf{inl}\, q_0), \mathsf{refl})$. Applying the principle of *neutral contractible base spaces* (again Lemma 2.2.9), the above type further simplifies to

$$\begin{aligned} &\Sigma\,(t : \Pi_{r:R}\; f(\mathsf{inl}\, q_0) = f(\mathsf{inr}\, r)) \\ &\quad \big(\Pi_{r:R}\; \mathsf{refl}^{-1} \cdot t(r) = c(\mathsf{inl}\, q_0\,,\, \mathsf{inr}\, r)\big)\,. \end{aligned} \tag{5.20}$$

We apply the distributivity law (Lemma 2.2.12) and use that refl is neutral and self-inverse and neutral with respect to $\cdot$ to make a further transformation to

$$\begin{aligned} &\Pi_{r:R}\Sigma\,(t : f(\mathsf{inl}\, q_0) =_B f(\mathsf{inr}\, r)) \\ &\qquad (t = c(\mathsf{inl}\, q_0\,,\, \mathsf{inr}\, r))\,. \end{aligned} \tag{5.21}$$

For any $r : R$, the dependent pair part is contractible as it is a singleton, so that function extensionality implies the required result. □

Theorem 5.2.6 was inspired by a discussion on the *homotopy type theory mailing list* [Hmail]. Shulman observed that, for two propositions Q and R, their *join* $Q * R$ [Uni13, Chapter 6.8], defined as the (homotopy) pushout of the diagram

$$Q \xleftarrow{\mathsf{fst}} Q \times R \xrightarrow{\mathsf{snd}} R, \tag{5.22}$$

is equivalent to $\|Q + R\|$. This means that, in the presence of higher inductive types, the type $\|Q + R\|$ has the (seemingly) stronger elimination rule of the join. Escardó 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 5.2.6: 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.

It is interesting to analyse how the factorisations that we have constructed in Propositions 5.2.2 and 5.2.3 and Theorem 5.2.6 bypass the difficulties discussed in Section 5.1. As we will prove in Chapter 8, in general we need an infinite tower of coherence conditions in order to factor a weakly constant function through the propositional truncation. In the cases that we have solved in the current chapter, we can construct coherent constancy proofs for the following reasons:

1. If the codomain of a function is a set as in Proposition 5.2.3, the proof of weak constancy is automatically coherent.

2. If, as in Theorem 5.2.6, the domain is the sum $Q + R$ of two propositions, the constancy proof is still not coherent in general. What we exploit is essentially that it can be replaced by a coherent one: given $f : Q + R \to Y$ and $c : \mathsf{const}_f$, a coherent $c' : \mathsf{const}_f$ can be constructed by mapping $(\mathsf{inl}\, q_1, \mathsf{inl}\, q_2)$ to the proof that is induced by the fact that Q is propositional, and similarly in the case of $(\mathsf{inr}\, r_1, \mathsf{inr}\, r_2)$. In the more interesting cases, $(\mathsf{inl}\, q, \mathsf{inr}\, r)$ is sent to $c(\mathsf{inl}\, q, \mathsf{inl}\, r)$, and $(\mathsf{inr}\, r, \mathsf{inl}\, q)$ to $c(\mathsf{inl}\, q, \mathsf{inr}\, r)^{-1}$.

3. Consider a constant function $f : X \to Y$ together with any function $g : Y \to X$ as in Proposition 5.2.2. The function f does induce some form of asymmetry on the type Y. Usually, this asymmetry seems to be too weak to be useful, but the function g "sends it back" to the type X where it does allow us to make a choice of a point, namely the fixed point of the composition.

Chapter 6

On the Computation Rule of the Propositional Truncation

Homotopy type theory as introduced in the standard reference ([Uni13, Chapter 3.7]) has propositional truncations with the judgmental computation (β-) rule for the recursor,

$$\mathsf{rec}_{\mathsf{tr}}(P, h, f, |x|) \equiv_\beta f(x) \tag{6.1}$$

for any function $f : X \to P$ where $x : X$ and P is propositional with proof h, as well as the analogous rule for the induction principle,

$$\mathsf{ind}_{\mathsf{tr}}(P, h, f, |x|) \equiv_\beta f(x) \tag{6.2}$$

(were P might now depend on $\|X\|$, and so on). In our discussions, we did not assume those two strict equations to hold so far. This is not because we think a theory without them is to be preferred, it is because we simply did not need them. We agree with the very common view (see the introduction of [Uni13, Chapter 6]) that judgmental computation rules are often advantageous, not only for truncations, but for higher inductive types (see Section 2.3.2) in general. Without them, some expressions will 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. An interesting aspect is that the propositional induction principle follows from the propositional recursion principle (Lemma 2.3.4), but an induction rule with the desired judgmental behaviour can (I believe) *not* be derived even if (6.1) holds. In particular, the term constructed in Lemma 2.3.4 does *not* have the expected judgmental computation rule.

Having said this, the judgmental computation rules do have some other consequences which we find interesting and which we discuss in this chapter. So far, (nearly) all our developments have been internal to type theory. This is only partially the case for the results from this section, as any statement saying that some equation holds judgmentally is meta-theoretic. We thus can not implement such a statement as a type in a proof assistant like Agda, but we can still use Agda to

check our claims; for example, if

$$p : x = y \tag{6.3}$$
$$p :\equiv \mathsf{refl}_x \tag{6.4}$$

type checks, we may conclude that the equality does hold judgmentally. The results marked with the symbol Ⓐ are "formalised" in this sense; it is probably more accurate to say "checked" in this case.

This chapter is part of our publication [KECA14].

6.1 The Interval

The interval $\mathfrak{I}$ [Uni13, Chapter 6.3] as a higher inductive type is a type in homotopy type theory that consists of two points $i_0, i_1 : \mathfrak{I}$ and a path $\mathsf{seg} : i_0 =_{\mathfrak{I}} i_1$ between them. Its *recursion*, or *non-dependent elimination* principle says: Given

$$Y : \mathcal{U} \tag{6.5}$$
$$y_0 : Y \tag{6.6}$$
$$y_1 : Y \tag{6.7}$$
$$p : y_0 = y_1, \tag{6.8}$$

there exists a function $f : \mathfrak{I} \to Y$ such that

$$f(i_0) \equiv y_0 \tag{6.9}$$
$$f(i_1) \equiv y_1 \tag{6.10}$$
$$\mathsf{ap}_f(\mathsf{seg}) = p. \tag{6.11}$$

The induction principle is the corresponding dependent version. Assume

$$Y : \mathfrak{I} \to \mathcal{U} \tag{6.12}$$
$$y_0 : Y(i_0) \tag{6.13}$$
$$y_1 : Y(i_1) \tag{6.14}$$
$$p : y_0 =^{Y}_{\mathsf{seg}} y_1. \tag{6.15}$$

Then, there exists a dependent function $f : \Pi_{\mathfrak{I}} Y$ such that

$$f(i_0) \equiv y_0 \tag{6.16}$$
$$f(i_1) \equiv y_1 \tag{6.17}$$
$$\mathsf{apd}_f(\mathsf{seg}) = p. \tag{6.18}$$

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 (6.9, 6.10, 6.16, 6.17), while the rules for the paths (6.11,6.18) are only propositional.

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

Ⓐ **Proposition 6.1.1.** $\|\mathbf{2}\|$ *can be understood as the interval, with* $|0_{\mathbf{2}}|$ *as* i_0, $|1_{\mathbf{2}}|$ *as* i_1 *and* $\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}$ *as* seg. *In particular, the recursion and the induction principles of the interval are derivable. The recursion principle of the interval satisfies the expected judgmental computation rule if the recursion principle of the truncation does (see* (6.1)*), and the analogue relation holds for the induction principles (see* (6.2) *for the case of the truncation).*

Proof. We first show that the recursion principle is derivable which already contains contains the idea. We need to show that, under the assumptions 6.5-6.8, there is a function $f : \|\mathbf{2}\| \to Y$ such that

$$f(|0_{\mathbf{2}}|) \equiv y_0 \tag{6.19}$$
$$f(|1_{\mathbf{2}}|) \equiv y_1 \tag{6.20}$$
$$\mathsf{ap}_f(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}) = p. \tag{6.21}$$

Let us define

$$g : \mathbf{2} \to \Sigma\,(y : Y)\,.\,y_0 = y \tag{6.22}$$
$$g(0_{\mathbf{2}}) :\equiv (y_0, \mathsf{refl}) \tag{6.23}$$
$$g(1_{\mathbf{2}}) :\equiv (y_1, p). \tag{6.24}$$

As $\Sigma\,(y : Y)\,.\,y_0 = y$ is contractible, g can (via the recursion principle of the truncation) be extended to a function $\overline{g} : \|\mathbf{2}\| \to \Sigma\,(y : Y)\,.\,y_0 = y$, and we define $f :\equiv \mathsf{fst} \circ \overline{g}$. It is easy to check that f has indeed the required judgmental properties (6.19) and (6.20):

$$f(|0_{\mathbf{2}}|) \equiv \mathsf{fst}(\overline{g}(|0_{\mathbf{2}}|)) \equiv \mathsf{fst}(g(0_{\mathbf{2}})) \equiv \mathsf{fst}(y_0, \mathsf{refl}) \equiv y_0 \tag{6.25}$$
$$f(|1_{\mathbf{2}}|) \equiv \mathsf{fst}(\overline{g}(|1_{\mathbf{2}}|)) \equiv \mathsf{fst}(g(1_{\mathbf{2}})) \equiv \mathsf{fst}(y_1, p) \equiv y_1. \tag{6.26}$$

The propositional equality (6.21) is only slightly more difficult: First, using the definition of f and a standard functoriality property of ap (see [Uni13, Lemma 2.2.2 (iii)]), we observe that $\mathsf{ap}_f(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|})$ may be written as

$$\mathsf{ap}_{\mathsf{fst}}(\mathsf{ap}_{\overline{g}}(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|})). \tag{6.27}$$

But here, the path $\mathsf{ap}_{\overline{g}}(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|})$ lives in the contractible type $(y_0, \mathsf{refl}) = (y_1, p)$ (note that both terms inhabit a singleton) and thereby unique. In particular, it is (propositionally) equal to the path which is built out of two components, the first of which is p, and the second is a canonically constructed inhabitant of $p_*(\mathsf{refl}) = p$ (this is Lemma 2.2.1 with $h \equiv \lambda y.y_0$, $k \equiv \mathsf{id}$, $t \equiv p$, $p \equiv \mathsf{refl}$).

The second part, namely the induction principle, uses the the dependent version of the same construction. However, it adds some technical difficulties which we want to spell out. Assume we are given

$$Y : \|\mathbf{2}\| \to \mathcal{U} \tag{6.28}$$
$$y_0 : Y(|0_{\mathbf{2}}|) \tag{6.29}$$
$$y_1 : Y(|1_{\mathbf{2}}|) \tag{6.30}$$
$$p : y_0 =^{Y}_{\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}} y_1. \tag{6.31}$$

We define

$$g : \Pi_{x:\mathbf{2}} \Sigma\,(y : Y(|x|))\,.\,y_0 =^{Y}_{\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|x|}} y \tag{6.32}$$

$$g(0_{\mathbf{2}}) :\equiv (y_0, \mathsf{refl}) \tag{6.33}$$

$$g(1_{\mathbf{2}}) :\equiv (y_1, p). \tag{6.34}$$

This definition type checks due to the definition of the "path over" construction and the codomain of g is contractible as before (see Section 2.2.1). We use the induction principle of the truncation to construct

$$\overline{g} : \Pi_{z:\|\mathbf{2}\|} \Sigma\,(y : Y(z))\,.\,y_0 =^{Y}_{\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,z}} y \tag{6.35}$$

and define $f :\equiv \mathsf{fst} \circ \overline{g}$ just as before. Literally the same calculations (6.25, 6.26) can be done to verify that f has the required judgmental properties (6.16, 6.17).

Finally, we need to show $\mathsf{apd}_f(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}) = p$, and we note that (by functoriality of apd) the first expression equals

$$\mathsf{ap}_{\mathsf{fst}}(\mathsf{apd}_{\overline{g}}(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|})), \tag{6.36}$$

with

$$\mathsf{apd}_{\overline{g}}(\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}) : (y_0, \mathsf{refl}) =^{\lambda z.\Sigma(y:Y(z))\,.\,y_0 =^{Y}_{\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,z}} y}_{\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}} (y_1, p). \tag{6.37}$$

In principle, it would be possible to mimic the argument that we used for the non-dependent case. However, the type expression in (6.37) above is fairly involved, making such a treatment laborious if done carefully. Instead, we present a different way. We first generalise the statement slightly and then use path induction twice, enabling us to have refl in the place of $\mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}$ as well as in the place of p. In detail, we prove the following statement:

Claim. *Let P be a proposition (where, for any two points $q, r : P$, we write $\mathsf{h}_{\mathsf{q},\mathsf{r}}$ for the corresponding proof of $q = r$) and $Y' : P \to \mathcal{U}$ be a type family. Further, assume*

$$z, w : P \tag{6.38}$$

$$t : z = w \tag{6.39}$$

$$y'_0 : Y(z) \tag{6.40}$$

$$y'_1 : Y(w) \tag{6.41}$$

$$p' : y'_0 =^{Y'}_{t} y'_1. \tag{6.42}$$

as well as

$$\overline{g'} : \Pi_{v:P} \Sigma\,(y : Y'(v))\,.\,y_0 =^{Y'}_{\mathsf{h}_{\mathsf{z},\mathsf{v}}} y. \tag{6.43}$$

Then, we have

$$\mathsf{ap}_{\mathsf{fst}}(\mathsf{apd}_{\overline{g'}}(\mathsf{h}_{\mathsf{z},\mathsf{w}})) = p'. \tag{6.44}$$

From this claim, we can recover our actual goal by setting $P :\equiv \|\mathbf{2}\|$, $z :\equiv |0_{\mathbf{2}}|$, $w :\equiv |1_{\mathbf{2}}|$, $t :\equiv \mathsf{h}_{\mathsf{tr}|0_{\mathbf{2}}|,|1_{\mathbf{2}}|}$, $y'_0 :\equiv y_0$, $y'_1 :\equiv y_1$, $p' :\equiv p$ and $\overline{g'} :\equiv \overline{g}$.

Let us prove the claim: By path induction, it is enough to consider the case $z \equiv w$ and $t \equiv \mathsf{refl}$. This makes p' an inhabitant of the "ordinary" equality type $y'_0 = y'_1$. Using path induction again, we may assume $y'_0 \equiv y'_1$ and $p' \equiv \mathsf{refl}$. What is left to show is

$$\mathsf{ap}_{\mathsf{fst}}(\mathsf{apd}_{\overline{g'}}(\mathsf{h}_{\mathsf{w},\mathsf{w}})) = \mathsf{refl}. \tag{6.45}$$

As P (and thereby $w = w$) is propositional, we may replace $\mathsf{h}_{\mathsf{w},\mathsf{w}}$ by refl_w which makes the equality (6.45) hold judgmentally. □

6.2 Function Extensionality

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

Ⓐ **Lemma 6.2.1** (Shulman [Shu11])**.** *In a type theory with $\mathfrak{I}$ and the judgmental η-law for functions (which we assume), function extensionality is derivable.*

Proof. Assume X, Y are types and $f, g : X \to Y$ are functions with the property $h : \Pi_{x:X} f(x) = g(x)$. Using the recursion principle of $\mathfrak{I}$, we may then define a family

$$k : X \to \mathfrak{I} \to Y \tag{6.46}$$

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

$$k' : \mathfrak{I} \to X \to Y \tag{6.47}$$

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

The combination of Proposition 6.1.1 and Lemma 6.2.1 implies:

Ⓐ **Corollary 6.2.2.** *From propositional truncation with judgmental β and judgmental η for functions, function extensionality can be derived.* □

6.3 Judgmental Factorisation

The judgmental computation rules 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 generalisation of the fact that $\|\mathbf{2}\|$ satisfies the judgmental properties of the interval (Proposition 6.1.1).

Ⓐ **Proposition 6.3.1.** *Assuming that the recursion principle of the truncation satisfies the judgmental computation rule* (6.1)*, any (non-dependent) function that factors through the propositional truncation can be factored judgmentally: assume types X, Y and a function $f : X \to Y$ between them. Assume that there is $\overline{f} : \|X\| \to Y$ such that*

$$h : \Pi_{x:X} f(x) = \overline{f}(|x|). \tag{6.48}$$

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

$$f(x) \equiv f'(|x|), \tag{6.49}$$

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

Proof. We define a function

$$g : X \to \Pi_{z:\|X\|} \Sigma\,(y : Y)\,.\, y = \overline{f}(z) \tag{6.50}$$

$$g(x) :\equiv \lambda z.\,\big(f(x), h(x) \cdot \mathsf{ap}_{\overline{f}}(\mathsf{h}_{\mathsf{tr}|x|,z})\big) \tag{6.51}$$

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

$$\overline{g} : \|X\| \to \Pi_{z:\|X\|} \Sigma\,(y : Y)\,.\, y = \overline{f}(z). \tag{6.52}$$

We define

$$f' :\equiv \lambda(z : \|X\|).\mathsf{fst}(\overline{g}\, z\, z) \tag{6.53}$$

and it is immediate to check that f' has the required property:

$$f'(|x|) \;\equiv\; \mathsf{fst}(\overline{g}\,|x|\,|x|) \;\equiv\; \mathsf{fst}\,\big(f(x), h(x) \cdot \mathsf{ap}_{\overline{f}}(\mathsf{h}_{\mathsf{tr}|x|,|x|})\big) \;\equiv\; f(x). \tag{6.54}$$

□

Note that in the above argument we have only used (6.1). We have avoided (6.2) by introducing the variable z in (6.50), which is essentially a duplication of the first argument of the function, as it becomes apparent in (6.53).

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 $f : \Pi_{z:\|X\|} Y(z)$. In this case, it seems to be unavoidable to use (6.2), but the above proof still works with minimal adjustments. We state it for the sake of completeness.

Ⓐ **Proposition 6.3.2.** *Let X be a type and $Y : \|X\| \to \mathcal{U}$ a type family. Assume we have functions*

$$f : \Pi_{x:X} Y(|x|) \tag{6.55}$$

$$\overline{f} : \Pi_{z:\|X\|} Y(z) \tag{6.56}$$

such that

$$\Pi_{x:X}\left(f(x) =_{Y(|x|)} \overline{f}(|x|)\right). \tag{6.57}$$

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*

$$f(x) \equiv f'(|x|). \tag{6.58}$$

Proof. Because we allow ourselves to use (6.2) the proof becomes actually simpler than the proof above. This time, we can define

$$g : \Pi_{x:X} \Sigma\,(y : Y)\,.\,\left(y = \overline{f}(|x|)\right) \tag{6.59}$$

$$g(x) :\equiv (f(x), h(x))\,. \tag{6.60}$$

Using the induction principle, we get

$$\overline{g} : \Pi_{z:\|X\|} \Sigma\,(y : Y)\,.\,y = \overline{f}(z). \tag{6.61}$$

Then,

$$\lambda z.\mathsf{fst}(\overline{g}(z)) \tag{6.62}$$

fulfils the required condition. □

6.4 An Invertibility Puzzle

If we only have $\|X\|$, we usually do not know an inhabitant of X. It therefore is a reasonable intuition that

$$|-| : X \to \|X\| \tag{6.63}$$

can be understood as an "information hiding" function: a concrete $x : X$ is turned into an anonymous inhabitant $|x| : \|X\|$. While this interpretation is justified to some degree as long as we think of internal properties, it may be misleading from a meta-theoretic point of view.

Let us assume the univalence axiom as it allows us to construct some interesting equalities. We show that, for a non-trivial class of types, the projection map $|-|$ can be "pseudo-reversed". For example, there is a term that we call $\mathsf{myst}_{\mathbb{N}}$ such that

$$\mathsf{id}' : \mathbb{N} \to \mathbb{N} \tag{6.64}$$

$$\mathsf{id}'(n) :\equiv \mathsf{myst}_{\mathbb{N}}(|n|) \tag{6.65}$$

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

$$\mathsf{p} : \forall(n : \mathbb{N}).\,\mathsf{id}'(n) = n \tag{6.66}$$

$$\mathsf{p} :\equiv \lambda n.\mathsf{refl}_n. \tag{6.67}$$

We think that the possibility to do this is counter-intuitive and surprising. The term $\mathsf{myst}_{\mathbb{N}}$ seems to contradict the intuition that $|-|$ does not make any distinction between elements of $\mathbb{N}$; it sends any such inhabitant to the unique inhabitant of $\|\mathbb{N}\|$. We do indeed have the equalities

$$\mathsf{myst}_{\mathbb{N}}(|0|) \equiv 0 \tag{6.68}$$
$$\mathsf{myst}_{\mathbb{N}}(|1|) \equiv 1, \tag{6.69}$$

and the fact that these are not only propositional, but even judgmental, makes it even stranger. As we know $|0| =_{\|\mathbb{N}\|} |1|$, it might seem that we could prove $0 =_{\mathbb{N}} 1$ from the equations above. Of course, this is not the case. The sketched proof of $0 =_{\mathbb{N}} 1$ would work if the type of $\mathsf{myst}_{\mathbb{N}}$ (which we have not talked about yet) was $\|\mathbb{N}\| \to \mathbb{N}$, but it is not that simple. Let us perform the the construction to see what happens.[1]

First, let us state a useful general definition.

Ⓐ **Definition 6.4.1** (Transitive Type)**.** Given a type X, we call it *transitive* and write $\mathsf{isTransitive}X$ if it satisfies

$$\Pi_{x,y:X}(X, x) =_{\mathcal{U}_\bullet} (X, y). \tag{6.70}$$

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 $e_{xy} : X \to X$ such that $e_{xy}(x) = y$.

We have the following examples of transitive types:

Ⓐ **Example 6.4.2.** 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 6.4.3.** 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 $\Omega^n(X)$ is transitive for any pointed type X.

For a proof with the univalence axiom, 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. Surprisingly, we do not need univalence in the following alternative proof: Fix x_1. For any x_2 and $p : x_1 = x_2$, the pointed type $(x_1 = x_2, p)$ is by based path induction equal to $(x_1 = x_1, \mathsf{refl}_{x_1})$. The same is true for $(x_1 = x_2, q)$; hence, $(x_1 = x_2, p) = (x_1 = x_2, p)$. The claim that $\Omega^n(X)$ is transitive is a special case.[2]

As mentioned by Andrej Bauer in a discussion on this result [Kra13b], we also have the following:

[1] Further discussion can be found at my homotopy type theory blog entry [Kra13b] where I have presented the result originally.

[2] This is the proof that is formalised in the electronic appendix.

Example 6.4.4. Any group [Uni13, Definition 6.11.1] is a transitive type.

As for equality tyes, 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 6.4.5. If X is any type and $Y : X \to \mathcal{U}$ is a family of transitive types, then $\Pi_{x:X} Y(x)$ is transitive.

In particular, $\times$ and $\to$ preserve transitivity of types.

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

$$f : X \to \mathcal{U}_\bullet \tag{6.71}$$
$$f(x) :\equiv (X, x). \tag{6.72}$$

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

$$\overline{f} : \|X\| \to \mathcal{U}_\bullet \tag{6.73}$$
$$\overline{f}(z) :\equiv (X, x_0). \tag{6.74}$$

If X is transitive, we have

$$\Pi_{x:X} f(x) = \overline{f}(|x|). \tag{6.75}$$

By Proposition 6.3.1, there is then a function

$$f' : \|X\| \to \mathcal{U}_\bullet \tag{6.76}$$

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

$$f'(|x|) \equiv f(x) \equiv (X, x). \tag{6.77}$$

Let us define

$$\mathsf{myst}_X : \Pi_{z:\|X\|} \mathsf{fst}(f'(z)) \tag{6.78}$$
$$\mathsf{myst}_X :\equiv \mathsf{snd} \circ f'. \tag{6.79}$$

At this point, we can see where the puzzle comes from. The type of myst_X is *not* just $\|X\| \to X$; however, for any $x : X$, the type of $f'(|x|)$ is *judgmentally* equal to X, and we have $f'(|x|) \equiv x$. This already proves the following:

Ⓐ **Theorem 6.4.6.** *Let X be an inhabited transitive type. Then, there is a term myst_X such that the composition*

$$\lambda x.\mathsf{myst}_X(|x|) : X \to X \tag{6.80}$$

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

$$p : \Pi_{x:X} \mathsf{myst}_X(|x|) =_X x \tag{6.81}$$
$$p(x) :\equiv \mathsf{refl}_x \tag{6.82}$$

it trivial. □

It is tempting to unfold the type expression $\Pi_{z:\|X\|}\,\mathsf{fst}(f'(z))$ in order to better understand it. Unfortunately, this is not very feasible as this plain type expression involves the whole proof term f', which, in turn, includes the complete construction of Proposition 6.3.1.

Note that Theorem 6.4.6 does *not* mean that the identity function factors through $\|X\|$; because, being careful with this notion, this would require a retraction of $|{-}| : X \to \|X\|$, which we do *not* have. If we are given $x, y : X$, we do know $\mathsf{h}_{\mathsf{tr}|x|,|y|} : |x| =_{\|X\|} |y|$, but we *cannot* conclude

$$\mathsf{ap}_{\mathsf{myst}_X} : \mathsf{myst}_X(|x|) =_X \mathsf{myst}_X(|y|) \tag{6.83}$$

as this does not type check. Instead, we only have

$$\mathsf{apd}_{\mathsf{myst}_X}\big(\mathsf{h}_{\mathsf{tr}|x|,|y|}\big) \;:\; \mathsf{myst}_X(|x|) =^{\mathsf{fst}\circ f'}_{\mathsf{h}_{\mathsf{tr}|x|,|y|}} \mathsf{myst}_X(|y|). \tag{6.84}$$

Unfolding the definition of the *path over*-notation, this becomes

$$\mathsf{apd}_{\mathsf{myst}_X}\big(\mathsf{h}_{\mathsf{tr}|x|,|y|}\big) \;:\; \big(\mathsf{transport}^{\mathsf{fst}\circ f'}(\mathsf{h}_{\mathsf{tr}|x|,|y|}, x)\big) =_X y. \tag{6.85}$$

But this does not look wrong at all any more as $\mathsf{fst} \circ f'$ is an automorphism on X that sends x to y.

Finally, we want to remark that the construction of myst does not need the full strength of Proposition 6.3.1. The weaker version in which $\overline{f} : \|X\| \to Y$ is replaced by a fixed $y_0 : Y$ is sufficient: in this case, $\overline{f}$ can be understood to be *strictly* constant, or *constant at* y_0. This leads to a simplification as the dependent function types in (6.50) and (6.52) 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 we explained it originally [Kra13b]: We define

$$\mathsf{f} : X \to \Sigma\,(A : \mathcal{U}_\bullet)\,.\; A =_{\mathcal{U}_\bullet} (X, x_0) \tag{6.86}$$

$$\mathsf{f}(x) :\equiv ((X, x), \mathsf{transitive}_X(x, x_0)), \tag{6.87}$$

where $\mathsf{transitive}$ is the proof that A is transitive. The function f in (6.71) is then simply the composition $\mathsf{fst}\circ\mathsf{f}$. As the codomain of f is a singleton, it is contractible and thereby propositional (let us write h for the proof thereof). Hence, we get

$$\mathsf{f}' : \|X\| \to \Sigma\,(A : \mathcal{U}_\bullet)\,.\; A =_{\mathcal{U}_\bullet} (X, x_0) \tag{6.88}$$

$$\mathsf{f}' :\equiv \mathsf{rec}_{\mathsf{tr}}\,(\Sigma\,(A : \mathcal{U}_\bullet)\,.\; A =_{\mathcal{U}_\bullet} (X, x_0))\; h\,\mathsf{f}. \tag{6.89}$$

We could now define myst'_X to be

$$\mathsf{myst}'_X : \Pi_{\|X\|}\,\mathsf{fst} \circ \mathsf{fst} \circ \mathsf{f}' \tag{6.90}$$

$$\mathsf{myst}'_X :\equiv \mathsf{snd} \circ \mathsf{fst} \circ \mathsf{f}' \tag{6.91}$$

which has the same property as (6.79), even though it is not judgmentally the same term.

Chapter 7

Higher Homotopies in a Hierarchy of Univalent Universes

This chapter, probably a highlight of the thesis, contains a proof that the universe $\mathcal{U}_n$ in a hierarchy $\mathcal{U}_0, \mathcal{U}_1, \ldots$ of univalent universes is not n-truncated, and a construction of a "strict" n-type. Note that we do everything in a theory which does not necessarily have higher inductive types.

A joint article with Christian Sattler, containing the main results of this chapter, is published in *Transactions on Computational Logic* [KS15].

7.1 Background of the Problem

One of the most basic and well-known implications of the univalence axiom is that the first type universe, here written $\mathcal{U}_0$, is not a set. This is due to the fact that, for an example, the type $\mathbf{2}$ of boolean values is isomorphic to itself in two different ways, and these two isomorphisms give rise to two different inhabitants of $\mathbf{2} =_{\mathcal{U}_0} \mathbf{2}$. Reading through this argument, it seems plausible to assume that, as we go up the hierarchy of universes, we get types that can be shown to be not n-truncated for higher and higher n, meaning that they have a more and more complicated homotopical structure; in type-theoretic notation, we would ask for an inhabitant of the type

$$\neg\mathsf{is}\text{-}n\text{-}\mathsf{type}(\mathcal{U}_n). \tag{7.1}$$

The question was discussed several times at the univalent foundations special year program at the IAS in Princeton 2012/13, together with the very related problem of constructing a type that is "strictly" an n-type, that is finding a type X that is n-truncated but not $(n-1)$-truncated, using a hierarchy of univalent universes without higher inductive types. Even though there was no restriction on the universe levels, it is (and was) somewhat intuitive that the necessary universe level is (at least) n. The problem could therefore have been stated as finding an

inhabitant of

$$\Sigma\,(X : \mathcal{U}_n)\,.\,\mathsf{is}\text{-}n\text{-}\mathsf{type}(X) \times \neg\mathsf{is}\text{-}(n-1)\text{-}\mathsf{type}(X), \tag{7.2}$$

possibly with an increased universe level.

Remark 7.1.1. Independently, I had thought about the problem (7.1) before and assumed that it would be solvable easily considering higher loops in the universes,

$$\Sigma\,(X : \mathcal{U}_n)\,.\,\mathsf{refl}_X^n = \mathsf{refl}_X^n. \tag{7.3}$$

Here, refl_X^n stands for the "tower" $\mathsf{refl}_{\mathsf{refl}_{\mathsf{refl}_{\cdots}}}$, i.e. the the point $\mathsf{snd}\,(\Omega^n(\mathcal{U}_n, X))$. One key to this was the discovery that I call the *local-global looping principle*, the statement that an $(n+2)$-loop in the universe with basepoint X corresponds to a family of $(n+1)$-loops in X itself (see Main Lemma 7.4.2): this allows to "shift" the current level by one, which seemed to make an inductive argument possible. However, at that time, I did not expect this problem to be open. I did not spell out the proof and therefore did not realise a problem with my approach, even though the gap should have been conspicuous, and I learned only much later from Altenkirch that the problem was harder than I expected.

In 2013 in Princeton, Finster and Lumsdaine were able to extend the argument for $\mathcal{U}_0$ by one or two steps, showing that $\mathcal{U}_1$ is not 1-truncated and $\mathcal{U}_2$ is not a 2-type. Their idea was to construct, as a first step, a "universe" which only contains the type $\mathbf{2}$, namely

$$\mathbf{2}^{(1)} :\equiv \Sigma\,(X : \mathcal{U}_0)\,.\,\|X = \mathbf{2}\|. \tag{7.4}$$

Note that (7.4) does make use of a higher inductive type, namely the propositional truncation, but this usage can then be eliminated with the help of Proposition 2.3.6, or rather the analogous statement for higher universes. However, we think that this makes the expected result weaker than we would want it to be: $X = \mathbf{2}$ is already not an inhabitant of $\mathcal{U}_0$, but only an inhabitant of $\mathcal{U}_1$, and it seems to be unavoidable to eliminate from $\|X = \mathbf{2}\|$ into a type that does not live in $\mathcal{U}_0$. Thus, this approach technically seems to be only suitable to show that $\mathcal{U}_2$ is not 1-truncated while it cannot help with a conclusion about $\mathcal{U}_1$ unless we allow the use of truncations. On the other hand, this probably did not bother Finster and Lumsdaine as the challenge at that time only was to construct a provably not n-truncated type, without requirements on universe levels.

The type $\mathbf{2}^{(1)}$ plays the same role as $\mathcal{U}_0$ itself. However, while $\mathcal{U}_0$ is not tame enough (see Section 7.2.2), it is reasonable that this type works better. The principle here is to take $\mathbf{2}$, which is sufficient to see that $\mathcal{U}_0$ is not a set, and "wrap" it, defining something like the *subuniverse* of $\mathcal{U}_0$ which contains only $\mathbf{2}$. This "wrapping" shifts the non-trivial proof of $\mathbf{2} = \mathbf{2}$ by one level. For the next step, the same construction can be applied again to define

$$\mathbf{2}^{(2)} :\equiv \Sigma\,(X : \mathcal{U}_1)\,.\,\left\|X = \mathbf{2}^{(1)}\right\|, \tag{7.5}$$

and so on, constructing types that one would expect to be non-trivial on a high level. Precisely this "wrapping" strategy was also suggested by Voevodsky in a discussion after a seminar talk in March 2013. However, it was at that time unclear whether the (negative) truncatedness properties could really be proved internally for every n, and Lumsdaine said they were not able to do more than the first few levels.

Another argument for that fact that $\mathcal{U}_1$ is not a 1-type was given by Coquand, using the type of $\mathbb{Z}/2\mathbb{Z}$-sets: a set X, together with an endomorphism on X, and a proof that this endomorphism is self-inverse,

$$\Sigma\,(X : \mathcal{U}_0)\,.\,\Sigma\,(\mathsf{isSet}X)\,.\,\Sigma\,(f : X \to X)\,.\,f \circ f = \mathsf{id}. \tag{7.6}$$

It is not clear how a generalisation of this construction could be used for higher cases, but in fact, the base case of our own construction is more similar to Coquand's suggestion than to Finster's and Lumsdaine's.

When the attempts to construct a type of non-trivial higher structure, or prove that universes have this property, remained inconclusive, the task of construction a "strict" n-type was added to the internal list of open problems of Princeton's special year. Around the same time, Christian Sattler independently came up with yet another proof for the fact that $\mathcal{U}_1$ is not 1-truncated, noticing that the trivial automorphism on the type

$$\Sigma\,(X : \mathcal{U}_0)\,.\,X = X \tag{7.7}$$

can be proved equal to itself in non-trivial ways. Note that (7.7) can be viewed as a simplification of Coquand's suggestion (7.6); by replacing $\Sigma\,(f : X \to X)\,.\,f \circ f = \mathsf{id}$ by $X = X$, the condition $\mathsf{isSet}X$ (which is necessary to make $f \circ f = \mathsf{id}$ propositional) can be dropped. Then, I noticed that (7.7) is just the instantiation $n \equiv 0$ in my previous idea (7.3). However, as said above, the general guess (7.3) does not seem to work for all n. One might think that this is only due to technical difficulties, such as complicated expressions involving too many transports, which are hard to compute manually. However, my coworker Chrstian has even conjectured that (7.3) has a much more fundamental problem. To understand this, one might start by trying to define the *Whitehead product*. Sattler conjectures that, if one can define the Whitehead product, a parametricity argument can show that (7.3) will not work for any odd n. Slightly more details can be found in Sattler's thesis [Sat15, Chapter 3.9, "Further work"].

Close to the end of the univalent foundations special year (March/April 2013), I found the crucial step for the solution which consisted of restricting the type of $(n+1)$-loops in universe $\mathcal{U}_n$ to the type of $(n+1)$-loops in $\mathcal{U}_n$'s "subuniverse" of n-types,

$$\Sigma\,(X : \mathcal{U}_n)\,.\,\Sigma\,(\mathsf{is}\text{-}n\text{-}\mathsf{type}X)\,.\,\mathsf{refl}^n_X = \mathsf{refl}^n_X, \tag{7.8}$$

and presented the solution in the program's regular seminar [Kra13a]. The case $n \equiv 0$ made use of Sattler's argument which can still be applied here. Later, we

changed the notation and wrote

$$\Sigma\,(X : \mathcal{U}_n^n)\,.\,\Omega^{n+1}(\mathcal{U}_n^n, X) \tag{7.9}$$

for the type (7.8).

Even later, I realised that the technology for this solution also greatly helps with the "wrapping" approach of Finster and Lumsdaine discussed above. Indeed, in this thesis, I can present a solution using their approach. However, the result will be slightly weaker as one universe level is lost due to the impredicative encoding.

The last part of this chapter deals with connectedness. While connectedness is usually defined via a higher inductive type, we give a reasonable definition in their absence and show that both definitions are equivalent in their presence. We then construct, given n, a type M_n that is $(n+1)$-truncated, not n-truncated, but n-connected in our sense. In a theory with higher inductive types, this implies that the n-th homotopy group[Uni13, Definition 8.0.1] of M_n is non-trivial, while all other homotopy groups are trivial.

For our Agda formalisation of the results, it is important to take note of a crucial difference between the theory that Agda implements and homotopy type theory which is actually a great advantage here, see Remark 7.1.2. A further difference (this time a disadvantage) is that Agda does not have cumulative universes, making explicit liftings necessary. This could easily have led to poor readability. The workaround that we found was using pointed equivalences instead. By univalence, those are equal to equalities between pointed types, but while equalities between types in different universes do not type-check in Agda, equivalences do.

Remark 7.1.2. (i) First, we want to make a remark on universe polymorphism. As stated above, our main results in this chapter are that $\mathcal{U}_n$ is not an n-type, and that $\mathcal{U}_n^n$, its restriction to n-types, is a strict $(n+1)$-type. Here, n necessarily is an externally fixed constant. If we would quantify over n internally, the statements would become something like

$$\Pi_{n:\mathbb{N}}\ \neg\mathsf{is}\text{-}n\text{-}\mathsf{type}(\mathcal{U}_n) \tag{7.10}$$

and

$$\Pi_{n:\mathbb{N}}\,\Sigma\,(X : \mathcal{U}_n)\,.\,\mathsf{is}\text{-}n\text{-}\mathsf{type}(X) \times \neg\mathsf{is}\text{-}n-1\text{-}\mathsf{type}(X). \tag{7.11}$$

However, homotopy type theory does not regard universe levels as a type that one can eliminate into, and these expressions are therefore not valid types. The only thing we can do is proving that for any given n, the type $\neg\mathsf{is}\text{-}n\text{-}\mathsf{type}(\mathcal{U}_n)$ is inhabited. We do this by an external induction on n, i.e. if we want to prove $\neg\mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(\mathcal{U}_{n+1})$, we assume that we already have a derivation of $\neg\mathsf{is}\text{-}n\text{-}\mathsf{type}(\mathcal{U}_n)$, or of corresponding lemmata. From the point of view of the type theory, occurrences of n are always in canonical form $S(\ldots(S0)\ldots)$, with the length of this expression depending on the current step in the external induction over derivations.

In an implementation of homotopy type theory, we could thus not hope for a formalisation of (7.10) and (7.11). At this point, it is actually an advantage that Agda is *not* an implementation of the type theory that we work in, but only a reasonably close approximation. In particular, Agda *does* allow to eliminate into universe levels. The expressions (7.10) and (7.11) are therefore valid types in Agda and we can construct, and have constructed, inhabitants of them. Applying the Agda term of this Agda type to any canonical natural number n reduces to a derivation of $\neg\mathsf{is}\text{-}n\text{-}\mathsf{type}(\mathcal{U}_n)$ which does not use quantification over universe indices, and is a valid term in a valid type in HoTT.

(ii) Our results (7.1,7.2) are as strong as they can be, in the sense that $\mathcal{U}_n$ can in neither case be replaced by a smaller universe. More concretely, it should be consistent to assume that every type in $\mathcal{U}_n$ is n-truncated (for any given n). We are not aware of any published proof of this fact, but one approach would be to use that the hierarchy $\mathcal{U}_0^0, \mathcal{U}_1^1, \ldots$ is (in an appropriate sense) closed under all type formers, including universe formation by Lemma 7.4.3. This makes it easy to construct a model in which the claimed property holds.

7.2 The First Cases

Let us first discuss several solutions for special cases with low n: we start with the standard argument of $\neg\mathsf{isSet}\mathcal{U}_0$, and demonstrate that the "straightforward" generalisation fails. We then continue with Sattler's argument for $\neg\mathsf{is}\text{-}1\text{-}\mathsf{type}(\mathcal{U}_1)$.

7.2.1 The Well-Known Basic Argument

It is a well-known and immediate consequence of the univalence axiom that the smallest universe is not a set; for example, see the standard reference [Uni13, Example 3.1.9]. The standard proof goes as follows. Suppose $\mathsf{isSet}(U_0)$. Then, by definition of isSet, we have $\mathsf{isProp}(\mathbf{2} = \mathbf{2})$. By univalence, we may replace $\mathbf{2} = \mathbf{2}$ by $\mathbf{2} \simeq \mathbf{2}$. However, there are two distinct automorphisms on $\mathbf{2}$, yielding a contradiction. In formulae:

$$\begin{aligned}
\mathsf{isSet}(\mathcal{U}_0) &\implies \mathsf{isProp}(\mathbf{2} = \mathbf{2}) \\
&\implies \mathsf{isProp}(\mathbf{2} \simeq \mathbf{2}) \\
&\implies (\mathsf{id}_{\mathbf{2}}, \mathsf{e}_{\mathsf{id}}) = (\mathsf{swap}, \mathsf{e}_{\mathsf{swap}}) \\
&\implies \mathsf{id}_{\mathbf{2}} = \mathsf{swap} \qquad (7.12) \\
&\implies \mathsf{id}_{\mathbf{2}}(1_{\mathbf{2}}) = \mathsf{swap}(1_{\mathbf{2}}) \\
&\implies 1_{\mathbf{2}} = 0_{\mathbf{2}} \\
&\implies \bot.
\end{aligned}$$

7.2.2 Failure of the Naive Approach

Intuitively, it may appear that the reason why $\mathcal{U}_0$ is not a set is that an inhabitant of it, namely $\mathbf{2}$, is already not a proposition. However, possibly somewhat surprisingly, this simple idea does not generalise. To make our point clear, let us try to prove $\neg\mathsf{is\text{-}1\text{-}type}(\mathcal{U}_1)$ in a similar way as we have proved $\neg\mathsf{isSet}(\mathcal{U}_0)$ above, choosing two inhabitants of $\mathcal{U}_1$ that seem homotopically complicated enough:

$$\begin{aligned}
\mathsf{is\text{-}1\text{-}type}(\mathcal{U}_1) &\implies \mathsf{isSet}(\mathcal{U}_0 = \mathcal{U}_0)\\
&\quad\ \text{(by univalence)}\\
&\implies \mathsf{isSet}(\mathcal{U}_0 \simeq \mathcal{U}_0)\\
&\quad\ \text{(choose two inhabitants of } \mathcal{U}_0 \simeq \mathcal{U}_0)\\
&\implies \mathsf{isProp}((\mathsf{id}_{\mathcal{U}_0}, \mathsf{e}_{\mathsf{id}}) = (\mathsf{id}_{\mathcal{U}_0}, \mathsf{e}_{\mathsf{id}}))\\
&\implies \ldots ?
\end{aligned} \tag{7.13}$$

In the attempt above, in the very first step, we have to choose two inhabitants of $\mathcal{U}_1$ with sufficiently complicated equality type. We have chosen $\mathcal{U}_0$ as we have already seen before that $\mathcal{U}_0$ is not a set.

The problem is that we seem unable to derive a contradiction from the assumption $\mathsf{isSet}(\mathcal{U}_0 \simeq \mathcal{U}_0)$. In fact, an expected meta-theoretic result is that the identity is the only definable auto-equivalence on $\mathcal{U}_0$. But, if this is the case, we should not even expect that $\mathsf{isContr}(\mathcal{U}_0 \simeq \mathcal{U}_0)$ implies a contradiction. Auto-equivalences on $\mathcal{U}_0$ correspond to families $\Pi_{X:\mathcal{U}_0} X = X$ by strong function extensionality (we will generalise and prove this statement in Main Lemma 7.4.2). This makes the construction of a non-trivial element of $\mathcal{U}_0 \simeq \mathcal{U}_0$ as least as hard as constructing a function $\Pi_{X:\mathcal{U}_0}(X \to X)$, which we do not expect to be possible without further assumptions such as LEM_{-1}. To the best of our knowledge, no one has rigorously proven this form of *parametricity* in the presence of univalence so far, but it is commonly believed to hold. In particular, we conjecture that there is no proof of the statement that a universe is not $(n+1)$-truncated as soon as it contains a type that is not n-truncated, as it was conjectured during the special year in Princeton.

7.2.3 Sattler's Argument

In this subsection, we want to present Sattler's proof that $\mathcal{U}_1$ is not 1-truncated. Effectively this proof will be the induction base for our generalisation; however, we will not reuse the presentation given here as it will easily follow from more general constructions. The approach we take for this special case contains some of the key ideas and could therefore be supportive for understanding the later developments, where all arguments are stated and proved rigorously. While not all steps in the current subsection are spelled out completely, we expect the argument here to be understandable.

The problematic step in the above attempt (7.13) is the first one, where we need to choose something in $\mathcal{U}_1$ and take $\mathcal{U}_0$. We have to choose something better

behaved. We use the type of loops in $\mathcal{U}_0$,

$$L :\equiv \Sigma\,(X : \mathcal{U}_0)\,.\,X = X. \tag{7.14}$$

Showing that the second universe is not a groupoid proceeds as follows:

$$\begin{aligned}
\mathsf{is\text{-}1\text{-}type}(\mathcal{U}_1) &\implies \mathsf{isSet}(L = L)\\
&\quad \text{(by univalence)}\\
&\implies \mathsf{isSet}(L \simeq L)\\
&\quad \text{(choose the identity)}\\
&\implies \mathsf{isProp}((\mathsf{id}_L, \mathsf{e}_{\mathsf{id}}) = (\mathsf{id}_L, \mathsf{e}_{\mathsf{id}}))\\
&\quad \text{(the second component of the equality is trivial)}\\
&\implies \mathsf{isProp}(\mathsf{id}_L = \mathsf{id}_L) \qquad (7.15)\\
&\quad \text{(by strong functional extensionality)}\\
&\implies \mathsf{isProp}(\Pi_{a:L}\, a = a)\\
&\quad \text{(unfold the definition of } L \text{ and curry)}\\
&\implies \mathsf{isProp}(\Pi_{X:\mathcal{U}_0}\Pi_{p:X=X}\ (X,p) = (X,p))\\
&\quad \text{(paths between pairs are pairs of paths)}\\
&\implies \mathsf{isProp}\left(\Pi_{X:\mathcal{U}_0}\Pi_{p:X=X}\Sigma\,(q : X = X)\,.\ q_*(p) = p\right)
\end{aligned}$$

By Lemma 2.2.1, transporting a path along a path can be written as path composition: $q_*(p) = q^{-1} \cdot p \cdot q$. Making this replacement and precomposing with q, we get $\mathsf{isProp}(K)$ where

$$K :\equiv \Pi_{X:\mathcal{U}_0}\Pi_{p:X=X}\Sigma\,(q : X = X)\,.\ p \cdot q = q \cdot p. \tag{7.16}$$

Two inhabitants of K are

$$\alpha :\equiv \lambda X.\lambda p.(\mathsf{refl}_X, u), \tag{7.17}$$

$$\beta :\equiv \lambda X.\lambda p.(p, \mathsf{refl}_{p \cdot p}) \tag{7.18}$$

where u is a proof of $p \cdot \mathsf{refl}_X = \mathsf{refl}_X \cdot p$. Since K is propositional, we may conclude $\alpha =_K \beta$ and consequently $\mathsf{fst}(\alpha(\mathbf{2})) = \mathsf{fst}(\beta(\mathbf{2}))$, which evaluates to

$$\lambda p.\mathsf{refl}_{\mathbf{2}} =_{\mathbf{2}=\mathbf{2}\to\mathbf{2}=\mathbf{2}} \lambda p.p \tag{7.19}$$

and, after replacing $\mathbf{2} = \mathbf{2}$ by $\mathbf{2} \simeq \mathbf{2}$ and applying on $(\mathsf{swap}, \mathsf{e}_{\mathsf{swap}})$, implies the same contradiction as we got above (7.12) in the proof of $\neg\mathsf{isSet}(\mathcal{U}_0)$. □

In the general case, we consider higher loops in higher universes. The core obstacle in translating the above proof is the step where $q_*(p) = p$ is observed to hold for $q :\equiv \mathsf{refl}$ and $q :\equiv p$ by virtue of $q_*(p) = q^{-1} \cdot p \cdot q$. In general, it is not so clear how a uniform presentation of transporting along higher loops would look like. However, as mentioned in the introduction of this chapter (Section 7.1), the approach is likely to fail due to a fundamental problem, even if one was able to resolve the technical obstacles.

Fortunately, we have discovered a simple but effective solution to bypass this problem. The slight modification (7.9) turns out to yield the key for our generalisation.

7.3 Pointed Types

Pointed types are a simple but helpful concept. Their properties can usually easily be formulated in terms of ordinary types. For our presentation of the result on univalent universes we will develop some of their theory explicitly in this section, aiming to provide an elegant way of expressing how Ω interacts with Σ and Π.

7.3.1 Dependent Pairs and Loops

We will first treat the interaction of Σ and Ω. Let us begin by recalling the following definition:

Let (P, p) be a pointed family over some pointed type (A, a). There is an induced type family $\tilde{P}$ over $\Omega(A, a)$, given by $\tilde{P}(q) :\equiv q_*(p) =_{P(a)} p$. The type over the basepoint is $P(\mathsf{refl}_a) \equiv (p = p)$ and therefore trivially inhabited by reflexivity. This allows us to define a fibered version of Ω:

Ⓐ **Definition 7.3.1** ($\tilde{\Omega}$)**.** For a pointed type $\mathfrak{A} \equiv (A, a)$, we define

$$\tilde{\Omega} : \mathrm{Fam}^\bullet_{\mathfrak{A}} \to \mathrm{Fam}^\bullet_{\Omega\mathfrak{A}} \tag{7.20}$$

$$\tilde{\Omega}(P, p) :\equiv \left(\lambda q.q_*(p) =_{P(a)} p\ ,\ \mathsf{refl}_p\right). \tag{7.21}$$

Consequently, Ω and $\tilde{\Omega}$ together form the following endofunction:

$$\langle \Omega, \tilde{\Omega} \rangle : \Sigma\,(\mathfrak{A} : \mathcal{U}_\bullet)\,.\,\mathrm{Fam}^\bullet_{\mathfrak{A}} \to \Sigma\,(\mathfrak{A} : \mathcal{U}_\bullet)\,.\,\mathrm{Fam}^\bullet_{\mathfrak{A}} \tag{7.22}$$

$$\langle \Omega, \tilde{\Omega} \rangle (\mathfrak{A}, \mathfrak{P}) \equiv (\Omega\mathfrak{A}, \tilde{\Omega}\mathfrak{P}) \tag{7.23}$$

Given a pair of a pointed type and a pointed family, it is straightforward to construct a pointed type corresponding to the dependent pair.

Ⓐ **Definition 7.3.2** ($\Sigma^\bullet$)**.** We define the operator $\Sigma^\bullet$ in the following way:

$$\Sigma^\bullet : (\Sigma\,(\mathfrak{A} : \mathcal{U}_\bullet)\,.\,\mathrm{Fam}^\bullet_{\mathfrak{A}}) \to \mathcal{U}_\bullet \tag{7.24}$$

$$\Sigma^\bullet((A, a), (P, p)) :\equiv (\Sigma\,(A)\,.\,P\ ,\ (a, p)) \tag{7.25}$$

We write $\Sigma^\bullet_{\mathfrak{A}}\mathfrak{P}$ synonymously for $\Sigma^\bullet(\mathfrak{A}, \mathfrak{P})$.

We are now ready to formulate precisely how dependent pairs and loop spaces interact:

Ⓐ **Lemma 7.3.3.** *The operators $\Sigma^\bullet$ and Ω commute in the following sense:*

$$\Omega \circ \Sigma^\bullet = \Sigma^\bullet \circ \langle \Omega, \tilde{\Omega} \rangle. \tag{7.26}$$

Proof. Let $\mathfrak{A} \equiv (A, a)$ be a pointed type with a pointed family $\mathfrak{P} \equiv (P, p)$. By function extensionality[1] it is enough to show that both sides of the equation are equal if applied to $(\mathfrak{A}, \mathfrak{P})$. Let us calculate:

$$
\begin{aligned}
& & (\Omega \circ \Sigma^\bullet)(\mathfrak{A}, \mathfrak{P}) & & (7.27)\\
(\text{by definition of } \Sigma^\bullet) & \equiv & \Omega\left(\Sigma\left(a : A\right).\, P(a)\ ,\ (a, p)\right) & & (7.28)\\
(\text{by definition of } \Omega) & \equiv & \left((a, p) = (a, p)\ ,\ \mathsf{refl}_{(a,p)}\right) & & (7.29)\\
(\text{by Lemma 2.2.10}) & = & \left(\Sigma\left(q : a = a\right).\, q_*(p) = p\ ,\ (\mathsf{refl}_a, \mathsf{refl}_p)\right) & & (7.30)\\
(\text{by definition of } \Sigma^\bullet) & \equiv & \Sigma^\bullet_{(a=a, \mathsf{refl}_a)}(\lambda q.q_*(p) =_{P(a)} p, \mathsf{refl}_p) & & (7.31)\\
(\text{by definition of } \Omega \text{ and } \tilde{\Omega}) & \equiv & (\Sigma^\bullet \circ \langle \Omega, \tilde{\Omega} \rangle)(\mathfrak{A}, \mathfrak{P}). & & (7.32)
\end{aligned}
$$

□

7.3.2 Dependent Functions and Loops

The situation is similar, and even simpler, if we want to examine the interaction of Π and Ω. Given a family of pointed types over some (ordinary) type A, there is a straightforward way to construct a pointed type out of the given data corresponding to the dependent function type.

Ⓐ **Definition 7.3.4** ($\Pi^\bullet$)**.** We define the operator $\Pi^\bullet$ by:

$$\Pi^\bullet : \left(\Sigma\left(A : \mathcal{U}\right).\, (A \to \mathcal{U}_\bullet)\right) \to \mathcal{U}_\bullet \tag{7.33}$$

$$\Pi^\bullet(A, \mathfrak{F}) :\equiv \left(\Pi_A \mathsf{fst} \circ \mathfrak{F}\ ,\ \mathsf{snd} \circ \mathfrak{F}\right) \tag{7.34}$$

We use the notations $\Pi^\bullet_{a:A}\mathfrak{F}(a)$ and $\Pi^\bullet_A\mathfrak{F}$ synonymously with $\Pi^\bullet(A, \mathfrak{F})$.

With this at hand, we are ready to prove:

Ⓐ **Lemma 7.3.5.** *Ω and $\Pi^\bullet$ commute in the following sense: given a type A and a family $\mathfrak{F}$ of pointed types over A, we have*

$$\Omega(\Pi^\bullet(A, \mathfrak{F})) = \Pi^\bullet(A, \Omega \circ \mathfrak{F}). \tag{7.35}$$

Proof. Let us do the following calculation:

$$
\begin{aligned}
& & \Omega(\Pi^\bullet(A, \mathfrak{F})) & & (7.36)\\
(\text{by definition of } \Pi^\bullet) & \equiv & \Omega\left(\Pi_A \mathsf{fst} \circ \mathfrak{F}\ ,\ \mathsf{snd} \circ \mathfrak{F}\right) & & (7.37)\\
(\text{by definition of } \Omega) & \equiv & \left(\mathsf{snd} \circ \mathfrak{F} = \mathsf{snd} \circ \mathfrak{F}\ ,\ \mathsf{refl}\right) & & (7.38)\\
(\text{by strong fun. ext.}) & = & \Pi_{a:A}\left(\mathsf{snd}(\mathfrak{F}(a)) = \mathsf{snd}(\mathfrak{F}(a))\ ,\ \lambda a.\mathsf{refl}\right) & & (7.39)\\
(\text{by definition of } \Pi^\bullet) & \equiv & \Pi^\bullet(A, \Omega \circ \mathfrak{F}) & & (7.40)
\end{aligned}
$$

□

[1] Recall that the univalence axiom implies function extensionality, so this is no additional assumption.

7.4 Homotopically Complicated Types

In this section, we will prove two main results of this article: first, in MLTT with the univalence axiom, we can construct a type that *strictly* has truncation level n; and second, the universe $\mathcal{U}_n$ is not n-truncated.

We begin with a lemma that tells us how a truncated Σ-component can be neutralized by Ω.

Ⓐ **Lemma 7.4.1.** *Let n be a natural number. Further, let $\mathfrak{A}$ be a pointed type and $\mathfrak{P}$ be a pointed family over $\mathfrak{A}$ of truncation level $n-2$. Then,*

$$\Omega^n(\Sigma^\bullet_{\mathfrak{A}}\mathfrak{P}) = \Omega^n(\mathfrak{A}). \tag{7.41}$$

Proof. We do induction on n. For the base case $n \equiv 0$, the statement is exactly given by Lemma 2.2.8(i). For the induction case, we have the following chain of equalities:

$$\begin{aligned}
& \Omega^{n+1}(\Sigma^\bullet_{\mathfrak{A}}\mathfrak{P}) && (7.42)\\
\equiv\ & \Omega^n(\Omega(\Sigma^\bullet_{\mathfrak{A}}\mathfrak{P})) && (7.43)\\
=\ & \Omega^n(\Sigma^\bullet_{\Omega\mathfrak{A}}\tilde{\Omega}\mathfrak{P}) && \text{(by Lemma 7.3.3)} \quad (7.44)\\
=\ & \Omega^n(\Omega(\mathfrak{A})) && \text{(by induction hypothesis)} \quad (7.45)\\
\equiv\ & \Omega^{n+1}(\mathfrak{A}) && (7.46)
\end{aligned}$$

For the second to last step, note that if $\mathfrak{P}$ is $n-1$-truncated, then $\tilde{\Omega}\mathfrak{P}$ is $n-2$-truncated. □

We are now ready to prove our local-global looping principle, stating that a loop in the universe is the same as a family of loops in its underlying type:

Ⓐ **Main Lemma 7.4.2** (local-global looping)**.** *Let A be a type and n be a natural number. Then,*

$$\Omega^{n+2}(\mathcal{U}, A) = \Pi^\bullet_{a:A}\Omega^{n+1}(A, a). \tag{7.47}$$

Proof. The proof is again done by a calculation, utilizing most of the theory we have developed so far:

$$\begin{aligned}
& \Omega^{n+2}(\mathcal{U}, A) && (7.48)\\
\equiv\ & \Omega^{n+1}\left(A = A\ ,\ \mathsf{refl}_A\right) && \text{(by definition)} \quad (7.49)\\
=\ & \Omega^{n+1}\left(A \simeq A\ ,\ (\mathsf{id}_A, \mathsf{e}_{\mathsf{id}})\right) && \text{(by univalence)} \quad (7.50)\\
\equiv\ & \Omega^{n+1}\left(\Sigma\,(f : A \to A)\,.\,\mathsf{isequiv}(f)\ ,\ (\mathsf{id}_A, \mathsf{e}_{\mathsf{id}})\right) && \text{(by definition of } \simeq) \quad (7.51)\\
\equiv\ & \Omega^{n+1}(\Sigma^\bullet_{(A\to A, \mathsf{id}_A)}(\mathsf{isequiv}, \mathsf{e}_{\mathsf{id}})) && \text{(by definition of } \Sigma^\bullet) \quad (7.52)\\
=\ & \Omega^{n+1}(A \to A, \mathsf{id}_A) && \text{(by Lemma 7.4.1)} \quad (7.53)\\
\equiv\ & \Omega^{n+1}(\Pi^\bullet_{a:A}(A, a)) && \text{(by definition of } \Pi^\bullet) \quad (7.54)\\
=\ & \Pi^\bullet_{a:A}\Omega^{n+1}(A, a) && \text{(by Lemma 7.3.5)} \quad (7.55)
\end{aligned}$$

□

Recall that we have introduce $\mathcal{U}^n$, that is, the universe $\mathcal{U}$ restricted to n-types, in Definition 2.2.4. The following simple and well-known observation will be useful. We give an explicit proof as it falls out nicely as a consequence of the above lemmata.

Ⓐ **Lemma 7.4.3** ([Uni13, Theorem 7.1.11])**.** *For any $n \geq -2$ and universe $\mathcal{U}$, the type $\mathcal{U}^n$ is $(n+1)$-truncated.*

Proof. For $n \equiv -2$, note that every contractible type is equivalent to the unit type. Assume $n \geq -1$. By Lemma 2.2.17, it suffices to show that, for any $(X, h) : \mathcal{U}^n$, the loop space $\Omega^{n+2}(\mathcal{U}^n, (X, h))$ is contractible. But $(\mathcal{U}^n, (X, h))$ is judgmentally equal to $\Sigma^\bullet_{(\mathcal{U},X)}(\mathsf{is}\text{-}n\text{-}\mathsf{type}, h)$. By Lemma 7.4.1 showing that $\Omega^{n+2}(\mathcal{U}, X)$ is contractible for any n-type X is therefore enough. This is easy to see for $n \equiv -1$. For $n \geq 0$ we apply Main Lemma 7.4.2, requiring us to show that $\Pi^\bullet_{x:X}\Omega^{n+1}(X, x)$ is contractible, and this is the case by Lemma 2.2.17. □

For $n \in \mathbb{N}$, let us write $P_n(X)$ for the type of $(n+1)$-loops that live in the universe $\mathcal{U}_n^n$ and have basepoint X. More precisely, we abbreviate

$$P_n : \mathcal{U}_n^n \to \mathcal{U}_{\bullet_{n+1}} \tag{7.56}$$
$$P_n(X) :\equiv \Omega^{n+1}(\mathcal{U}_n^n, X). \tag{7.57}$$

Homotopically, these loops $P_n(X)$ are rather tame:

Ⓐ **Corollary 7.4.4** (of Lemma 7.4.3)**.** *P_n is a family of sets, that is,*

$$\Pi_{\mathcal{U}_n^n} \mathsf{isSet} \circ P_n. \tag{7.58}$$

□

An $(n+1)$-loop consists of a basepoint X and the actual loop around X. For all $n \in \mathbb{N}$, the type of $(n+1)$*-loops in universe* $\mathcal{U}_n^n$ is therefore given by

$$\mathsf{Loop}_n : \mathcal{U}_{n+1} \tag{7.59}$$
$$\mathsf{Loop}_n :\equiv \Sigma\,(\mathcal{U}_n^n)\,.\,\mathsf{fst} \circ P_n. \tag{7.60}$$

We further choose to define $\mathsf{Loop}_{-1} :\equiv \mathbf{2}$ in the lowest universe $\mathcal{U}_0$, which will allow us to treat all universes uniformly.

This type is also fairly tame homotopically:

Ⓐ **Lemma 7.4.5.** *For all natural numbers n, the type Loop_{n-1} is n-truncated, that is, we can construct*

$$h_n : \mathsf{is}\text{-}n\text{-}\mathsf{type}(\mathsf{Loop}_{n-1}). \tag{7.61}$$

Proof. The claim is fulfilled for $n \equiv 0$ by the standard argument that we discussed in Section 7.2, so let us assume $n \geq 1$. By Lemma 2.2.3, it is enough to examine the two parts of the dependent pair separately. The required property for the first part is given by Lemma 7.4.3. Further, the second component is a family of sets by Corollary 7.4.4, which suffices by Lemma 2.2.2. □

For a pointed type (A, a), we say that an element $b : A$ is *trivial* if it is equal to the basepoint, i.e. if we have a proof of $a = b$. We mark the following result as "formalised" Ⓐ. However, note that the formalised version is slightly weaker: instead of "has a non-trivial inhabitant", we show only "not all inhabitants are trivial". The latter version has turned out to be easier to implement while being totally sufficient for our later results, which can all be found at full strength in the formalisation.

Ⓐ **Lemma 7.4.6.** *For all $n \geq 0$, the type $\Omega^{n+1}(\mathcal{U}_n\ ,\ \mathsf{Loop}_{n-1})$ has a non-trivial inhabitant. The same is true for $\Omega^{n+1}(\mathcal{U}_n^n\ ,\ (\mathsf{Loop}_{n-1}, h_n))$.*

Proof. Observe that the pointed type $(\mathcal{U}_n^n\ ,\ (\mathsf{Loop}_{n-1}, h_n))$ can be written as (and is judgmentally equal to) the expression $\Sigma^\bullet_{(\mathcal{U}_n, \mathsf{Loop}_{n-1})}(\mathsf{is}\text{-}n\text{-}\mathsf{type}, h_n)$. As the predicate $\mathsf{is}\text{-}n\text{-}\mathsf{type}$ is propositional, Lemma 7.4.1 implies the equivalence of the two loop spaces of this lemma, and we may restrict ourselves to showing that the claim holds for $\Omega^{n+1}(\mathcal{U}_n\ ,\ \mathsf{Loop}_{n-1})$.

We do induction on n. For $n \equiv 0$, we have to provide a non-trivial inhabitant of $2 = 2$. This is swap, just as in Section 7.2.1.

Assume $n \equiv m + 1$ and calculate:

$$\Omega^{m+2}(\mathcal{U}_{m+1}, \mathsf{Loop}_m) \tag{7.62}$$

$$= \Pi^\bullet_{(X,q):\mathsf{Loop}_m} \Omega^{m+1}(\mathsf{Loop}_m, (X, q)) \quad \text{("local-global", 7.4.2)} \tag{7.63}$$

$$\equiv \Pi^\bullet_{(X,q):\mathsf{Loop}_m} \Omega^{m+1}(\Sigma^\bullet_{(\mathcal{U}_m^m, X)}(\mathsf{fst} \circ P_m, q)) \quad \text{(by definition of } \Sigma^\bullet\text{)} \tag{7.64}$$

$$\equiv \Pi^\bullet_{(X,q):\mathsf{Loop}_m} \Sigma^\bullet_{\Omega^{m+1}(\mathcal{U}_m^m, X)} \tilde{\Omega}^{m+1}(\mathsf{fst} \circ P_m, q) \quad \text{("}\Omega\text{, }\Sigma^\bullet\text{ commute", 7.3.3)} \tag{7.65}$$

The underlying type of this last pointed type has the following inhabitant:

$$\xi :\equiv \lambda(X, q).(q, d_q) \tag{7.66}$$

where d_q is defined as follows:

- for $m \equiv 0$, the type of d_q is $q_*(q) = q$, which is inhabited by Lemma 2.2.1 and the fact that equality carries a groupoid structure. Note that this case corresponds to the special case we already handled, and that we are not able to obviate this additional coherence condition here.
- for $m \geq 1$, the type of d_q is contractible by Corollary 7.4.4 and the definition of $\tilde{\Omega}$, providing a canonical choice for d_q.

Once again, let us view Loop_{m-1}, together with h_m, as an inhabitant of $\mathcal{U}_m^m$. Note that $P_m(\mathsf{Loop}_{m-1}, h_m)$ is, by definition, judgmentally equal to

$$\Omega^{m+1}(\mathcal{U}_m^m\ ,\ (\mathsf{Loop}_{m-1}, h_m)). \tag{7.67}$$

By the induction hypothesis, we can construct a non-trivial inhabitant $\tilde{q}$ of the underlying type, so that we have

$$((\mathsf{Loop}_{m-1}, h_m)\ ,\ \tilde{q}) : \mathsf{Loop}_m. \tag{7.68}$$

If ξ was trivial, the term $\mathsf{fst}(\xi(X,q)) \equiv q$ would be trivial in $P_m(X)$ for any $(X,q) : \mathsf{Loop}_m$. But this is invalidated by (7.68). □

This allows us to prove:

Ⓐ **Theorem 7.4.7.** *In Martin-Löf type theory with a hierarchy of univalent universes $\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \ldots$, the universe $\mathcal{U}_n$ is not an n-type. That is, for any natural number n, the type*

$$\neg\mathsf{is}\text{-}n\text{-}\mathsf{type}\,(\mathcal{U}_n) \tag{7.69}$$

is inhabited.

Proof. If $\mathcal{U}_n$ was an n-type, then $\Omega^{n+1}\,(\mathcal{U}_n\,,\,\mathsf{Loop}_{n-1})$ would be propositional, contradicting Lemma 7.4.6. □

At the same time, we have solved the question of constructing a "strict" n-type.

Ⓐ **Theorem 7.4.8.** *For a given $n \geq -2$, there is (in the settings of Theorem 7.4.7) a type that is an $(n+1)$-type but not an n-type. In particular, for $n \geq -1$, the type Loop_n has this property. Further, for $n \geq 0$, the universe of n-types at level n, namely $\mathcal{U}_n^n$, is such a strict $(n+1)$-type.*

Proof. For $n \equiv -2$, the empty type proves the statement. The claim for $\mathcal{U}_n^n$ follows in the same way as Theorem 7.4.7, combined with Lemma 7.4.3. $\mathsf{Loop}_{-1} \equiv \mathbf{2}$ is clearly strictly a set. For $n \geq 0$, Lemma 7.4.5 shows that Loop_n is $(n+1)$-truncated. To see that it is not n-truncated, observe that the first component is $\mathcal{U}_n^n$ and therefore not n-truncated while the second component is always inhabited. □

7.5 A Solution with the "Wrapping" Approach

The tools that we have developed allow us to solve the problem of constructing a strict $(n+1)$-type in at least one additional way. It was originally the idea of Finster and Lumsdaine, but also suggested by Voevodsky, that the question could possibly be attacked in this way (see Section 7.1). We start by freely using propositional truncation, even though this is "against the rules" of the problem. Afterwards, we observe how we can replace truncations by appropriate impredicative encodings. This turns out to work, but this is not a priori clear: one needs to be careful to choose the correct universe levels at each point, and it could have happened that the dependencies are cyclic in a way that makes the construction impossible. Maybe somewhat surprisingly, we only loose one universe level (that is, we need one more universe for a given n) compared to the above result, and the consequence is a probably less elegant statement than Theorem 7.4.8. An advantage might be that we can give the highest non-trivial loop space explicitly: by construction, it is equivalent to $\mathbf{2}$. It would be possible to formalise the results of this section, but we have chosen to omit it in our Agda implementation.

We start by defining the "wrapping" (see Section 7.1) explicitly:

Definition 7.5.1. For all $n \geq 0$, we define the type $\mathbf{2}^{(n)}$ in the universe $\mathcal{U}_n$ by

$$\mathbf{2}^{(0)} :\equiv \mathbf{2} \tag{7.70}$$
$$\mathbf{2}^{(n+1)} :\equiv \Sigma\,(X : \mathcal{U}_n)\,.\,\left\|X = \mathbf{2}^{(n)}\right\|. \tag{7.71}$$

We may regard every $\mathbf{2}^{(n)}$ as pointed type, where the point ${1_\mathbf{2}}^{(n)}$ is

$${1_\mathbf{2}}^{(0)} :\equiv 1_\mathbf{2} \tag{7.72}$$
$${1_\mathbf{2}}^{(n+1)} :\equiv (\mathbf{2}^{(n)}, |\mathsf{refl}|). \tag{7.73}$$

This definition allows us to apply the tools we have developed before and prove:

Lemma 7.5.2. *For all n and $Z : \mathbf{2}^{(n)}$, there is a pointed equivalence from the pointed type $\Omega^n(\mathbf{2}^{(n)}, Z)$ to the pointed type $(\mathbf{2}, 1_\mathbf{2})$.*

Proof. The crucial point is that "being pointed-equivalent to $(\mathbf{2}, 1_\mathbf{2})$" is a propositional property. We define it (for any universe $\mathcal{U}$) as

$$\mathsf{is}\mathbf{2}^\bullet : \mathcal{U}_\bullet \to \mathcal{U} \tag{7.74}$$
$$\mathsf{is}\mathbf{2}^\bullet(Y, y_0) :\equiv \Sigma\,(f : Y \to \mathbf{2})\,.\,\mathsf{isequiv}(f) \times (f(y_0) = 1_\mathbf{2}). \tag{7.75}$$

We do the proof by induction on n. For $n \equiv 0$, the claim holds by definition. For $n \equiv 1$, assume $Z \equiv (Y, q)$ with $Y : \mathcal{U}_0$ and $q : \|Y = \mathbf{2}\|$. We then have

$$\Omega^1(\mathbf{2}^{(1)}, Z) \tag{7.76}$$
$$\text{(by definition)} \quad \equiv \quad \Omega\left(\Sigma\,(X : \mathcal{U}_0)\,.\,\|X = \mathbf{2}\|, (Y, q)\right) \tag{7.77}$$
$$\text{(by Lemma 7.4.1)} \quad = \quad \Omega(\mathcal{U}_0, Y) \tag{7.78}$$
$$\equiv \quad Y = Y. \tag{7.79}$$

We thus need to show $\mathsf{is}\mathbf{2}^\bullet(Y = Y, \mathsf{refl}_Y)$. This would clearly be the case if we had a proof of $Y = \mathbf{2}$, and as the goal is a proposition, q suffices.

For $(n+2)$, assume $Z \equiv (X, p)$ with $X : \mathcal{U}_{n+1}$ and $p : \|X = \mathbf{2}^{(n+1)}\|$. Then,

$$\Omega^{n+2}(\mathbf{2}^{(n+2)}, Z) \tag{7.80}$$
$$\text{(by definition)} \quad \equiv \quad \Omega^{n+2}\left(\Sigma\,(X : \mathcal{U}_{n+1})\,.\,\left\|X = \mathbf{2}^{(n+1)}\right\|, (X, p)\right) \tag{7.81}$$
$$\text{(by Lemma 7.4.1)} \quad = \quad \Omega^{n+2}(\mathcal{U}_{n+1}, X) \tag{7.82}$$
$$\text{(by Main Lemma 7.4.2)} \quad = \quad \Pi^\bullet_{x:X}\Omega^{n+1}(X, x). \tag{7.83}$$

As our goal is propositional, we may assume $X = \mathbf{2}^{(n+1)}$ instead of $\|X = \mathbf{2}^{(n+1)}\|$, and the above type becomes

$$\Pi^\bullet_{x:\mathbf{2}^{(n+1)}}\Omega^{n+1}(\mathbf{2}^{(n+1)}, x). \tag{7.84}$$

The type $\Omega^{n+1}(\mathbf{2}^{(n+1)}, x)$ is pointed-equivalent to $(\mathbf{2}, 1_\mathbf{2})$ by the induction hypothesis. Consequently, the type (7.84) is equivalent to $\Pi^\bullet_{x:\mathbf{2}^{(n+1)}}(\mathbf{2}, 1_\mathbf{2})$. The required statement now follows from the fact that $\mathbf{2}^{(n+1)}$ is *connected* (or *0-connected*, see [Uni13, Chapter 7.5]). □

Using Lemma 2.2.18, we see that Lemma 7.5.2 has an immediate consequence:

Corollary 7.5.3. *For any n, the type $\mathbf{2}^{(n)}$ is a strict n-type.* □

Further, it is easy to see that the projection $\mathbf{2}^{(n+1)} \to \mathcal{U}_n$ is an embedding. This implies that $\mathcal{U}_n$ is not n-truncated, as $\mathbf{2}^{(n+1)}$ would otherwise be n-truncated by [Uni13, Theorem 7.1.6], contradicting Corollary 7.5.3.

Let us come back to the original question: The open problem of constructing a strict n-type without higher inductive types. Corollary 7.5.3 does this, but with the use of propositional truncations. The idea behind the approach was that the need of the truncation operator could be removed afterwards, using the impredicative encoding (Proposition 2.3.6). We have to be very careful to choose the correct universe levels. In the proof of Lemma 7.5.2, in the case $n \equiv 1$, the type $Y = Y$ is in $\mathcal{U}_1$, and so it $\mathsf{is2^\bullet}(Y = Y, \mathsf{refl})$. The component $q : \|Y = \mathbf{2}^{(0)}\|$ cannot be encoded as $\Pi_{P:\mathcal{U}_0} \mathsf{isProp}(P) \to ((Y = \mathbf{2}^{(0)}) \to P) \to P$ as we could not choose P to be $\mathsf{is2^\bullet}(Y = Y, \mathsf{refl})$.

A working definition is:

$$\underline{\mathbf{2}^{(0)}} :\equiv \mathbf{2} \qquad : \mathcal{U}_0 \tag{7.85}$$

$$\underline{\mathbf{2}^{(1)}} :\equiv \Sigma\,(X : \mathcal{U}_0)\,.\,\Pi_{P:\mathcal{U}_1^{-1}}((X = \underline{\mathbf{2}^{(0)}}) \to P) \to P \qquad : \mathcal{U}_2 \tag{7.86}$$

$$\underline{\mathbf{2}^{(n+1)}} :\equiv \Sigma\,(X : \mathcal{U}_{n+1})\,.\,\Pi_{P:\mathcal{U}_{n+1}^{-1}}((X = \underline{\mathbf{2}^{(n)}}) \to P) \to P \qquad : \mathcal{U}_{n+2}. \tag{7.87}$$

Note that (7.86) is not an instance of (7.87) exactly because the first component of the Σ-type (X) is in $\mathcal{U}_0$, not in $\mathcal{U}_1$. This is necessary, as the universe levels would not work out otherwise. It is reflected in the proof of Lemma 7.5.2, where we need to treat the case $n \equiv 1$ separately as well, due to the fact that we cannot "save" one universe level by applying "local-global". This is made up for by (7.85), which is in $\mathcal{U}_0$ instead of only in $\mathcal{U}_1$.

With the above definition, the proof of Lemma 7.5.2 works for $\underline{\mathbf{2}^{(n)}}$. The proof with the impredicative encoding is also fairly tricky: For example, when we eliminate the truncation to show $\mathsf{is2^\bullet}\,(\Omega^{n+2}(\underline{\mathbf{2}^{(n+2)}}, Z))$, this "real" goal is in universe $\mathcal{U}_{n+3}$. From what we know, we are not able to eliminate the (encoded) truncation if the goal is in $\mathcal{U}_{n+3}$; only because that goal happens to be equivalent to a type in $\mathcal{U}_{n+2}$, namely $\mathsf{is2^\bullet}(\Omega^{n+2}(\mathcal{U}_{n+1}, X))$, the proof works. Every time when the eliminator for the truncation is applied, the levels in the definition of $\underline{\mathbf{2}^{(n)}}$ are just the correct ones. In the last step, we need to show that $(\underline{\mathbf{2}^{(n+1)}} \to \mathbf{2})$ is pointed-equivalent to $(\mathbf{2}, 1_{\mathbf{2}})$. For this, it suffices to show that any function $(\underline{\mathbf{2}^{(n+1)}} \to \mathbf{2})$ is weakly constant, so that we only need to apply the "eliminator" to show an equality in $\mathbf{2}$ at the lowest universe level.

However, note that $\underline{\mathbf{2}^{(n+1)}}$ is not in $\mathcal{U}_{n+1}$, but only in $\mathcal{U}_{n+2}$. We thus have here constructed a strict $(n+1)$-type in Universe $\mathcal{U}_{n+2}$. Compared to our result Theorem 7.4.8 where we had a strict $(n+1)$-type in universe $\mathcal{U}_{n+1}$, we have lost one universe level.

One advantage compared to the previous result (Theorem 7.4.8) is that we know the highest non-trivial loop space (and thereby also the highest non-trivial homotopy group) explicitly. This seems nice, so let us record it:

Theorem 7.5.4. *In Martin-Löf type theory with a hierarchy of univalent universes, the type* $\underline{\mathbf{2}^{(n+1)}}$ *(in universe* $\mathcal{U}_{n+2}$*) is a strict* $(n+1)$*-type, and its* n*-th loop space is equivalent to* $\mathbf{2}$*.* □

7.6 Connectedness

So far, we have shown how to construct types that have some truncation level strictly. If we had worked in a theory with appropriate truncation operations, this could have been made precise by saying that the n-th homotopy group (at some basepoint) is non-trivial, while all higher groups are trivial. Our goal for this section is to control the levels below n as well, still without using higher inductive types. However, this property is tricky to express in type theory: if we directly state that a type is trivial at some dimension, it immediately implies that it is trivial at all higher dimensions as well (see the introduction in Section 1.2). The way HoTT deals with this problem is the following: in order to express that a type is n-connected, the type is first "artificially" made trivial above dimension n, and then required to be contractible. Unfortunately, this requires truncation operators, which we count as one kind of higher inductive types.

7.6.1 Truncations via Universal Properties

As higher inductive types allow us to add paths at an arbitrarily high level to a type, it is not surprising that they can be used to construct types that are not n-truncated for a given n. A canonical candidate for a HIT that is not an $(n-1)$-type is the sphere $\mathbb{S}^n$, which can be generated with one point-constructor base and one path-constructor loop that gives an inhabitant of $\Omega^n(\mathbb{S}^n, \mathsf{base})$. Unfortunately, even the seemingly simple statement that loop is non-trivial is not amenable to an instant argument. While $\mathbb{S}^n$ has $\mathbb{Z}$ as n-th homotopy group, which immediately implies that it is not an $(n-1)$-type, calculating it in HoTT for example via the long exact sequence requires some effort [Lic13].

On the other hand, the construction of Loop_n that we present in this article can be understood as a way to use spheres even if the theory does not support HITs. Recall that our type Loop_n was (after unfolding the definition of P_n) defined as

$$\mathsf{Loop}_n :\equiv \Sigma\,(X : \mathcal{U}_n^n)\,.\,\mathsf{fst}(\Omega^{n+1}(\mathcal{U}_n^n, X)). \tag{7.88}$$

If HITs are available, Loop_n is equivalent to the function type

$$\mathbb{S}^n \to \mathcal{U}_n^n. \tag{7.89}$$

Even if we do not have $\mathbb{S}^n$ available in the theory, we can thus still talk about how the sphere could be mapped into another type (this holds true for any non-recursive HIT).

If we have truncation operators available, connectedness can be defined as follows:

Ⓐ **Definition 7.6.1** ([Uni13, special case of Definition 7.5.1])**.** In a theory with truncations, the property of being *n-connected* can then be defined as

$$\mathsf{is}\text{-}n\text{-}\mathsf{connected}(A) :\equiv \mathsf{isContr}(\|A\|_n). \tag{7.90}$$

We will define a notion of connectedness that does not require truncations to be part of the theory.

From now on, let us denote $\mathsf{MLTT}^{\mathsf{UA}}$ the setting that we have considered so far (MLTT with univalence). We say that a statement holds in MLTT if it further does not require univalence. If instead the proof of a lemma also needs truncations in the sense of Definition 2.3.8, we say that it holds in $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$.

We will use the fact that truncations can be characterized up to homotopy by their universal properties in MLTT. This allows us to formulate the statement that a type is n-connected* in MLTT, with the name being justified by the fact that we can prove $\mathsf{is}\text{-}n\text{-}\mathsf{connected} = \mathsf{is}\text{-}n\text{-}\mathsf{connected}^\star$ in $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$.

Given a type A with an inhabitant a, we will (in MLTT) construct the n-connected version of A with basepoint a. In $\mathsf{MLTT}^{\mathsf{UA}}$, that type has the same loop spaces as A above dimension n and is n-connected*. For $n \equiv -1$, this corresponds to constructing the connected component of a in the ordinary topological sense.

The construction of this "n-connected version" can also be done fairly easily in $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$. It corresponds to

$$\Sigma\,(b : A)\,.\,\|b = a\|_{n-1}. \tag{7.91}$$

It may be hard to appreciate the contents of this section for the reader who is not interested in the formalisation. The mathematical construction is somewhat straightforward; the actual challenge is the formalisation and the fiddling with details. Getting all the universe levels right seems to be the hardest part from the mathematical point of view. As one can see in the electronic appendix, the formalisation of this section is about twice the length of the formalisation of the rest of this chapter.

We first want to specify what it means in MLTT to have the universal property of a truncation.

Ⓐ **Definition 7.6.2.** Let $\mathcal{U}$ and $\mathcal{V}$ be universes, A be a type in $\mathcal{U}$, and $n \geq -2$ be a number. Let X be an n-type in $\mathcal{U}$ and c a function from A to X. We say that (X, c) (or just c) has the *universal property of the n-truncation of A with respect to* $\mathcal{V}$ if, for any n-type Y in $\mathcal{V}$, the function types $X \to Y$ and $A \to Y$ are equivalent, and the equivalence is given by composition with c:

$$\mathsf{up}^{A:\mathcal{U}}_{\mathcal{V}^n}(c) :\equiv \Pi_{(Y,k):\mathcal{V}^n}\,\mathsf{isequiv}(\lambda f : X \to Y.\, f \circ c). \tag{7.92}$$

Remark 7.6.3. Let us say that a *truncation algebra* of degree n over A is an n-type Y together with a map $A \to Y$. It is to be expected that such truncation algebras bear a canonical weak $(\omega, 1)$-categorical structure, where a morphism between truncation algebras (Y, d) and (Z, e) consists of a map $f : Y \to Z$ and a proof $f \circ d = e$. Let us ignore hierarchy issues for the moment. Saying that a truncation algebra (X, c) has the universal property of the n-truncation of A is then equivalent to stating that (X, c) is a homotopy initial object, i.e. classes of morphisms to arbitrary truncation algebras are contractible.

If we had an internal representation of the 2-cells of the preliminary $(\omega, 1)$-categorical structure alluded to above, this should yield an alternative approach for proving Lemmata 7.6.6 and 7.6.7 which is more in line with work by Awodey, Gambino, and Sojakova [AGS12] and Sojakova [Soj15]. However, the construction of the required weak $(\omega, 1)$-categorical structure has yet to be given. We expect that, once we have this construction, all the usual logical equivalences between universal properties, homotopy initiality, recursion with propositional computation unique up to homotopy, and induction with propositional computation admit a unifying treatment that is more abstract than the current approach of rather explicit definitions and hands-on path computation.

Let us now define a type with the property that any of its inhabitant can serve as an n-truncation. The crucial point will consist of perspicaciously inserting the assumption that certain truncations exist deep into the connectedness construction of types, taking care not to change their expected properties.

Ⓐ **Definition 7.6.4.** Given universes $\mathcal{U}$ and $\mathcal{V}$, a type A in $\mathcal{U}$ and $n \geq -2$ in MLTT, define the *type of n-truncations* of A to be the type of maps $c : A \to X$ for some n-type $X : \mathcal{U}$ which satisfy the universal property:

$$\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A) :\equiv \Sigma\left((X, h) : \mathcal{U}^n\right). \Sigma\left(c : A \to X\right). \ \mathsf{up}^{A,\mathcal{U}}_{\mathcal{V}^n}(c). \tag{7.93}$$

Note that this type is not in universe $\mathcal{U}$ or $\mathcal{V}$, but it inhabits every universe that $\mathcal{U}$ and $\mathcal{V}$ both inhabit. Given $t \equiv (X, h, c, u) : \mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$, we write $\mathsf{type}(t)$ for the component X (i.e. $\mathsf{type} \equiv \mathsf{fst}$) and $\mathsf{cons}(t)$ for the component c (i.e. $\mathsf{cons} \equiv \mathsf{fst} \circ \mathsf{snd}$).

Remark 7.6.5. In $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$, the type $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ is naturally inhabited by $\|A\|_n$ (Lemma 2.3.10). For MLTT or even $\mathsf{MLTT}^{\mathsf{UA}}$, which we consider here, we strongly believe that an inhabitant of $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ can, in general, not be constructed: the n-truncation is not *definable*.

Our intention is to use an (assumed) inhabitant of $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ in the same way as $\|A\|_n$ could be used if it was part of the theory. While this turns out to be possible, there are a couple of obstacles:

First, the only assumption we make is that $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ includes the encoding of a universal property. The truncation $\|A\|_n$ of homotopy type theory has this universal property with respect to n-types of any universe. Being unable to polymorphically quantify over all universes, we have to restrict ourselves to a fixed

elimination universe $\mathcal{V}$. We need to be careful though: it is not sufficient for our purposes to require the universal property with respect to all n-types in the same universe as A, as this would prevent us from performing *large elimination*. For that reason, we require the universal property for all n-types that live in some fixed universe $\mathcal{V}$, which for most constructions will have to be larger than $\mathcal{U}$.

Second, the truncation $\|A\|_n$, defined as a HIT, has important judgmental computation rules. We have seen in Chapter 6 that such rules have the potential to make the theory stronger. It is therefore not a priori clear that the universal property that we have at hand suffices for everything we want to do. However, even disregarding that, the judgmental computational rules offer in many cases huge simplifications, and in this section, this does affect us – the Agda formalisation of the proof is considerably more tedious than the analogous proof using $\|A\|_n$ would be.

Third, note that in the definition of $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$, we only ask for a non-dependent universal property, while (in $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$) $\|A\|_n$ has the dependent version of this property. While we could encode the dependent universal property in the definition as well, we do not need to: as we will see, the non-dependent universal property implies the dependent one.

7.6.2 Consequences of the Universal Property

Our next goal is to prove a couple of properties of $\mathsf{up}^{A:\mathcal{U}}_{\mathcal{V}^n}$ and $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}$ in MLTT. For the lemmata in this subsection, let $\mathcal{U}$ and $\mathcal{V}$ be universes such that $\mathcal{V}$ is at least as large as $\mathcal{U}$, i.e. every type in $\mathcal{U}$ is also of type $\mathcal{V}$. Further, let $A : \mathcal{U}$ be a type and $n \geq -2$ a number.

Ⓐ **Lemma 7.6.6.** *We say that a map $e : A \to X$ for some n-type $X : \mathcal{U}$ has the* dependent universal property *if the type*

$$\mathsf{dup}^{A:\mathcal{U}}_{\mathcal{V}^n}(e) :\equiv \Pi_{\langle Y,h\rangle : X \to \mathcal{V}^n} \mathsf{isequiv}(\lambda f : \Pi_X Y. f \circ e). \tag{7.94}$$

is inhabited. Then, e has the universal property if and only if it has the dependent universal property,

$$\mathsf{up}^{A:\mathcal{U}}_{\mathcal{V}^n}(e) \;\longleftrightarrow\; \mathsf{dup}^{A:\mathcal{U}}_{\mathcal{V}^n}(e). \tag{7.95}$$

Proof. The direction "if" is trivial. For the other direction, we need the following statement: For a function $u : C \to D$ between types C, D and $E : D \to \mathcal{U}'$ a type family in any universe $\mathcal{U}'$, dependent functions from $c : C$ to $E(u(c))$ are equivalent to factorisations of u through $\Sigma\,(D)\,.\,E \to D$:

$$\Sigma\,(s : C \to \Sigma\,(D)\,.\,E)\,.\;\mathsf{fst} \circ s = u \;\;\simeq\;\; \Pi_C\, E \circ u. \tag{7.96}$$

Note that for $C \equiv D$ and $u \equiv \mathsf{id}$ this just says that $\Pi_C E$ is the type of sections of the first projection. The equivalence (7.96) could be slightly strengthened by

making D (and E) dependent on C, but we do not need this generality here. An easy way to prove (7.96) is combining a couple of equivalences. First,

$$\Sigma\,(s : C \to \Sigma\,(D)\,.\,E)\,.\;\mathsf{fst} \circ s = u \;\simeq\; \Pi_{c:C}\,\Sigma\,(p : \Sigma\,(D)\,.\,E)\,.\;\mathsf{fst}(p) = u(c) \tag{7.97}$$

holds by strong function extensionality combined with the distributivity law for Π and Σ (Lemma 2.2.12). Further, we have

$$\begin{aligned}\Sigma\,(p : \Sigma\,(D)\,.\,E)\,.\;\mathsf{fst}(p) = u(c) &\simeq \Sigma\,(d : D)\,.\;E(d) \times (d = u(c)) \\ &\simeq \Sigma\,(\Sigma\,(d : D)\,.\,d = u(c))\,.\;E \circ \mathsf{fst} \\ &\simeq E(u(c))\end{aligned} \tag{7.98}$$

The third step follows from contractibility of $\Sigma\,(d : D)\,.\,d = d_0$ and Lemma 2.2.9(i). The equivalences (7.97) and (7.98) together prove (7.96).

To prove the "only if" direction of the lemma, assume $e : A \to X$ satisfies the non-dependent universal property. Let $\langle Y, h\rangle : X \to \mathcal{V}^n$ be a family of n-types over X. Set $\tilde{Y} :\equiv \Sigma\,(X)\,.\,Y$, which is an n-type in $\mathcal{V}$. By $\mathsf{up}^{A:\mathcal{U}}_{\mathcal{V}^n}(e)$, the map $\lambda f.f \circ e$ induces an equivalence

$$(X \to \tilde{Y}) \simeq (A \to \tilde{Y}). \tag{7.99}$$

Fix $s : X \to \tilde{Y}$. We then have $\mathsf{fst} \circ s : X \to X$. By the assumed universal property, composition with e is an equivalence $(X \to X) \simeq (A \to X)$. As equivalences preserve path spaces (Lemma 2.2.11), we have

$$\Sigma\left(s : X \to \tilde{Y}\right).\,\mathsf{fst} \circ s = \mathsf{id}_X \;\simeq\; \Sigma\left(s : X \to \tilde{Y}\right).\,\mathsf{fst} \circ s \circ e = e. \tag{7.100}$$

In general, for any types C, D, a type family $P : D \to \mathcal{U}'$ and an equivalence $h : C \to D$, we have that $\Sigma\,(C)\,.\,P \circ h \simeq \Sigma\,(D)\,.\,P$. Applying this rule with $C \equiv X \to \tilde{Y}$, $D \equiv A \to \tilde{Y}$, $P(s) \equiv \mathsf{fst} \circ s = e$ and the equivalence (7.99) for h, we get

$$\Sigma\left(s : X \to \tilde{Y}\right).\,\mathsf{fst} \circ s \circ e = e \;\simeq\; \Sigma\left(s : A \to \tilde{Y}\right).\,\mathsf{fst} \circ s = e. \tag{7.101}$$

Finally, we are ready to prove $\Pi_X Y \simeq \Pi_A Y \circ e$. We start with $\Pi_X Y$ and apply (7.96) with $u \equiv \mathsf{id}_X$ to transform it into $\Sigma\left(s : X \to \tilde{Y}\right).\,\mathsf{fst} \circ s = \mathsf{id}_X$. Using first (7.100) and then (7.101), this type is equivalent to $\Sigma\left(s : A \to \tilde{Y}\right).\,\mathsf{fst} \circ s = e$. Equivalence (7.96), this time with $u \equiv e$, transforms that type into $\Pi_A Y \circ e$ as required.

Going through the proof again and only checking what the function part of the constructed equivalence does, it is easy to see that $f : \Pi_X Y$ is mapped to $f \circ e : \Pi_A Y \circ e$, proving $\mathsf{dup}^{A:\mathcal{U}}_{\mathcal{V}^n}(e)$. □

Another interesting point is that the type $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ can be shown to be propositional already in $\mathsf{MLTT}^{\mathsf{UA}}$, implying that it is contractible in $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$ by Remark 7.6.5.

Ⓐ **Lemma 7.6.7.** *In* $\mathsf{MLTT}^{\mathsf{UA}}$*, the type* $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ *is propositional.*

Proof. This is a consequence of the fact that $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ is characterized via a universal property. Given inhabitants $A \xrightarrow{e_1} X_1$ and $A \xrightarrow{e_2} X_2$, we need to show that they are equal. Note that the universal property itself is propositional, and thus, we do not have to construct a (dependent) path between the two witnesses of this property. Such an equality proof therefore just corresponds to an equivalence $e : X_1 \simeq X_2$ and a proof of $e \circ e_1 = e_2$. The universal property of e_1 tells us that $X_1 \to X_2$ and $A \to X_2$ are equivalent. As the latter type is inhabited by e_2, we find an inhabitant e of the former, and we get the property $e \circ e_1 = e_2$ for free. The construction of the inverse of e is completely analogous, and the proof that their composition is the identity proceeds by elimination using the universal properties of e_1 and e_2. □

Ⓐ **Corollary 7.6.8.** *Write $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ for the type $\Pi_{(n,A):\mathbb{N}_{-2}\times\mathcal{U}}\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$. Then, $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ is propositional as Π preserves the truncation level (Lemma 2.2.6).* □

Assuming $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ is inhabited at the necessary points, we need information on how our notion of truncation interacts with dependent pair types and path spaces. The corresponding lemmata are well-known for the truncation with judgmental computation rule ("well-known" in the sense that they are listed in [Uni13]). Their proofs can be modified to only use the universal property, allowing us to transfer them to our setting. As explained in Remark 7.6.5, reasoning about reduction behaviour propositionally is tedious, particularly so whenever it occurs in a type. The second lemma is the reason why we need to parametrize $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ over *two* universes, enabling us to perform the correct elimination steps. Still, the proofs of the lemmata follow well-known ideas. We only give rough sketches. Rigorous proofs can be found in the Agda formalisation in the electronic appendix.

The following lemma can be understood as "flattening" (see [Uni13, Section 6.12]) for truncations.

Ⓐ **Lemma 7.6.9.** *Let $t_A : \mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ and $\langle P, h_P\rangle : \mathsf{type}(t_A) \to \mathcal{U}^n$ (i.e. P is a family of types over $\mathsf{type}(t_A)$ and h_P is a proof that it is a family of n-types). Given $t_\Sigma : \mathcal{T}^n_{\mathcal{U},\mathcal{V}}(\Sigma(A).P \circ \mathsf{cons}(t_\Sigma))$, we have*

$$\mathsf{type}(t_\Sigma) \simeq \Sigma(\mathsf{type}(t_A)).P. \tag{7.102}$$

Proof. We can directly define functions

$$f : \Sigma(A).P \circ \mathsf{cons}(t_A) \to \Sigma(\mathsf{type}(t_A)).P \qquad f(a,x) :\equiv (\mathsf{cons}(t_A)(a), p) \tag{7.103}$$

$$g : \Pi_{a:A}(P(\mathsf{cons}(t_A)(a)) \to \mathsf{type}(t_\Sigma)) \qquad g(a) :\equiv \lambda p.\mathsf{cons}(t_\Sigma)(a,p) \tag{7.104}$$

After applying the universal property of $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(\Sigma(A).P \circ \mathsf{cons}(t_\Sigma))$, the function f gives us a map from the left-hand side to the right-hand side of the equivalence (7.102). For the other direction, we need to use the dependent universal property of $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(A)$ together with g and uncurry the constructed function.

To prove that the two maps are inverses of each other, we use the (dependent) universal properties again. Both directions are straightforward. □

In the statement of the next lemma, note that $\mathcal{T}^m_{\mathcal{U},\mathcal{U}}(X)$ is always implied by $\mathcal{T}^n_{\mathcal{U},\mathcal{V}}(X)$ (if $\mathcal{U} : \mathcal{V}$) and is therefore a more minimalistic assumption.

Ⓐ **Lemma 7.6.10.** *Let $\mathcal{U}$, A, n be as before, but let $\mathcal{V}$ be a universe larger than $\mathcal{U}$ in the sense that we have $\mathcal{U} : \mathcal{V}$. Assume we have $t_A : \mathcal{T}^{n+1}_{\mathcal{U},\mathcal{V}}(A)$ as well as $t_P : \Pi_{a_1,a_2:A} \mathcal{T}^n_{\mathcal{U},\mathcal{U}}(a_1 = a_2)$. Then, for any given $a_1, a_2 : A$, the equivalence*

$$\mathsf{type}(t_P\, a_1\, a_2) \;\simeq\; \mathsf{cons}(t_A)(a_1) =_{\mathsf{type}(t_A)} \mathsf{cons}(t_A)(a_2) \tag{7.105}$$

holds.

Proof. The proof is analogous to the one in [Uni13, Theorem 7.3.12], but more tedious for lack of judgmental computation rules. In the application of the encode-decode method, it is necessary to use the dependent universal property of t_A to construct a map into $\mathcal{U}^n$, a type that does not inhabit $\mathcal{U}$, and that is why we need to ask for the universal property with respect to types in $\mathcal{V}$ instead. □

7.6.3 Construction of n-connected Types

With the previous lemmata at hand, we are able to perform the discussed construction.

Ⓐ **Definition 7.6.11.** In MLTT, given a pointed type (A, a) and $n \geq -1$, we write $\lfloor A, a \rfloor^n$ for the *n-connected version* of (A, a), defined by

$$\lfloor A, a \rfloor^n :\equiv \Sigma\,(b : A)\,.\,\Pi_{t:\mathcal{T}^{n-1}_{\mathcal{U},\mathcal{U}}(a=_A b)} \mathsf{type}(t). \tag{7.106}$$

It has the canonical inhabitant $(a, \overline{a})$ where $\overline{a} :\equiv \lambda t.\mathsf{cons}(t)(\mathsf{refl}_a)$. Note that $\lfloor A, a \rfloor^n$ is not a type in $\mathcal{U}$ but a type in $\mathcal{V}$ for any $\mathcal{V}$ satisfying $\mathcal{U} : \mathcal{V}$.

In the above definition, we include the case $n \equiv -1$, but note that it trivial in the sense that $\lfloor X \rfloor^{-1} \simeq X$. The construction is interesting already for $n \equiv 0$ though, which corresponds to the *connected component* of a.

Ⓐ **Lemma 7.6.12.** *In $\mathsf{MLTT}^{\mathsf{UA}}$, $\lfloor A, a \rfloor^n$ is, on dimension $n+1$ and above, equivalent to A, in the sense that*

$$\Omega^{n+1}(A, a) \;=\; \Omega^{n+1}(\lfloor A, a \rfloor^n, (a, \overline{a})). \tag{7.107}$$

Proof. If we write $(\lfloor A, a \rfloor^n, \overline{a})$ as a pointed dependent pair type with an $(n-1)$-truncated pointed family over (A, a), the statement follows from Lemma 7.4.1. □

Note that the formulation of the above lemma is justified by the fact that the n-th dimension of a type is entirely described by the n-th loop spaces. If $a = b$, then the types $a = a$ and $a = b$ are equivalent, and therefore, examining the elements of the loop space is enough to determine the whole structure of a type above some dimension.[2]

[2] This principle was used in the proof of Lemma 2.2.17.

Let us now discuss the connectedness properties of the type $\lfloor X \rfloor^n$ for appropriate X and n.

Ⓐ **Definition 7.6.13.** In plain MLTT, given universes $\mathcal{U}$ and $\mathcal{V}$ as well as $C : \mathcal{V}$, we define the following notion of n-connectedness:

$$\mathsf{is}\text{-}n\text{-}\mathsf{connected}^{\star}_{\mathcal{U},\mathcal{V}}(C) :\equiv \Pi_{t:\mathcal{T}_{\mathcal{U},\mathcal{V}}} \Pi_{t_C:\mathcal{T}^n_{\mathcal{V},\mathcal{V}}(C)} \mathsf{isContr}(\mathsf{type}(t_C)) \tag{7.108}$$

The notation $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ is used as introduced in Corollary 7.6.8.

Ⓐ **Lemma 7.6.14.** *In* $\mathsf{MLTT}^{\mathsf{UA}}$*, for any pointed type* $A : \mathcal{U}$ *(with a point* $a : A$*) and universe* $\mathcal{V}$ *that contains* $\mathcal{U}$*, i.e.* $\mathcal{U} : \mathcal{V}$*, the type* $\lfloor A, a \rfloor^n$ *is* n*-connected in the above sense, i.e.*

$$\mathsf{is}\text{-}n\text{-}\mathsf{connected}^{\star}_{\mathcal{U},\mathcal{V}}(\lfloor A, a \rfloor^n). \tag{7.109}$$

Proof. Unfolding the definitions, given

$$\begin{aligned} t &: \mathcal{T}_{\mathcal{U},\mathcal{V}}, \\ t_C &: \mathcal{T}^n_{\mathcal{V},\mathcal{V}}\Big(\Sigma\,(b : A)\,.\,\Pi_{s:\mathcal{T}^{n-1}_{\mathcal{U},\mathcal{U}}(a =_A b)} \mathsf{type}(s)\Big), \end{aligned} \tag{7.110}$$

we need to show that $\mathsf{type}(t_C)$ is contractible.

We perform a sequence of steps, each transforming $\mathsf{type}(t_C)$ into an equivalent type. This is especially true for the very first step: by Corollary 7.6.8 and Lemma 2.2.9(ii), the remaining occurrence of Π in (7.110) can be removed, as $t(n-1, a =_A b)$ can be viewed as an inhabitant of $\mathcal{T}^{n-1}_{\mathcal{U},\mathcal{U}}(a =_A b)$. Consequently it is enough to prove the type

$$\mathsf{type}\big(t\,(n, \Sigma\,(b : A)\,.\,\mathsf{type}(t(n-1, a =_A b)))\big) \tag{7.111}$$

contractible. Next, we apply Lemma 7.6.10 which provides an equivalence

$$\mathsf{type}(t(n-1, a =_A b)) \;\simeq\; P(\mathsf{cons}(t(n, A))(b)) \tag{7.112}$$

where

$$P \;:\; \mathsf{type}(t(n, A)) \to \mathcal{U} \tag{7.113}$$

$$P(x) :\equiv \mathsf{cons}(t(n, A))(a) =_{\mathsf{type}(t(n,A))} x, \tag{7.114}$$

transforming the type (7.111) into

$$\mathsf{type}\,(t\,(n, \Sigma\,(A)\,.\,P \circ \mathsf{cons}(t(n, A))))\,. \tag{7.115}$$

Clearly, P is a family of $(n-1)$-types. We may therefore apply Lemma 7.6.9, telling us that the type (7.115) is equivalent to

$$\Sigma\,(\mathsf{type}(t(n, A)))\,.\,P, \tag{7.116}$$

and this is a singleton. □

The name of the entity in Definition 7.6.13 is justified in that we took the standard definition of connectivity and replaced the use of truncation with an inhabitant of our type of truncations before quantifying over it. Unsurprisingly, it turns out to be equivalent to the standard notion of connectedness (7.90) if available.

Ⓐ **Lemma 7.6.15.** *In* $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$*, for any* $\mathcal{U}$ *and* $\mathcal{V}$*, we have*

$$\Pi_{A:\mathcal{U}}\Pi_{n:\mathbb{N}}\ \mathsf{is}\text{-}n\text{-}\mathsf{connected}^{\star}_{\mathcal{U},\mathcal{V}}(A) \simeq \mathsf{is}\text{-}n\text{-}\mathsf{connected}(A). \tag{7.117}$$

Proof. In $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$, the function type $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ has a canonical inhabitant, as mentioned in Remark 7.6.5. Together with Lemma 7.6.7 and Corollary 7.6.8, this shows that $\mathcal{T}_{\mathcal{U},\mathcal{V}}$ is contractible, and Lemma 2.2.9(ii) shows the stated equivalence. □

Ⓐ **Remark 7.6.16.** n-connectedness can be defined without referring to general truncations and only by using propositional truncations instead [Uni13, Exercise 7.6]. This definition could be seen as somewhat more "elementary" in the sense that propositional truncations, or notions similar to them, have been considered a long time before HoTT was developed. Instead of saying that a type is n-connected if its n-truncation is contractible, we could ask for it to be "merely" inhabited (in the sense of [Uni13]), and all its path spaces to be $n-1$-connected:

$$\mathsf{is}\text{-}{-2}\text{-}\mathsf{connected}'(A) :\equiv \mathbf{1} \tag{7.118}$$

$$\mathsf{is}\text{-}(n+1)\text{-}\mathsf{connected}'(A) :\equiv \|{-1}\|_A \times \Pi_{x,y:A}\mathsf{is}\text{-}n\text{-}\mathsf{connected}'(x = y) \tag{7.119}$$

This notion of connectedness is equivalent to the one stated in Definition 7.6.1. We think that, generally in HoTT, there might be cases where this notion is more convenient to use, as it supports direct induction on n. Unfortunately, it does not seem suitable for a modification that gives a strong enough definition of connectedness in $\mathsf{MLTT}^{\mathsf{UA}}$ as does our Definition 7.6.13.

7.7 Combining the Results

In Section 7.4, we have seen that $\mathcal{U}_n$ is not an n-type and that $\mathcal{U}_{n+1}$ contains a "strict" $(n+1)$-type, namely $\mathcal{U}_n^n$. Combining that result with the construction of the previous section, we immediately get:

Ⓐ **Theorem 7.7.1.** *In* $\mathsf{MLTT}^{\mathsf{UA}}$*, for a given natural number* n*, we can construct a type, defined by*

$$M_n : \mathcal{U}_{n+2} \tag{7.120}$$

$$M_n :\equiv \lfloor \mathcal{U}_n^n, (\mathsf{Loop}_{n-1}, h_n) \rfloor^n \tag{7.121}$$

(with h_n as in Lemma 7.4.5) such that, purely in $\mathsf{MLTT}^{\mathsf{UA}}$*, the following propositions are provable:*

$$M_n \text{ is } (n+1)\text{-truncated:} \quad \mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(M_n) \tag{7.122}$$

$$M_n \text{ is not } n\text{-truncated:} \quad \neg\mathsf{is}\text{-}n\text{-}\mathsf{type}(M_n) \tag{7.123}$$

$$M_n \text{ is } n\text{-connected:} \quad \mathsf{is}\text{-}n\text{-}\mathsf{connected}^{\star}_{\mathcal{U}_{n+2},\mathcal{U}_{n+3}}(M_n) \tag{7.124}$$

The first two properties come from the construction of homotopically complicated types, the third comes from the previous section on connectedness. Note that these properties also imply that, in $\mathsf{MLTT}^{\mathsf{UA}}_{\mathsf{TRUNC}}$*, the n-th homotopy group of M_n is non-trivial, and all other homotopy groups are trivial.*

Proof. The first statement holds as both components of the Σ-type M_n are $(n+1)$-truncated. For the second, we combine Lemmata 2.2.17, 7.4.6 and 7.6.12. The third part is an application of Lemma 7.6.14 with $\mathcal{U} \equiv \mathcal{U}_{n+2}$ and $\mathcal{V} \equiv \mathcal{U}_{n+3}$. □

A related result with much stronger assumptions was shown before using *Eilenberg-Mac Lane Spaces* [Uni13, Theorem 8.10.3],[LF14]: in MLTT with univalence and not just truncations, but general higher inductive types, it is possible to construct a type $K(G,n)$ that is an n-type such that the n-th homotopy group equals some abelian group G and all the others are trivial. That construction uses higher inductive types not just to truncate, but also to produce the actual non-trivial higher paths. This might not be too surprising as the property of $K(G,n)$ points directly to that usage of HITs. We have shown that, even without them, we can get quite close. If we wanted to, we could use the second approach of defining a strict $(n+1)$-type (the one from Section 7.5); then, we could control the single non-trivial homotopy group of our construction explicitly (it would, of course, be $\mathbf{2}$). Obviously, we would again loose a single universe level.

Chapter 8

The General Universal Properties of Truncations

The propositional truncation has an elimination rule which only allows us to construct functions into other propositions. So, what are we supposed to do if we need a map $\|A\| \to B$ in general? If B is not propositional, the elimination rule of the truncation feels very restrictive. A map $A \to B$ will not be sufficient and one has to find a workaround.

Let us compare a function $f : A \to B$ and a function $g : \|A\| \to B$. The first can "look at its input" (that is, $f(a)$ can of course depend on a), while the second can not make a distinction between any two of its inputs.[1] g only gets an "anonymous" inhabitant of A. Intuitively, this corresponds to saying that g gets some inhabitant of A, but has to return the same value in B anyway, no matter what the input is. In other words, we might say that g corresponds to a constant function from A to B in some sense. Recall (from Definition 3.1.2) that we call f *weakly constant* if

$$\mathsf{const}_f :\equiv \forall(x_1\, x_2 : X).\, f(x_1) = f(x_2). \tag{8.1}$$

It is not surprising that, in homotopy type theory, this notion of constancy is not entirely satisfactory. We always emphasise that there might be different paths between two points, so just asking for *any* such path does not feel right. Indeed, we have discussed weakly constant functions in Chapter 5, and we have (in Section 5.1) seen that a weakly constant function $A \to B$ is (intuitively) still not enough to get $\|A\| \to B$, even though it works in some interesting special cases (see Section 5.2).

Let us go back to the question of how to define a function $\|A\| \to B$ in general. The standard approach of finding a propositional type "in the middle" has been described at the beginning of Section 5.2 (and slightly improved with Principle 5.2.1). It feels somewhat odd that this propositional type has to be found

[1] Of course, we only think of internal properties here. When it comes to computation, the *term* f can certainly behave differently if applied on not judgmentally equal *terms* of type $\|A\|$.

by an ad-hoc construction. In Proposition 5.2.3, we have seen that a weakly constant function *is* enough if the codomain is a set, and by Proposition 5.2.4, the types $\|A\| \to B$ and $\Sigma\,(f : X \to Y)\,.\,\mathsf{const}_f$ are indeed equivalent. This is already better than the original elimination principle, as it weakens the condition on B from "propositional" to "being a set". It is hard to imagine a situation in which constructing a function $\|A\| \to B$ could be done more easily than by construction a weakly constant $A \to B$. The type which we have used in the proof of Proposition 5.2.3 is essentially the "propositional type in the middle". If we apply that lemma, the task of defining $\|A\| \to B$ becomes easier *at least* in the sense that we do not need to find Q ourselves any more. It can thus be used to streamline proofs of that form.

Admittedly, Proposition 5.2.3 is very simple. But what if we want to get $\|A\| \to B$, but B is not even a set? It could be that B is a 1-type, a 2-type, of some other (known) truncation level, or maybe a totally unknown type. It is natural to ask whether we can formulate functions $A \to B$ with some kind of "better" constancy condition which let us deal with that situation.

The contribution of this chapter in an answer to these questions. It turns out that, for 1-types B, a function $\|A\| \to B$ corresponds exactly to a weakly constant function $A \to B$ which, in addition, satisfies one coherence condition. If we weaken the assumptions on B further, we need to add more and more such coherence conditions. The most interesting case is the one in which we do not know anything about B (we could say that the only truncation level we know is ω). Unfortunately, in this case, we need an infinite tower of coherence conditions to define the type $A \xrightarrow{\omega} B$ of "completely coherent" constant functions from A to B. Then, we can prove that $A \xrightarrow{\omega} B$ and $\|A\| \to B$ are equivalent. We can not do this in the type theory that we have considered so far; however, we can do it if we assume the existence of certain *Reedy limits* (see Shulman's work on diagrams over inverse categories [Shu15]).

More precisely, we use Shulman's setting of a type theoretic fibration category, which representing a dependent type theory with at least $\mathbf{1}$, Σ, Π, identity types, and Reedy ω^{op}-limits. The only assumptions here that are not satisfied in homotopy type theory are the strict (judgmental) uniqueness (or η-) rule for Σ-types, and the existence of the mentioned Reedy ω^{op}-limits. Strict η for Σ-types is completely natural and holds, for example, in the theory that Agda implements, as we have already discussed in Section 1.4. Shulman even writes that strict η is essentially unnecessary but simplifies the presentation [Shu15, Example 2.9]. This is certainly true in many cases, but it seems unclear whether referring to strict η can really be avoided in the proof of our main result (Theorem 8.8.5).

More interesting is the assumption of Reedy ω^{op}-limits. It allows us to consider Σ-types with infinitely many components in a sense that we will make precise in Section 8.2, which is just what we need to define $A \xrightarrow{\omega} B$. However, if we consider an n-truncated type B for some finite fixed number n, then nearly all of the coherence conditions captured by $A \xrightarrow{\omega} B$ become trivial, and that type expression can be simplified to a nested Σ-type with $n + 2$ components, for which we will

write $A \xrightarrow{[n+1]} B$. It can be formulated in the standard syntactical version of HoTT, where we can then prove that, for any A and any n-truncated B, the type $A \xrightarrow{[n+1]} B$ is equivalent to $\|A\| \to B$.

We thereby generalise the usual universal property of the propositional truncation (see Lemma 2.3.3), because if B is not only n-truncated, but propositional, then $A \xrightarrow{[n+1]} B$ is equivalent to $A \to B$. This equivalence will in fact be simply a projection: the type expression $A \xrightarrow{[n+1]} B$ will be a nested Σ-type with $n+2$ components, the very first of which will be $A \to B$. Moreover, all except this first one will represent contractible types. We call such a component of a nested Σ-type expression which represents a contractible type a *contractible Σ-component*. Note however that these are components in the purely syntactical sense, and have nothing to do with components in the topological interpretation, which we will use in Section 8.10.2. A "practical" application of the result is thus the construction of maps $\|A\| \to B$ if B is known to be n-truncated (for some fixed n).

Nevertheless, we want to stress that we consider the correspondence between $A \xrightarrow{\omega} B$ and $\|A\| \to B$ in a type theoretic fibration category with Reedy ω^{op}-limits our main result, and the finite special cases described in the previous paragraph essentially fall out as a corollary. In fact, we think that Reedy ω^{op}-limits are a somewhat reasonable assumption. Recently, it has been discussed regularly how these or similar concepts can be introduced into syntactical type theory (for example, see the blog posts [Shu14] and [Oli14] with the comments sections, and the discussion on the HoTT mailinglist [Hmail] titled *Infinitary type theory*). A major motivation is the question whether HoTT can serve as its own meta-theory, whether we can write an interpreter for HoTT in HoTT, and related questions such as the definition of semi-simplicial types [Her15]. Moreover, a concept that is somewhat similar has been suggested earlier as *very dependent types* ([Hic96]; see also our Section 9.2 below), even though this suggestion was made in the setting of NuPRL [Con+86].

The general principle for constructing equivalences is "adding and removing contractible Σ-components" to expressions, or "expanding and contracting". This is a very clear example of our general equivalence reasoning strategy (see Section 2.2.5), and we strive to explain the principle further with the help of the examples in Section 8.1. As we have mentioned in Section 1.4, we do not expect that the main results of this chapter can be be internalised in the considered type theory. These main contents are to appear in the post-proceedings of TYPES'14 [Kra14b].

The special case for higher truncations, Theorem 8.10.2, can be formalised, and this has been done by Paolo Capriotti [Cap15].

As an anonymous reviewer of [Kra14b] has pointed out, our result can probably be seen as a type theoretic version of Proposition 6.2.3.4 in Lurie's *Higher Topos Theory* [Lur09]. It certainly would be interesting to check whether the assumptions of that proposition are satisfied in our situation and whether the proof of Lurie's proposition is constructive. More general, many of the connections between type theory and higher topos theory have yet to be explored.

Organisation of the chapter We first discuss the cases that the codomain B is a set or a groupoid in Section 8.1. This is intended to provide some intuition for our general strategy of proving a correspondence between coherently constant functions and maps out of propositional truncations. In Section 8.2, we briefly review the notion of a type theoretic fibration category, of an inverse category, and, most importantly, constructions related to Reedy fibrant diagrams, as described by Shulman [Shu15]. Some simple observations about the restriction of diagrams to subsets of the index categories are recorded in Section 8.3. We proceed by defining the *equality diagram* over a given type for a given inverse category in Section 8.4. The special case where the inverse category is Δ_+^{op} (the category of nonempty finite sets and strictly increasing functions) gives rise to the *equality semi-simplicial type*, which is discussed in Section 8.5. We show that the projection of a full n-dimensional tetrahedron to any of its horns is a homotopy equivalence. Then, in Section 8.6, we manually construct a fibrant diagram that more or less represents the exponential of a fibrant and a non-fibrant diagram. We extend the category Δ_+^{op} in Section 8.7, which allows us to make precise how contractible Σ-components can be "added and removed" in general. Our main result, namely that the types $A \xrightarrow{\omega} B$ and $\|A\| \to B$ are homotopy equivalent, is shown in Section 8.8. In Section 8.9, we derive the finite versions of the main theorem: if B is n-truncated for any fixed finite number n, we can formulate an equivalence between $\|A\| \to B$ and a type of constant functions with n coherence condition. This equivalence can be stated and proved in standard MLTT with propositional truncation. Further, we discuss the case of higher truncations in Section 8.10. We characterise functions $\|A\|_n \to B$, i.e. we prove a universal property of $\|-\|_n$, but only for $(n+1)$-types B. The intuition is that, to eliminate from the n-truncation of A into some m-type B, we need a function from A to B that is constant (in an appropriate sense) on all $(n+1)$-st loop spaces, and satisfies "all possible" coherence conditions. For example, to get a map $\|A\|_0 \to B$, where B is 1-truncated, it is necessary to have a function $f : A \to B$ such that, for any $a : A$ and $p : a = a$, the loop $\mathsf{ap}_f(p)$ is trivial. For the case $m \equiv n+1$, we give two proofs: the first uses a construction that is similar to the Rezk completion [AKS15], and the second is based on, and generalises, a (privately discussed) argument by Vezzosi. In Section 8.11, we summarise which sort of universal properties (i.e. which cases for n, m) we have solved and which are open. We provide an argument for the fact that the main difficulties in the case of the propositional truncation vanish for higher truncations, which introduce their own coherence issues instead.

Notation If C is some category and $x \in C$ an object, we write (as it is standard) x/C for the co-slice category of arrows $x \to y$. We do many constructions involving subcategories, but we want to stress that we always and exclusively work with *full* subcategories (apart from the subcategory of fibrations in Definition 8.2.1). Thus, we write $C - x$ for the full subcategory of C that we get by removing the object x. Further, if D is a full subcategory of C (we write $D \subset C$) which does not contain x, we write $D + x$ for the full subcategory of C that has

all the objects of D and the object x.

We also recall our convention that we outlined in Section 2.4: Notions of *equality* that are internal to the type theory in question (the identity type and equivalence of types) are written using "two-line" symbols ($=$, $\simeq$), while non-internal concepts (such as isomorphism of objects in a category, or judgmental equality) are denoted by "three-line" symbols (such as $\equiv$, $\cong$).

8.1 A First Few Special Cases

In this section, we want to discuss some simple examples and aim to build up intuition for the general case. For now, we work entirely in standard (syntactical) homotopy type theory as specified in Chapter 2 (or [Uni13, Appendix A.2]), together with function extensionality and (weak) propositional truncation. These examples here probably show best why we can avoid "computation overhead" thanks to the equivalence-reasoning style, even if we do not assume judgmental computation rules. It might be worth mentioning that we actually do not require much of the power of HoTT: we only use $\mathbf{1}$, Σ, Π, identity types, propositional truncations, and assume function extensionality. We do not need univalence, higher inductive types, general truncations, a hierarchy of universes, or any other additional "feature". This will in later sections turn out to be a key feature which enables us to perform the construction in the infinite case (assuming the existence of Reedy ω^{op}-limits).

Assume we want to construct an inhabitant of $\|A\| \to B$ and B is an n-type, for a fixed given n. The case $n \equiv -2$ is trivial. For $n \equiv -1$, the universal property (or the elimination principle) can be applied directly. In this section, we explain the cases $n \equiv 0$ and $n \equiv 1$. The following auxiliary statement will be useful:

Lemma 8.1.1. *Let $C_1, C_2, \dots, C_m$ be types dependent on A, possibly with C_j depending on C_i for $i < j$. Consider a nested Σ-type, built out of components of the form $\Pi_A C_k$. Then, functions from $\|A\|$ into that type correspond directly to elements of that type. That is, the types*

$$\begin{aligned}\|A\| \to \big(&\Sigma\,(f_1 : \Pi_{a:A} C_1(a))\,.\\ &\Sigma\,(f_2 : \Pi_{a:A} C_2(a, f_1(a)))\,.\\ &\Sigma \quad \dots\\ &\quad (\Pi_{a:A} C_m(a, f_1(a), f_2(a), \dots, f_{m-1}(a)))\big)\end{aligned}$$

and

$$\begin{aligned}&\Sigma\,(f_1 : \Pi_{a:A} C_1(a))\,.\\ &\Sigma\,(f_2 : \Pi_{a:A} C_2(a, f_1(a)))\,.\\ &\Sigma \dots\\ &\quad (\Pi_{a:A} C_m(a, f_1(a), f_2(a), \dots, f_{m-1}(a)))\end{aligned}$$

are equivalent.

Proof. This holds by the usual distributivity law (Lemma 2.2.12) of Π (or $\to$) and Σ, together with the equivalence $\|A\| \times A \simeq A$. □

8.1.1 Constant Functions into Sets

We consider the case $n \equiv 0$ first; that is, we assume that B is a set. We have already seen and proved the following statement before, namely in Chapter 5 as Proposition 5.2.4. At that point, we have shown the equivalence by constructing functions in both ways, while we here use a strategy that is closer to the way in which we prove the general case later.

Proposition 8.1.2. *Let B be a set and A any type. Then, we have the equivalence*

$$(\|A\| \to B) \;\simeq\; \Sigma\,(f : A \to B)\,.\,\mathsf{const}_f. \tag{8.2}$$

Note that, if B is not only a set but even a proposition, the condition const_f is not only automatically satisfied, but it is actually contractible as a type. By the usual equivalence lemma (2.2.8), the type on the right-hand side of (8.2) then simplifies to $A \to B$, which exactly is the universal property. Thus, we view (8.2) as a first generalisation.

Proof of Proposition 8.1.2. Assume $\mathfrak{a}_0 : A$ is some point in A. In the following, we construct a chain of equivalences. The variable names for certain components might seem somewhat odd: for example, we introduce a point $f_1 : B$. The reason for this choice will become clear later. For now, we simply emphasise that f_1 is "on the same level" as $f : A \to B$ in the sense that they both give points, rather than for example paths (like, for example, an inhabitant of const_f).

$$\begin{aligned}
& B \\
(\text{S1}) \quad \simeq \quad & \Sigma\,(f_1 : B)\,.\,\big(A \to \Sigma\,(b : B)\,.\,b = f_1\big) \\
(\text{S2}) \quad \simeq \quad & \Sigma\,(f_1 : B)\,.\,\Sigma\,(f : A \to B)\,.\,\Pi_{a:A} f(a) = f_1 \\
(\text{S3}) \quad \simeq \quad & \Sigma\,(f_1 : B)\,.\,\Sigma\,(f : A \to B)\,.\,(\Pi_{a:A} f(a) = f_1) \times (\mathsf{const}_f) \times (f(\mathfrak{a}_0) = f_1) \\
(\text{S4}) \quad \simeq \quad & \Sigma\,(f : A \to B)\,.\,(\mathsf{const}_f) \times \Sigma\,(f_1 : B)\,.\,(f(\mathfrak{a}_0) = f_1) \times (\Pi_{a:A} f(a) = f_1) \\
(\text{S5}) \quad \simeq \quad & \Sigma\,(f : A \to B)\,.\,(\mathsf{const}_f) \times (\Sigma\,(f_1 : B)\,.\,f(\mathfrak{a}_0) = f_1) \\
(\text{S6}) \quad \simeq \quad & \Sigma\,(f : A \to B)\,.\,\mathsf{const}_f
\end{aligned}$$

Let us explain the validity of the single steps. In the first step, we add a family of singletons. In the second step, we apply the distributivity law (Lemma 2.2.12). In the third step, we add two components, and B being a set ensures that both of them are propositional. But it is very easy to derive both of them from $\Pi_{a:A} f(a) = f_1$, showing that both of them are contractible. In the fourth step, we simply reorder some components, and in the fifth step, we use that $\Pi_{a:A} f(a) = f_1$

is contractible by an argument analogous to that of the third step. Finally, we can remove two components which form a contractible singleton.

If we carefully trace the equivalences, we see that the function part

$$e : B \to \Sigma\,(f : A \to B)\,.\,\mathsf{const}_f \tag{8.3}$$

is given by

$$e(b) \equiv \left(\lambda a.b\,,\ \lambda a^1 a^2.\mathsf{refl}_b\right), \tag{8.4}$$

not depending on the assumed $\mathfrak{a}_0 : A$. But as e is an equivalence assuming A, it is also an equivalence assuming $\|A\|$.

As $\|A\| \to \left(B \simeq (\Sigma\,(f : A \to B)\,.\,\mathsf{const}_f)\right)$ implies that the types $(\|A\| \to B)$ and $\left(\|A\| \to (\Sigma\,(f : A \to B)\,.\,\mathsf{const}_f)\right)$ are equivalent, the statement follows from Lemma 8.1.1. □

The core strategy of the steps $(S1)$ to $(S6)$ is to add and remove contractible Σ-components, and to reorder and regroup them. This principle of expanding and contracting a type expression can be generalised and, as we will see, even works for the infinite case when B is not known to be of any finite truncation level. Generally speaking, we use two ways of showing that components of Σ-types are contractible. The first is to group two of them together such that they form a singleton, as we did in $(S1)$ and $(S6)$. The second is to use the fact that B is truncated, as we did in $(S3)$. We consider the first to be the key technique, and in the general (infinite) case of an untruncated B, the second can not be applied at all. We thus view the second method as a tool to deal with single components that lack a "partner" only because the case that we consider is finite, and which is unneeded in the infinite case.

8.1.2 Constant Functions into Groupoids

The next special case is $n \equiv 1$. Assume that B is a groupoid (1-type). Let us first clarify which kind of constancy we expect for a map $f : A \to B$ to be necessary. Not only do we require $c : \mathsf{const}_f$, we also want this constancy proof (which is in general not propositional any more) to be *coherent*: given a^1 and $a^2 : A$, we expect that c only allows us to construct essentially *one* proof of $f(a^1) = f(a^2)$. The reason is that we want the data (which includes f and c) together to be just as powerful as a map $\|A\| \to B$, and from such a map, we only get trivial loops in B.

We claim that the required coherence condition is

$$\mathsf{coh}_{f,c} :\equiv \Pi_{a^1 a^2 a^3 : A}\, c(a^1, a^2) \cdot c(a^2, a^3) = c(a^1, a^3). \tag{8.5}$$

A first sanity check is to see whether from $d : \mathsf{coh}_{f,c}$ we can now prove that $c(a, a)$ is equal to refl_a, something that should definitely be the case if we do not want to be able to construct possibly different parallel paths in B. To give a positive answer, we only need to see what $d(a, a, a)$ tells us.

Proposition 8.1.3 (case $n \equiv 1$)**.** *Let B be a groupoid (1-type) and A any type. Then, we have*

$$(\|A\| \to B) \;\simeq\; \big(\Sigma\,(f : A \to B)\,.\,\Sigma\,(c : \mathsf{const}_f)\,.\,\mathsf{coh}_{f,c}\big). \tag{8.6}$$

Note that Proposition 8.1.3 generalises Proposition 8.1.2: if B is a set (as in Proposition 8.1.2), it is also a groupoid and the type $\mathsf{coh}_{f,c}$ becomes contractible, as it talks about equality of equalities.

Proof. Although not conceptually harder, it is already significantly more tedious to write down the chain of equivalences. We therefore choose a slightly different representation. Assume $\mathfrak{a}_0 : A$ as before. We then have:

$$\begin{aligned}
& B \\
(\text{S1}) \quad \simeq\; & \\
& \Sigma\,(f_1 : B)\,. \\
& \Sigma\,(f : A \to B)\,.\,\Sigma\,(c_1 : \Pi_{a:A} f(a) = f_1)\,. \\
& \Sigma\,(c : \mathsf{const}_f)\,.\,\Sigma\,\big(d_1 : \Pi_{a^1 a^2 : A} c(a^1, a^2) \cdot c_1(a^2) = c_1(a^1)\big)\,. \\
& \Sigma\,(c_2 : f(\mathfrak{a}_0) = f_1)\,.\,\Sigma\,(d_3 : c(\mathfrak{a}_0, \mathfrak{a}_0) \cdot c_1(\mathfrak{a}_0) = c_2)\,. \\
& \Sigma\,(d : \mathsf{coh}_{f,c_0})\,. \\
& \quad (d_2 : \Pi_{a:A} c(\mathfrak{a}_0, a) \cdot c_1(a) = c_2) \\
(\text{S2}) \quad \simeq\; & \\
& \Sigma\,(f : A \to B)\,.\,\Sigma\,(c : \mathsf{const}_f)\,.\,\Sigma\,(d : \mathsf{coh}_{f,c})\,. \\
& \Sigma\,(f_1 : B)\,.\,\Sigma\,(c_2 : f(\mathfrak{a}_0) = f_1)\,. \\
& \Sigma\,(c_1 : \Pi_{a:A} f(a) = f_1)\,.\,\Sigma\,(d_2 : \Pi_{a:A} c(\mathfrak{a}_0, a) \cdot c_1(a) = c_2)\,. \\
& \Sigma\,\big(d_1 : \Pi_{a^1 a^2 : A} c(a^1, a^2) \cdot c_1(a^2) = c_1(a^1)\big)\,. \\
& \quad (d_3 : c(\mathfrak{a}_0, \mathfrak{a}_0) \cdot c_1(\mathfrak{a}_0) = c_2) \\
(\text{S3}) \quad \simeq\; & \\
& \Sigma\,(f : A \to B)\,.\,\Sigma\,(c : \mathsf{const}_f)\,.\,(d : \mathsf{coh}_{f,c})
\end{aligned}$$

In the first step (S1), we add five contractible parts of a nested Σ-type, each line apart from the first representing one.

To bring the lines two and three in the form of singletons, we apply the distributivity law (Lemma 2.2.12), while line four already is a singleton. Lines five and six clearly represent propositional types which are additionally derivable from the other Σ-components. In the second step, we simply re-order some components. Then, in step (S3), we remove several contractible parts (again, each but the first line is a contractible part of the nested Σ-type). We trace the canonical equivalences to see that the function-part of the constructed equivalence is

$$e : B \to \Sigma\,(f : A \to B)\,.\,\Sigma\,(c : \mathsf{const}_f)\,.\,(d : \mathsf{coh}_{f,c}) \tag{8.7}$$

$$e(b) \equiv (\lambda a.b\,,\; \lambda a^1 a^2.\mathsf{refl}_b\,,\; \lambda a^1 a^2 a^3.\mathsf{refl}_{\mathsf{refl}_b}), \tag{8.8}$$

and the conclusion follows as in the proof of Proposition 8.1.3. □

8.1.3 Outline of the General Idea

At this point, it seems plausible that what we have done for the special cases of $n \equiv 0$ and $n \equiv 1$ can be done for any (fixed) $n < \infty$. Nevertheless, we have seen that the case of groupoids is already significantly more involved than the case of sets. To prove a generalisation, we have to be able to state what it means for a function to be "coherently constant" on n levels, rather than just the first one or two.

Let us try to specify what "coherently constant" should mean in general. If we have a function $f : A \to B$, we get a point in B for any $a : A$. A constancy proof $c : \mathsf{const}_f$ gives us, for any pair of points in A, a path between the corresponding points in B. Given three points, c gives us three paths which form a "triangle", and an inhabitant of $\mathsf{coh}_{f,c}$ does nothing else than providing a filler for such a triangle. It does not take much imagination to assume that, on the next level, the appropriate coherence condition should state that the "boundary" of a tetrahedron, consisting of four filled triangles, can be filled.

To gain some intuition, let us look at the following diagram:

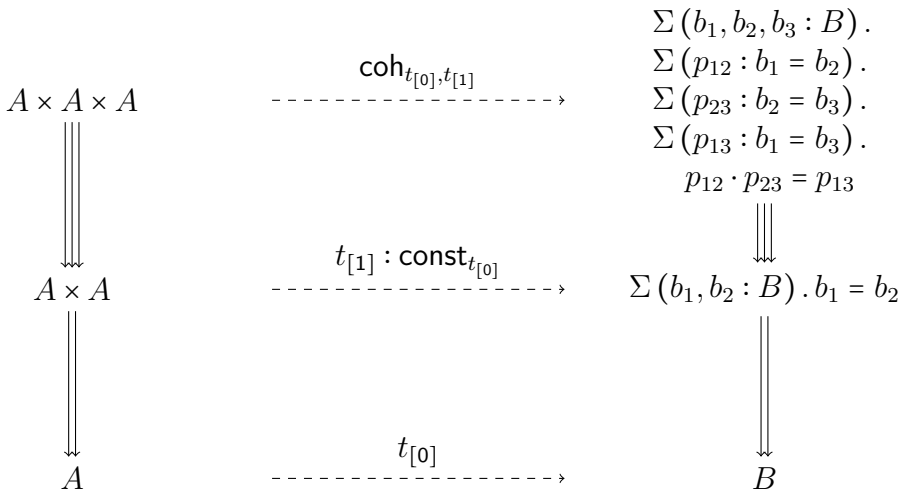


Figure 8.1: Constancy as a natural transformation

All vertical arrows are given by projections. Consider the category D with objects the finite ordinals $[0]$, $[1]$ and $[2]$ (with 1, 2, and 3 objects, respectively), and arrows the strictly monotonous maps. Then, the left-hand side and the right-hand side can both be seen as a diagram over D^{op}. The data that we need for a "coherently constant function" from A into B, if B is a groupoid, can now be viewed as a natural transformation t from the left to the right diagram (the dashed horizontal arrows). On the lowest level, such a natural transformation consists of a function $t_{[0]} : A \to B$, which we called f before. On the next level, we have $t_{[1]} : A^2 \to \Sigma\,(b_1, b_2 : B)\,.\, b_1 = b_2$, but in such a way that the diagram commutes

(strictly, not up to homotopy), enforcing

$$\mathsf{fst}(t_{[1]}(a^1, a^2)) \equiv (t_{[0]}(a^1), t_{[0]}(a^2)) \tag{8.9}$$

and thereby making $t_{[1]}$ the condition that $t_{[0]}$ is weakly constant. Finally, $t_{[2]}$ yields the coherence condition coh.

In the most general case, where we do not put any restriction on B, we certainly cannot expect that a finite number of coherence conditions can suffice. Instead of the diagram over D^{op}, as pictured on the right-hand side of Figure 8.1, we will need a diagram over the the category of all non-zero finite ordinals. This is what we call the *equality semi-simplicial type* over B, written $\mathcal{E}$ (with B being kept implicit). In the language of model categories, this is a fibrant replacement of the constant diagram. It would be reasonable to expect that our $\mathcal{E}$ extends the diagram shown in Figure 8.1, but this will only be true up to (levelwise) equivalence of types. Defining $\mathcal{E}$ as a strict extension of that diagram is tempting, but it seems to be combinatorically non-trivial to continue in the same style, as it would basically need Street's orientals [Str87]. Our construction will be much simpler to write down and easier to work with, with the only potential disadvantage being that, compared to the diagram Figure 8.1, the lower levels will look rather bloated. The other diagram in Figure 8.1, i.e. the left-hand side, is easy to extend, and we call it the *trivial* diagram over A. In the terminology of simplicial sets, it is the $[0]$-coskeleton of the constant diagram. Our main result is essentially an internalised version, stated as an equivalence of types, of the following slogan:

> *Functions $\|A\| \to B$ correspond to natural transformations from the trivial diagram over A to the semi-simplicial equality type over B.*

Our type of natural transformations is basically a Reedy limit of an exponential of diagrams. We will perform the *expanding and contracting* principle that we have exemplified in the proofs of Propositions 8.1.2 and 8.1.3 by modifying the index category of the diagram of which we take the limit, step by step, taking care that every single step preservers the Reedy limit in question up to homotopy equivalence. As we will see, these steps correspond indeed to the steps that we took in the proofs of Propositions 8.1.2 and 8.1.3.

8.2 Fibration Categories, Inverse Diagrams, and Reedy Limits

In his work on *Univalence for Inverse Diagrams and Homotopy Canonicity*, Shulman has proved several deep results [Shu15]. Among other things, he shows that diagrams over inverse categories can be used to build new models of univalent type theory, and uses this to prove a partial solution to Voevodsky's homotopy-canonicity conjecture. We do not require those main results; in fact, we do not even assume that there is a universe, an consequently we also do not use univalence! At the same time, what we want to do can be explained nicely in terms

of diagrams over inverse diagrams, and we therefore choose to work in the same setting. Luckily, it is possible to do this with only a very short introduction to type theoretic fibration categories, inverse diagrams and Reedy limits, and this is what the current section servers for.

Type-theoretic fibration categories A *type theoretic fibration category* (as defined in [Shu15, Definition 2.1] is a category with some structure that allows to model dependent type theory with identity types. Let us recall the definition, where we use a lemma by Shulman to give an equivalent (more "type-theoretic") formulation:

Definition 8.2.1 (Type-theoretic fibration category, [Shu15, Definition 2.1 combined with Lemma 2.4])**.** A type theoretic fibration category is a category $\mathfrak{C}$ which has the following structure.

1. A terminal object $\mathbf{1}$.
2. A (not necessarily full) subcategory $\mathfrak{F} \subset \mathfrak{C}$ containing all the objects, all the isomorphisms, and all the morphisms with codomain $\mathbf{1}$. A morphism in $\mathfrak{F}$ is called a *fibration*, and written as $A \twoheadrightarrow B$. Any morphism i is called an *acyclic cofibration* and written $i : X \overset{\sim}{\rightarrowtail} Y$ if it has the left lifting property with respect to all fibrations, meaning that every commutative square

$$\begin{array}{ccc} X & \longrightarrow & A \\ {\scriptstyle i}\downarrow{\scriptstyle\sim} & & \downarrow{\scriptstyle f} \\ Y & \longrightarrow & B \end{array}$$

   has a (not necessarily unique) filler $h : Y \to A$ that makes both triangles commute.
3. All pullbacks of fibrations exist and are fibrations.
4. For every fibration $g : A \twoheadrightarrow B$, the pullback functor $g^\star : \mathfrak{C}/B \to \mathfrak{C}/A$ has a partial right adjoint Π_g, defined at all fibrations over A, whose values are fibrations over B.
5. For any fibration $A \twoheadrightarrow B$, the diagonal morphism $A \to A \times_B A$ factors as $A \overset{\sim}{\rightarrowtail} P_B A \twoheadrightarrow A \times_B A$, with the first map being an acyclic cofibration and the second being a fibration.
6. For any $A \twoheadrightarrow B$, there exists a factorisation as in (5) such that in any diagram of the shape

$$\begin{array}{ccccc} X & \longrightarrow & Y & \longrightarrow & Z \\ \downarrow & & \downarrow & & \downarrow \\ A & \overset{\sim}{\rightarrowtail} & P_B A & \twoheadrightarrow & B \end{array}$$

we have the following: if both squares are pullback squares (which implies that $Y \to Z$ and $X \to Z$ are fibrations), then $X \to Y$ is an acyclic cofibration.

Remark 8.2.2. From the above definition, it follows that *every* morphism factors as an acyclic cofibration followed by a fibration. Shulman's proof [Shu15, Lemma 2.4], a translation of the proof by Gambino and Garner [GG08] into category theory, relies on the fact that every morphism $A \to \mathbf{1}$ is a fibration ("all objects are fibrant") by definition.

The example of a type theoretic fibration category that we mainly have in mind is [Shu15, Example 2.9], the category of contexts of a dependent type theory with a unit type, Σ- and Π-types, and identity types. The unit, Σ- and Π-types are required to satisfy judgmental η-rules. Because of these η-rules, we do not need to talk about contexts; we can view every object of the category as a nested Σ-type with some finite number of components. Of course, the terminal object is the unit type. The subset of fibrations is the closure of the projections under isomorphisms. One nice property is that the η-rules also imply that we can assume that all fibrations are a projection of the form $\big(\Sigma\,(x : X)\,.\,Y(x)\big) \twoheadrightarrow X$. Pullbacks correspond to substitutions, and the partial functor Π_g comes from dependent function types. For any fibration $f : A \twoheadrightarrow B$, the factorisation in item (5) can be obtained using the intensional identity type: if B is the unit type, then the factorisation can be written as $A \overset{\sim}{\twoheadrightarrow} \big(\Sigma\,((x, y) : A \times A)\,.\,x = y\big) \twoheadrightarrow A \times A$, and similar otherwise (see [GG08]). The acyclic cofibration is given by reflexivity.

It is not exactly true that a type theoretic fibration category has an intensional dependent type theory as its internal language due to the well-known issue that substitution in type theory is strictly functorial. Fortunately, coherence theorems (see e.g. [Awo14; LW15]) can be applied to solve this problem, and we do not worry about it but simply refer to Shulman's explanation [Shu15, Chapter 4]. The crux is that, disregarding these coherence issues, the syntactic category of the dependent type theory with unit, Σ, Π, and identity types is essentially the initial type theoretic fibration category. This means that we can use type-theoretic constructions freely (as long as they can be performed using unit, Π, Σ, and identity types); and we will exploit this heavily. For example, the same notion of function extensionality and type equivalence $A \simeq B$ can be defined. This means, of course, that we have to be very careful with the terminology. We call a morphism that is an equivalence in the type-theoretic sense a *homotopy equivalence*, while an *isomorphism* is really an isomorphism in the usual categorical sense. Note that any isomorphism is not only a fibration by definition, but it is automatically an acyclic cofibration, and acyclic cofibrations are further automatically homotopy equivalences.

Definition 8.2.3 (acyclic fibration)**.** We say that a morphism is an *acyclic fibration* if it is a fibration and a homotopy equivalence.

An important property to record is that acyclic fibrations are stable under pullback [Shu15, Corollary 3.12].

Inverse categories and Reedy fibrant diagrams For objects x and y of a category, write $y \prec x$ if y receives a nonidentity morphism from x (and $y \preceq x$ if $y \prec x$ or $y \equiv x$). A category $\mathfrak{I}$ is called an *inverse category* (also sometimes called *one-way category*) if the relation $\prec$ is well-founded. In this case, the *ordinal rank* of an object x in $\mathfrak{I}$ is defined by

$$\rho(x) :\equiv \sup_{y \prec x} (\rho(y) + 1). \tag{8.10}$$

As described by Shulman [Shu15, Section 11], diagrams on $\mathfrak{I}$ can be constructed by well-founded induction in the following way. If x is an object, write $x /\!\!/ \mathfrak{I}$ for the full subcategory of the co-slice category $x/\mathfrak{I}$ which excludes only the identity morphism id_x. Consider the full subcategory $\{\, y \mid y \prec x \,\} \subset \mathfrak{I}$. There is the forgetful functor $U : x /\!\!/ \mathfrak{I} \to \{\, y \mid y \prec x \,\}$, mapping any $x \xrightarrow{f} y$ to the codomain y. If further A is a diagram in a type theoretic fibration category $\mathfrak{C}$ that is defined on this full subcategory, if the limit

$$M^A(x) :\equiv \mathsf{lim}_{x /\!\!/ \mathfrak{I}}(A \circ U). \tag{8.11}$$

exists, it it called the corresponding *matching object*. To extend the diagram A to the full subcategory $\{\, y \mid y \preceq x \,\} \subset \mathfrak{I}$, it is then sufficient to give an object $A(x)$ and a morphism $A(x) \to M^A(x)$. The diagram $A : \mathfrak{I} \to \mathfrak{C}$ is *Reedy fibrant* if all matching objects $M^A(x)$ exist and all the maps $A(x) \to M^A(x)$ are fibrations. We use the fact that fibrations can be regarded as "one-type projections" in the following way:

Notation 8.2.4 (Decomposition in matching object and fibre)**.** If $A : \mathfrak{I} \to \mathfrak{C}$ is a Reedy fibrant diagram, we write (as said above) $M^A(x)$ for its matching objects, and $F^A(x, m)$ for the fibre over m; that is, we have

$$A(x) \;\cong\; \Sigma\big(m : M^A(x)\big).\, F^A(x, m). \tag{8.12}$$

There is the more general notion of a *Reedy fibration* (a natural transformation between two diagrams over $\mathfrak{I}$ with certain properties), so that a diagram is Reedy fibrant if and only if the unique transformation to the terminal diagram is a Reedy fibration. Further, $\mathfrak{C}$ is said to have *Reedy $\mathfrak{I}$-limits* if any Reedy fibrant $A : \mathfrak{I} \to \mathfrak{C}$ has a limit which behaves in the way one would expect; in particular, if a natural transformation between two Reedy fibrant diagrams is levelwise a homotopy equivalence, then the map between the limits is a homotopy equivalence. We omit the exact definitions as our constructions do not require them and refer to [Shu15, Chapter 11] for the details instead. For us, it is sufficient to record that a consequence of the definition of having Reedy ω^{op}-limits is the following:

Lemma 8.2.5. *Let a type theoretic fibration category $\mathfrak{C}$ that has Reedy ω^{op}-limits be given. Suppose that*

$$F :\equiv F_0 \xleftarrow{\sim} F_1 \xleftarrow{\sim} F_2 \xleftarrow{\sim} \dots \tag{8.13}$$

is a diagram $F : \omega^{\mathrm{op}} \to \mathfrak{C}$, where all maps are acyclic fibrations. For each i, the canonical map $\mathsf{lim}(F) \to F_i$ is a homotopy equivalence.

Proof. Consider the diagram that is constantly F_i apart from a finite part,

$$G \coloneqq F_0 \twoheadleftarrow F_1 \twoheadleftarrow \ldots \twoheadleftarrow F_{i-1} \twoheadleftarrow F_i \twoheadleftarrow F_i \twoheadleftarrow F_i \ldots . \tag{8.14}$$

There is a canonical natural transformation $F \to G$, induced by the arrows in F, which is a Reedy fibration and levelwise an acyclic fibration. It follows directly from the precise definition of Reedy limits [Shu15, Definition 11.4] that the induced map between the limits $\mathsf{lim}(F) \to F_i$ is a fibration and a homotopy equivalence. □

For later, we further record the following:

Lemma 8.2.6. *If $A : I \to \mathfrak{C}$ is Reedy fibrant, then so is $A \circ U : x/I \to \mathfrak{C}$.*

Proof. This is due to the fact that for a (nonidentity) morphism $k : x \to y$ in $\mathfrak{I}$ the categories $k /\!\!/ (x /\!\!/ \mathfrak{I})$ and $y /\!\!/ \mathfrak{I}$ are isomorphic. This argument is already used by Shulman ([Shu15, Lemma 11.8]). □

An inverse category $\mathfrak{I}$ is *admissible* for $\mathfrak{C}$ if $\mathfrak{C}$ has all Reedy $(x /\!\!/ \mathfrak{I})$-limits. If $\mathfrak{I}$ is finite, then any type theoretic fibration category has Reedy $\mathfrak{I}$-limits by [Shu15, Lemma 11.8]. From the same lemma, it follows that for all constructions that we are going to do, it will be sufficient if $\mathfrak{C}$ has Reedy ω^{op}-limits. Further, in all our cases of interest, all co-slices of $\mathfrak{I}$ are finite, and $\mathfrak{C}$ is automatically admissible.

Because of the above, let us fix the following:

Notation 8.2.7. For the rest of this chapter, let $\mathfrak{C}$ be a type theoretic fibration category with Reedy ω^{op}-limits, which further satisfies function extensionality. We refer to the objects of $\mathfrak{C}$ (which are by definition always fibrant) as *types*. Let us further introduce the term *tame category*. We say that an inverse category is a *tame category* if all co-slices $x/\mathfrak{I}$ are finite (which implies that $\rho(x)$ is finite for all objects x) and, for all n, the set of objects at "level" n, that is $\{\, x \in \mathfrak{I} \mid \rho(x) \equiv n \,\}$, is finite. The important property is that a tame category $\mathfrak{I}$ is admissible for $\mathfrak{C}$, and that $\mathfrak{C}$ has Reedy $\mathfrak{I}$-limits. Thus, tame categories make it possible to perform constructions without worrying whether required limits exist, and we will not be interested in any non-tame inverse categories.

8.3 Subdiagrams

Let $\mathfrak{I}$ be a tame category. We are interested in full subcategories of $\mathfrak{I}$, and we mean "subcategory" in the strict sense that the set of objects is a subset of the set of objects of $\mathfrak{I}$. We say that a full subcategory J of $\mathfrak{I}$ is downwards closed if, for any pair x, y of objects in $\mathfrak{I}$ with $y \prec x$, if x is in J, then so is y. The full downwards closed subcategories of $\mathfrak{I}$ always form a poset (a partially ordered set) $\mathsf{Sub}(\mathfrak{I})$, with an arrow $J \to J'$ if J' is a subcategory of J. Again, "subcategory" is to be understood in the set-theoretic sense. In particular, we do not identify

isomorphic subcategories (since their objects will in general be different, in other words, the isomorphism will not commute with the embeddings into $\mathfrak{I}$).

It is easy to see that the poset $\mathsf{Sub}(\mathfrak{I})$ has all limits and colimits. For example, given downwards closed full subcategories J and J', their product is given by taking the union of their sets of objects. We therefore write $J \cup J'$. Dually, coproducts are given by intersection and we can write $J \cap J'$. An object x of $\mathfrak{I}$ generates a subcategory $\{\, y \mid y \preceq x \,\}$, for which we write $\overline{x}$.

If $A : \mathfrak{I} \to \mathfrak{C}$ is a Reedy fibrant diagram and $\mathfrak{C}$ has Reedy $\mathfrak{I}$-limits, we can consider the functor

$$\mathsf{lim}_{\_} A : \mathsf{Sub}(\mathfrak{I}) \to \mathfrak{C} \tag{8.15}$$

which maps any downwards closed full subcategory $J \subseteq \mathfrak{I}$ to $\mathsf{lim}_J A$, the Reedy limit of A restricted to J.

Lemma 8.3.1. *Let $\mathfrak{I}$ be a tame category and J, K two downwards closed subcategories of $\mathfrak{I}$. Then, the functor $\mathsf{lim}_{\_} A$ maps the pullback square*

$$\begin{array}{ccc} J \cup K & \longrightarrow & K \\ \downarrow & & \downarrow \\ J & \longrightarrow & J \cap K \end{array}$$

in $\mathsf{Sub}(\mathfrak{I})$ to a pullback square in $\mathfrak{C}$.

Proof. For an object X, a cone $X \to A|_{J \cup K}$ corresponds to a pair of two cones, $X \to A|_J$ and $X \to A|_K$, which coincide on $J \cap K$. □

Lemma 8.3.2. *Under the same assumptions as before, the functor $\mathsf{lim}_{\_} A$ maps all morphisms to fibrations. In other word, if K is a downwards closed subcategory of the inverse category J, then*

$$\mathsf{lim}_J A \twoheadrightarrow \mathsf{lim}_K A \tag{8.16}$$

is a fibration.

Proof. We only need to consider the case that J has exactly one object that K does not have, say $J \equiv K + x$, because the composition of fibrations is a fibration (this is true even for "infinite compositions", with the same short proof as Lemma 8.2.5). Further, we may assume that all objects of J are predecessors of x, i.e. we have $\overline{x} \equiv J$; otherwise, we could view $J \to K$ as a pullback of $\overline{x} \to \overline{x} - x$ and apply Lemma 8.3.1.

The cone $\mathsf{lim}_K A \to A|_K$ gives rise to a cone $\mathsf{lim}_K A \to (A \circ U)|_{x/\!\!/K}$ (the morphism into $x \xrightarrow{f} y$ is given by the morphism into y), and we thereby get a morphism $m : \mathsf{lim}_K A \to M^A(x)$. If we pull the fibration $A(x) \twoheadrightarrow M^A(x)$ back along the morphism m, we get a fibration $P \twoheadrightarrow \mathsf{lim}_K A$, and it is easy to see that $P \cong \mathsf{lim}_J A$. □

Remark 8.3.3. From the above proof, we also get a description of how the fibration $\mathsf{lim}_{K+x}A \twoheadrightarrow \mathsf{lim}_K A$ looks like in type-theoretic notation. It can be written as

$$\Sigma\,(k : \mathsf{lim}_K A)\,.\,F^A(x, m(k)) \twoheadrightarrow \mathsf{lim}_K A. \tag{8.17}$$

This remains true even if not all objects in J are predecessors of x.

8.4 Equality Diagrams

Given any tame inverse category $\mathfrak{I}$ and a fixed type B in $\mathfrak{C}$, the diagram $\mathfrak{I} \to \mathfrak{C}$ that is constantly B is, in general, not Reedy fibrant. Fortunately, the axioms of a type theoretic fibration category allow us to define a *fibrant replacement* (see, for example, Hoveys textbook [Hov07]). We call the resulting diagram, which we construct explicitly, the *equality diagram* of B over $\mathfrak{I}$. We define by simultaneous induction:

1. a diagram $\mathcal{E} : \mathfrak{I} \to \mathfrak{C}$, the *equality diagram*
2. a cone $\eta : B \to \mathcal{E}$ (i.e. a natural transformation from the functor that is constantly B to $\mathcal{E}$)
3. a diagram $M^{\mathcal{E}} : \mathfrak{I} \to \mathfrak{C}$ (the diagram of matching objects)
4. an auxiliary cone $\tilde{\eta} : B \to M^{\mathcal{E}}$.
5. a natural transformation $\iota : \mathcal{E} \to M^{\mathcal{E}}$

such that $\iota \circ \eta$ equals $\tilde{\eta}$.

Assume that i is an object in $\mathfrak{I}$ such that the five components are defined for all predecessors of i. This is in particular the case if i has no predecessors. We define the matching object $M_i^{\mathcal{E}} :\equiv \mathsf{lim}_{i/\!\!/\mathfrak{I}}\mathcal{E}$ as discussed in Section 8.2. The universal property of this limit yields

- for every non-identity morphism $f : i \to j$, an arrow $\overline{f} : M_i^{\mathcal{E}} \to \mathcal{E}_j$, which lets us define $M^{\mathcal{E}}(f)$ to be $\iota_j \circ \overline{f}$; and
- an arrow $\tilde{\eta}_i : B \to M_i^{\mathcal{E}}$ such that, for every non-identity $f : i \to j$ as in the first point, we have that $\overline{f} \circ \tilde{\eta}_i$ equals η_j.

We further define $\mathcal{E}$ on objects by

$$\mathcal{E}_i :\equiv \Sigma\left(m : M_i^{\mathcal{E}}\right).\,\Sigma\,(x : B)\,.\,\tilde{\eta}_i(x) = m. \tag{8.18}$$

This allows us to choose the canonical projection map for ι_i, and we can define $\mathcal{E}$ on non-identity morphisms by

$$\mathcal{E}(f) :\equiv \overline{f} \circ \iota_i. \tag{8.19}$$

Finally, we set

$$\eta_i(x) :\equiv (\tilde{\eta}_i(x), x, \mathsf{refl}_{\tilde{\eta}_i(x)}). \tag{8.20}$$

By construction, η, $\tilde{\eta}$, and ι satisfy the required naturality conditions.

Lemma 8.4.1. *For all $i : \mathfrak{I}$, the morphism $\eta_i : B \to \mathcal{E}_i$ is a homotopy equivalence.*

Proof. This is due to the fact that

$$\begin{aligned} \mathcal{E}_i &\equiv \Sigma\left(m : M_i^{\mathcal{E}}\right).\, \Sigma\left(x : B\right).\, \tilde{\eta}_i(x) = m \\ &\simeq \Sigma\left(x : B\right).\, \Sigma\left(m : M_i^{\mathcal{E}}\right).\, \tilde{\eta}_i(x) = m \\ &\simeq B, \end{aligned} \tag{8.21}$$

where the last step uses that the last two components have the form of a singleton. □

The proceeding lemma tells us that $\mathcal{E}$ is levelwise homotopy equivalent to the constant diagram. The crux is that, unlike the constant diagram, $\mathcal{E}$ is Reedy fibrant by construction, i.e. a *fibrant replacement* in the usual terminology of model category theory.

Lemma 8.4.2. *For all morphisms f in the category $\mathfrak{I}$, the fibration $\mathcal{E}(f)$ is a homotopy equivalence.*

Proof. If $f : i \to j$ is a morphism in $\mathfrak{I}$, we have $\mathcal{E}(f) \circ \eta_i \equiv \eta_j$ due to the naturality of η. The claim than follows by Lemma 8.4.1 as homotopy equivalences satisfy "2-out-of-3". □

8.5 The Equality Semisimplicial Type

Let Δ_+ be the category of non-zero finite ordinals and strictly increasing maps between them. We write $[\mathsf{k}]$ for the objects, $[\mathsf{k}] \equiv \{\, 0, 1, \ldots, k \,\}$, and $[\mathsf{k}] \twoheadrightarrow [\mathsf{m}]$ for the hom-sets. We can now turn to our main case of interest, which is the tame category $\mathfrak{I} \equiv \Delta_+^{\mathrm{op}}$. In this case, we call $\mathcal{E}$ the *equality semi-simplicial type* of the (given) type B. We could write down the first few values of $M^{\mathcal{E}}_{[\mathsf{n}]}$ and $\mathcal{E}_{[\mathsf{n}]}$ explicitly. However, these type expressions would look rather bloated. More revealing might be the homotopically equivalent presentation in Figure 8.2.

$$
\begin{array}{lcl}
M_{[0]} & \equiv & \mathbf{1} \\
\mathcal{E}_{[0]} & \simeq & B \\
M_{[1]} & \simeq & B \times B \\
\mathcal{E}_{[1]} & \simeq & \Sigma\left(b^1, b^2 : B\right).\, b^1 = b^2 \\
M_{[2]} & \simeq & \Sigma\left(b^1, b^2, b^3 : B\right).\, (b^1 = b^2) \times (b^2 = b^3) \times (b^1 = b^3) \\
\mathcal{E}_{[2]} & \simeq & \Sigma\left(b^1, b^2, b^3 : B\right).\, \Sigma\left(p : b^1 = b^2\right).\, \Sigma\left(q : b^2 = b^3\right).\, \Sigma\left(r : b^1 = b^3\right).\, p \cdot q = r.
\end{array}
$$

Figure 8.2: The "nicer" formulation of the equality semi-simplicial type

We think of $\mathcal{E}_{[0]}$ as the type of points, $\mathcal{E}_{[1]}$ as the type of lines (between two points), and of $\mathcal{E}_{[2]}$ as the type of triangles (with its faces). The "boundary" of a triangle, as represented by $M_{[2]}$, consists of three points with three lines, and so on. In general, we think of $\mathcal{E}_{[\mathsf{n}]}$ as (the type of) n-dimensional tetrahedra, while $M^{\mathcal{E}}_{[\mathsf{n}]}$ are their "complete boundaries". In principle, we could have defined $\mathcal{E}$ in a way such that Figure 8.2 are judgmental equalities rather than only equivalences: the stated types could be completed to form a Reedy fibrant diagram. However, we do not think that this is possible using a definition that is as uniform and short as the one above. Already for $\mathcal{E}_{[3]}$, it seems unclear what the best formulation would be if we wanted to follow the presentation of Figure 8.2. In general, such a construction would most likely make use of Street's orientals [Str87].

For any $[\mathsf{n}]$, the co-slice category $[\mathsf{n}]/\Delta_+^{\mathrm{op}}$ is a poset. This is a consequence of the fact that all morphisms in Δ_+ are monic. We have the forgetful functor $U : [\mathsf{n}]/\Delta_+^{\mathrm{op}} \to \Delta_+^{\mathrm{op}}$. Further, $[\mathsf{n}]/\Delta_+^{\mathrm{op}}$ is isomorphic to the poset $\mathcal{P}_+([\mathsf{n}])$ of nonempty subsets of the set $[\mathsf{n}] \equiv \{0, 1, \ldots, n\}$, where we have an arrow between two subsets if the first is a superset of the second. The downwards closed full subcategories of $[\mathsf{n}]/\Delta_+^{\mathrm{op}}$ correspond to downwards closed subsets of $\mathcal{P}_+([\mathsf{n}])$. If S is such a downwards closed subset, we write $\mathsf{lim}_S(\mathcal{E} \circ U)$, omitting the implied functor $S \to [\mathsf{n}]/\Delta_+^{\mathrm{op}}$.

Any set $s \subseteq [\mathsf{n}]$ generates such a downwards closed set for which we write $\overline{s} :\equiv \mathcal{P}_+(s)$. For $k \in s$, we write $\overline{s}_{-k}$ for the set that we get if we remove exactly two sets from $\overline{s}$, namely s itself and the set $s - k$ (i.e. s without the element k). We call $\mathsf{lim}_{\overline{[\mathsf{n}]}_{-k}}(\mathcal{E} \circ U)$ the *k-th n-horn*.

Main Lemma 8.5.1. *For any $n \geq 1$ and $k \in [\mathsf{n}]$, call the fibration from the full n-dimensional tetrahedron to the k-th n-horn*

$$\mathsf{lim}_{\overline{[\mathsf{n}]}}(\mathcal{E} \circ U) \twoheadrightarrow \mathsf{lim}_{\overline{[\mathsf{n}]}_{-k}}(\mathcal{E} \circ U) \tag{8.22}$$

a horn-filler fibration. *All horn-filler fibrations are homotopy equivalences.*

Remark and Corollary 8.5.2 (*Types are Kan complexes*)**.** As Steve Awodey and an anonymous reviewer of [Kra14b] have pointed out to me, Main Lemma 8.5.1

can be seen as a simplicial variant of Lumsdaine's [Lum09] and van den Berg-Garner's [BG11] result that types are weak ω-groupoids. Both of these (independent) articles use Batanin's [Bat98] definition, slightly modified by Leinster [Lei02], of a weak ω-groupoid.

Let us make the construction of a simplicial weak ω-groupoid, i.e. of a Kan complex, concrete. We can do this for the assumed type theoretic fibration category $\mathfrak{C}$ as long as it is locally small (i.e. all hom-sets are sets). As before, we can without loss of generality assume that the type we want to consider lives in the empty context, i.e. is given by an object B. We can define a semi-simplicial set

$$S : \Delta_+^{\mathrm{op}} \to \mathsf{Set} \tag{8.23}$$
$$S_{[\mathsf{n}]} :\equiv \mathfrak{C}(\mathbf{1}, \mathcal{E}_{[\mathsf{n}]}). \tag{8.24}$$

For a morphism f of Δ_+^{op}, the functor S is given by simply composing with $\mathcal{E}(f)$. Note that, of course, the simplicity of (8.24) benefits from the assumption that the context is empty, and that sections of $B \twoheadrightarrow \mathbf{1}$ are the same as morphisms $\mathfrak{C}(\mathbf{1}, B)$.

Shulman's *acyclic fibration lemma* [Shu15, Lemma 3.11], applied on the result of our Main Lemma 8.5.1, gives us sections of all horn-filler fibrations. Therefore, S satisfies the Kan condition. By a result Rourke and Sanderson [RS71] (see also McClure [McC13] for a combinatorical proof), such a semi-simplicial set can be given the structure of a Kan *simplicial* set, an incarnation of a weak ω-groupoid.

To get the result that types in HoTT are Kan complexes, we simply take $\mathfrak{C}$ to be the syntactic category of HoTT, where we have to assume strict η for Π (as we do anyway) and Σ (which we can certainly do as well). In Remark 9.2.5, we will give an elegant type-theoretic definition of $\mathcal{E}$. This allows us to say very concretely that the terms of the types that we will write down form a Kan complex.

Proof of Main Lemma 8.5.1. Fix $[\mathsf{n}]$. We show more generally that, for any $s \subseteq [\mathsf{n}]$ with cardinality $|s| \geq 2$ and $k \in s$, the fibration

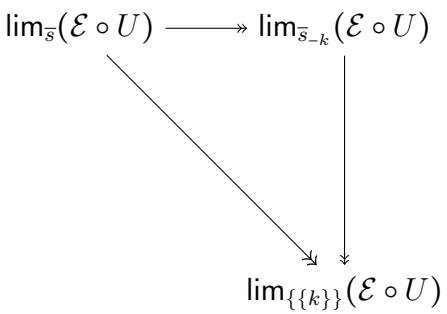
$$\mathsf{lim}_{\overline{s}}(\mathcal{E} \circ U) \twoheadrightarrow \mathsf{lim}_{\overline{s}_{-k}}(\mathcal{E} \circ U) \tag{8.25}$$

is an equivalence. The proof is performed by induction on the cardinality of s. In case s has only one element apart from k, the proof is immediate.

Let us explain the induction step. For the one-object downwards closed category $\{\{k\}\} \subset \overline{s}$ we have

$$\mathsf{lim}_{\{\{k\}\}}(\mathcal{E} \circ U) \cong \mathcal{E}_{[0]} \simeq B. \tag{8.26}$$

The inclusion $\{\{k\}\} \subseteq \overline{s}_{-k} \subset \overline{s}$ gives rise to a triangle



of fibrations. The top horizontal fibration is the one which we want to prove of that it is an equivalence. Using "2-out-of-3" and the fact that the left (diagonal) fibration is an equivalence by Lemma 8.4.2, it is sufficient to show that the right vertical fibration is an equivalence. To do this, we decompose it into $2^{|s|-1} - 1$ fibrations, each of which can be viewed as the pullback of a smaller horn-filler fibration:

Consider the set $\mathcal{P}_+(s - k)$ of those nonempty subsets of s that do not contain k. The number of those is $2^{|s|-1} - 1$. We label those sets as $\alpha_1, \alpha_2, \ldots, \alpha_{2^{|s|-1}-1}$, where the order is arbitrary with the only condition that their cardinality is non-decreasing, i.e. $i < j$ implies $|\alpha_i| < |\alpha_j|$.

We further define $2^{|s|-1}$ subsets of $\mathcal{P}_+(s)$, named $S_0, S_1, \ldots, S_{2^{|s|-1}}$. Define S_0 to be $\{\{k\}\}$. Then, define S_i to be S_{i-1} with two additional elements, namely α_i and $\alpha_i \cup \{k\}$. In this process, every element of $\mathcal{P}_+(s)$ is clearly added exactly once. In particular, $S_{2^{|s|-1}} \equiv \overline{s}$ and $S_{2^{|s|-1}-1} \equiv \overline{s}_{-k}$. Further, all S_i are downwards closed, which is easily seen to be the case by induction on i: it is the case for $i \equiv 0$, and in general, S_i contains all proper subsets of $\alpha_i \cup \{k\}$ due to the single ordering condition that we have put on the sequence (α_j).

It is easy to see that

$$S_i \;\equiv\; S_{i-1} \cup \overline{\alpha_i \cup \{k\}} \tag{8.27}$$

$$\overline{\alpha_i \cup \{k\}}_{-k} \;\equiv\; S_{i-1} \cap \overline{\alpha_i \cup \{k\}}. \tag{8.28}$$

By Lemma 8.3.1, we thus have a pullback square

$$\begin{array}{ccc} \mathsf{lim}_{S_i}(\mathcal{E} \circ U) & \twoheadrightarrow & \mathsf{lim}_{\overline{\alpha_i \cup \{k\}}}(\mathcal{E} \circ U) \\ \downarrow & & \downarrow \\ \mathsf{lim}_{S_{i-1}}(\mathcal{E} \circ U) & \twoheadrightarrow & \mathsf{lim}_{\overline{\alpha_i \cup \{k\}}_{-k}}(\mathcal{E} \circ U) \end{array}$$

For $i \leq 2^{|s|-1} - 2$, the right vertical morphism is a homotopy equivalence by the induction hypothesis. As acyclic fibrations are stable under pullback, the left vertical morphism is one as well. As the composition of equivalences is an equivalence, we conclude that

$$\mathsf{lim}_{\overline{s}_{-k}}(\mathcal{E} \circ U) \twoheadrightarrow \mathsf{lim}_{\{\{k\}\}}(\mathcal{E} \circ U) \tag{8.29}$$

is indeed an equivalence. □

Remark 8.5.3. Recall that a simplicial object $X : \Delta^{\mathrm{op}} \to \mathcal{D}$ satisfies the *Segal condition* (see [Seg68]) if the "fibration"

$$X_{[\mathsf{n}]} \to \underbrace{X_{[1]} \times_{X_{[0]}} X_{[1]} \times_{X_{[0]}} \dots \times_{X_{[0]}} X_{[1]}}_{n \text{ factors}} \tag{8.30}$$

is an equivalence. In our situation, it looks as if it was easy to check the Segal condition; more precisely, a shorter argument than the one in the proof could show that *all* the fibrations of the form (8.30)) are homotopy equivalences. Our construction with the sequence $\alpha_1, \alpha_2, \dots, \alpha_{2^{|s|-1}-1}$ seems to contain a "manual" proof of the fact that checking this form of the Segal condition would be sufficient.

8.6 Fibrant Diagrams of Natural Transformations

Let us first formalise what we mean by the "type of natural transformations between two diagrams". If I is a tame category and $D, E : I \to \mathfrak{C}$ are Reedy fibrant diagrams, the exponential $E^D : I \to \mathfrak{C}$ in the functor category $\mathfrak{C}^I$ exists and is Reedy fibrant [Shu15, Theorem 11.11] and thus has a limit in $\mathfrak{C}$. What we are interested in is the more general case that D might not be fibrant, but we also do not need any exponential. (I expect that the exponential E^D exists and is fibrant even if only E is fibrant. This would lead to an alternative representation of the same construction, but I have decided to use the less abstract one presented here as it seems to give a more direct argument.) On a more abstract level, what we want to do can be described as follows. For any downwards closed subcategory of I, we consider the exponential of D and E restricted to this subcategory, and take its limit. We basically construct approximations to the "type of natural transformations" from D to E which, in fact, corresponds to the limit of these approximations, should it exist. Fortunately, it is easy to do everything "by hand" on a very basic level.

We write $(\!|I|\!)$ for the underlying partially ordered set of I that we get if we make any two parallel arrows equal (we "truncated" all hom-sets). This makes sense even if I is not inverse, but if it is, then so is $(\!|I|\!)$. There is a canonical functor $|-|_I : I \to (\!|I|\!)$. As the objects of I are the same as those of $(\!|I|\!)$, we omit this functor when applied to an object, i.e. for $i \in I$ we write $i \in (\!|I|\!)$ instead of $|i|_I \in (\!|I|\!)$.

Definition 8.6.1 (Diagram of Natural Transformations)**.** Given an inverse category I, a diagram $D : I \to \mathfrak{C}$ and a fibrant diagram $E : I \to \mathfrak{C}$ with

$$E(i) \equiv \Sigma\big(m : M^E(i)\big).\, F^E(i, m) \tag{8.31}$$

as introduced in Notation 8.2.4, we define a fibrant diagram $N : (\!|I|\!) \to \mathfrak{C}$ together with a natural transformation

$$v : \big((N \circ |-|_I) \times D\big) \to E \tag{8.32}$$

simultaneously, where $(N \circ |-|_I) \times D$ is the functor $I \to \mathfrak{C}$ that is given by taking the product pointwise.

Assume i is an object in I. Assume further that we have defined both N and v for all predecessors of i (i.e. N is defined on $\{ x \in (\!|I|\!) \mid x \prec i \}$ and v is defined on $\{ x \in I \mid x \prec i \}$). v then gives rise to a map

$$\overline{v} : \mathsf{lim}_{\{ x \in (\!|I|\!) \mid x \prec i \}} N \times D(i) \to M^E(i). \tag{8.33}$$

Note that we have $\mathsf{lim}_{\{ x \in (\!|I|\!) \mid x \prec i \}} N \equiv \mathsf{lim}_{i /\!\!/ (\!|I|\!)} (N \circ U) \equiv M^N(i)$ since $(\!|I|\!)$ is a poset. Now, define $N(i) \equiv \Sigma\,(m : M^N(i))\,.\,F^N(i,m)$ by choosing the fibre over m to be

$$F^N(i,m) :\equiv \Pi_{d:D(i)} F^E(i, \overline{v}(m,d)). \tag{8.34}$$

This definition also gives a canonical morphism $v_i : N(i) \times D(i) \to E(i)$ which extends v.

Let us apply this construction to define the type of constant functions between types A and B in the way that we already suggested in Figure 8.1 on page 123. First, we define the functor $\mathcal{A} : \Delta_+^{\mathrm{op}} \to \mathfrak{C}$. This is the $[0]$-coskeleton of the constant diagram on A. For objects, it is simply given by

$$\mathcal{A}_{[\mathsf{k}]} :\equiv \underbrace{A \times A \times \ldots \times A}_{(k+1)\ \mathit{factors}}. \tag{8.35}$$

If we view an element of $\mathcal{A}_{[\mathsf{i}]}$ as a function $[\mathsf{i}] \to A$, for a map $f : [\mathsf{i}] \xrightarrow{+} [\mathsf{j}]$ we get $\mathcal{A}(f) : \mathcal{A}_{[\mathsf{j}]} \to \mathcal{A}_{[\mathsf{i}]}$ by composition with f. We then define the functor $\mathcal{N}_{A,B} : (\!|\Delta_+^{\mathrm{op}}|\!) \to \mathfrak{C}$ via the above construction as the "fibrant diagram of natural transformations" from $\mathcal{A}$ to $\mathcal{E}$. Note that $(\!|\Delta_+^{\mathrm{op}}|\!)$ is isomorphic to ω^{op}. Using the homotopy equivalent formulation of $\mathcal{E}$ stated in (8.2) and the definitions of const and coh of Section 8.1, we get

$$\mathcal{N}_{A,B}([0]) \simeq (A \to B) \tag{8.36}$$

as well as

$$\mathcal{N}_{A,B}([1]) \simeq \Sigma\,(f : A \to B)\,.\,\mathsf{const}_f \tag{8.37}$$

and

$$\mathcal{N}_{A,B}([2]) \simeq \Sigma\,(f : A \to B)\,.\,\Sigma\,(c : \mathsf{const}_f)\,.\,\mathsf{coh}_{f,c}. \tag{8.38}$$

We want to stress the intuition that we think of functions with an infinite tower of coherence condition by introducing the following notation:

Notation 8.6.2 ($A \xrightarrow{\omega} B$)**.** Given types A and B, we write $A \xrightarrow{\omega} B$ synonymously for $\mathsf{lim}_{(\!|\Delta_+^{\mathrm{op}}|\!)} \mathcal{N}_{A,B}$.

In the same way as we write simply $\mathcal{E}$ (and not $\mathcal{E}_B$), we usually omit the indices of $\mathcal{N}_{A,B}$ and just write $\mathcal{N}$, provided that A, B are clear from the context. This allows us write $\mathcal{N}_{[\mathsf{n}]}$ instead of $\mathcal{N}_{A,B}([\mathsf{n}])$.

Analogously to Notation 8.6.2, let us write the following:

Notation 8.6.3 ($A \xrightarrow{[\mathsf{n}]} B$)**.** Given types A and B and a (usually finite) number n, we write $A \xrightarrow{[\mathsf{n}]} B$ synonymously for $\mathcal{N}_{[\mathsf{n}]}$. To enable a uniform presentation, we define $A \xrightarrow{[-1]} B$ to be the unit type.

We are now able to make the main goal, as outlined in Section 8.1.3, precise: we will construct a function $(\|A\| \to B) \to (A \xrightarrow{\omega} B)$ and prove that it is a homotopy equivalence. For now, let us record that we can get a function $B \to (A \xrightarrow{\omega} B)$. In the following definition, we use the cones $\eta : B \to \mathcal{E}$ and $\tilde{\eta} : B \to M^{\mathcal{E}}$ from Section 8.4.

Definition 8.6.4 (Canonical function $s : B \to (A \xrightarrow{\omega} B)$)**.** Define a cone $\gamma : B \to \mathcal{N}$ which maps $b : B$ to the function that is "judgmentally constantly b", in the following way. First, notice that the matching object $M^{\mathcal{N}}_{[\mathsf{n}]}$ is simply $\mathcal{N}_{[\mathsf{n-1}]}$ (due to the fact that $(\!|\Delta_+^{\mathrm{op}}|\!)$ is a total order). Assume we have already defined the component $\gamma_{[\mathsf{n-1}]} : B \to \mathcal{N}_{[\mathsf{n-1}]}$ such that $\overline{v}(\gamma_{[\mathsf{n-1}]}(b), x) \equiv \tilde{\eta}_{[\mathsf{n}]}(b)$, with $\overline{v}$ as in (8.33), for all $x : \mathcal{A}_{[\mathsf{n}]}$. We can then define $\gamma_{[\mathsf{n}]}(b)$ by giving an element of $F^{\mathcal{N}}([\mathsf{n}], \gamma_{[\mathsf{n-1}]}(b))$, but that expression evaluates to $\Pi_{x:\mathcal{A}_{[\mathsf{n}]}} \Sigma\,(x : B)\,.\,\tilde{\eta}_{[\mathsf{n}]}(x) = \tilde{\eta}_{[\mathsf{n}]}(b)$. Thus, we can take $\gamma_{[\mathsf{n}]}(b)$ to be

$$\gamma_{[\mathsf{n}]}(b) :\equiv \left(\gamma_{[\mathsf{n-1}]}(b), \lambda z.(b, \mathsf{refl}_{\tilde{\eta}_{[\mathsf{n-1}]}(b)})\right). \tag{8.39}$$

It is straightforward to check that the condition $\overline{v}(\gamma_{[\mathsf{n}]}, x) \equiv \tilde{\eta}_{[\mathsf{n+1}]}(b)$ is preserved. Define the function $s : B \to (A \xrightarrow{\omega} B)$ to be $\mathsf{lim}_{(\!|\Delta_+^{\mathrm{op}}|\!)}\gamma$, the arrow that is induced by the universal property of the limit.

8.7 Extending Semi-Simplicial Types

In this section, we first define the category $\mathfrak{d}_+$. We can then view $\mathfrak{d}_+^{\mathrm{op}}$ as an extension of Δ_+^{op}, as Δ_+^{op} can be embedded into $\mathfrak{d}_+^{\mathrm{op}}$, and this embedding has a retraction T with the property that the co-slice $c/\mathfrak{d}_+^{\mathrm{op}}$ is always isomorphic to $T(c)/\Delta_+^{\mathrm{op}}$. With the help of this category, we can describe precisely the components precisely that we want to use in our "expanding and contracting" argument. The definition of $\mathfrak{d}_+$ is motivated by the proofs of Propositions 8.1.2 and 8.1.3, and this will become clear when we show how exactly we use $\mathfrak{d}_+$, see especially Figure 8.4. In particular, we draw the connection to Proposition 8.1.3 explicitly on page 140.

Definition 8.7.1 ($\mathfrak{d}_+$)**.** Let $\mathfrak{d}_+$ be the following category. For every object $[\mathsf{k}]$ of Δ_+ (i.e. every natural number k), and every number $i \in [\mathsf{k}+\mathsf{1}]$, we have an object $c^i_{[\mathsf{k}]}$. Given objects $c^i_{[\mathsf{k}]}$ and $c^j_{[\mathsf{m}]}$, we define $\mathfrak{d}_+\left(c^i_{[\mathsf{k}]}, c^j_{[\mathsf{m}]}\right)$ to a subset of the set of maps $\Delta_+([\mathsf{k}], [\mathsf{m}])$. It is given by

$$\mathfrak{d}_+\left(c^i_{[\mathsf{k}]}, c^j_{[\mathsf{m}]}\right) :\equiv \{\, f : [\mathsf{k}] \twoheadrightarrow [\mathsf{m}] \mid \alpha(k, m, i, j) \,\} \tag{8.40}$$

where the condition α is defined as

$$\alpha(k,m,i,j) :\equiv \begin{cases} f(x) \equiv x \text{ for all } x < i, \text{ and } f(x) > x \text{ for all } x \geq i & \text{if } i < j \\ f(x) \equiv x \text{ for all } x < i & \text{if } i \equiv j \\ \bot & \text{if } i > j. \end{cases} \tag{8.41}$$

What will be useful for us is the opposite category $\mathfrak{d}_+^{\mathrm{op}}$. A part of it, namely the subcategory $\left\{ c^i_{[\mathsf{k}]} \in \mathfrak{d}_+^{\mathrm{op}} \;\middle|\; k \leq 3 \right\}$, can be pictured as shown in Figure 8.3. We only draw the "generating" arrows $c^j_{[\mathsf{m}+1]} \to c^i_{[\mathsf{m}]}$. The idea is that the full subcategory

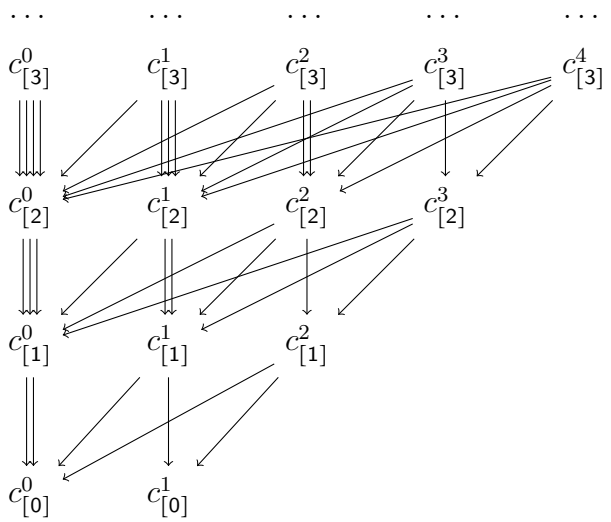


Figure 8.3: The category $\mathfrak{d}_+^{\mathrm{op}}$

of objects $c^0_{[\mathsf{m}]}$ is exactly Δ_+, and that every object $c^i_{[\mathsf{m}]}$ in $\mathfrak{d}_+$ receives exactly one arrow for every $[\mathsf{k}] \twoheadrightarrow [\mathsf{m}]$. We make this precise as follows:

Lemma 8.7.2. *The canonical embedding $\Delta_+^{\mathrm{op}} \hookrightarrow \mathfrak{d}_+^{\mathrm{op}}$, defined by $[\mathsf{m}] \mapsto c^0_{[\mathsf{m}]}$, has a retraction*

$$T : \mathfrak{d}_+^{\mathrm{op}} \to \Delta_+^{\mathrm{op}} \tag{8.42}$$

$$T(c^j_{[\mathsf{m}]}) :\equiv [\mathsf{m}] \tag{8.43}$$

and, for all objects $c^j_{[\mathsf{m}]}$ in $\mathfrak{d}_+^{\mathrm{op}}$, the functor that T induces on the co-slice categories

$$c^j_{[\mathsf{m}]}/\mathfrak{d}_+^{\mathrm{op}} \;\to\; [\mathsf{m}]/\Delta_+^{\mathrm{op}} \tag{8.44}$$

is an isomorphism of categories.

Proof. It is clear that $\Delta_+^{\mathrm{op}} \hookrightarrow \mathfrak{d}_+^{\mathrm{op}} \xrightarrow{T} \Delta_+^{\mathrm{op}}$ is the identity on Δ_+^{op}. For any $c^j_{[\mathsf{m}]}$, fix an object $[\mathsf{k}]$ in Δ_+^{op} and take a morphism $f : [\mathsf{k}] \twoheadrightarrow [\mathsf{m}]$. There is exactly one i

such that the condition $\alpha(k, m, i, j)$ in (8.41) is fulfilled. This proves the second claim. □

Let us extend the functor $\mathcal{A} : \Delta_+^{\mathrm{op}} \to \mathfrak{C}$ (see Section 8.6) to the whole category $\mathfrak{d}_+^{\mathrm{op}}$. Assume that a type A is given. We want to define a diagram $\widehat{\mathcal{A}}$ that extends $\mathcal{A}$. This corresponds to the point where, in Section 8.1, we had assumed that a point $\mathfrak{a}_\mathsf{o} : A$ was given, in other words, we had added $(\mathfrak{a}_\mathsf{o} : A)$ to the context. We do the same here. Categorically, this means we work in the slice over A. The slice category $\mathfrak{C}/A$ is not necessarily a type theoretic fibration category as not all morphisms $B \to A$ are fibrations, but we can simply restrict ourselves to those that are. Shulman denotes this full subcategory of $\mathfrak{C}/A$ by $(\mathfrak{C}/A)_\mathsf{f}$. The diagram that we define is thus a functor

$$\widehat{\mathcal{A}} : \mathfrak{d}_+^{\mathrm{op}} \to (\mathfrak{C}/A)_\mathsf{f} \,. \tag{8.45}$$

In order to be closer to the type-theoretic notation and to hopefully increase readability, we write objects of $(\mathfrak{C}/A)_\mathsf{f}$ simply as $B(\mathfrak{a}_\mathsf{o})$ if they are of the form $\Sigma\,(a : A)\,.\,B(a) \twoheadrightarrow A$. This uses that we can do the whole construction fibrewise, i.e. that we can indeed assume a fixed but arbitrary $\mathfrak{a}_\mathsf{o} : A$ "in the context". Of course, objects in $(\mathfrak{C}/A)_\mathsf{f}$ of the form $A \times B \twoheadrightarrow A$ are simply denoted by B.

Using this notation, we define $\widehat{\mathcal{A}}$ on objects by

$$\widehat{\mathcal{A}}(c^j_{[\mathsf{m}]}) :\equiv \underbrace{A \times A \times \ldots \times A}_{(m+1-j)\text{ factors}}, \tag{8.46}$$

for which we simply write A^{m+1-j}. Given $c^j_{[\mathsf{m}]} \xrightarrow{f} c^i_{[\mathsf{k}]}$ in $\mathfrak{d}_+^{\mathrm{op}}$, we thus need to define a map $\widehat{\mathcal{A}}(f) : A^{m+1-j} \to A^{k+1-i}$. As in the definition of $\mathcal{A}$, the map $f : [\mathsf{k}] \twoheadrightarrow [\mathsf{m}]$ gives rise to a function $\overline{f} : A^{m+1} \to A^{k+1}$ by "composition". We define $\widehat{\mathcal{A}}(f)$ as the composite

$$\begin{array}{c} A^{m+1-j} \\ \Big\downarrow \vec{a} \mapsto (\underbrace{\mathfrak{a}_\mathsf{o}, \mathfrak{a}_\mathsf{o}, \ldots, \mathfrak{a}_\mathsf{o}}_{j\text{ times }\mathfrak{a}_\mathsf{o}}, \vec{a}) \\ A^j \times A^{m+1-j} \\ \Big\downarrow \overline{f} \\ A^i \times A^{k+1-i} \\ \Big\downarrow \mathsf{snd} \\ A^{k+1-i} \end{array}$$

We have a diagram $\mathcal{E} \circ T : \mathfrak{d}_+^{\mathrm{op}} \to \mathfrak{C}$, which we can (pointwise) pull back along $A \twoheadrightarrow \mathbf{1}$, which gives us a diagram that we call $\widehat{\mathcal{E}} : \mathfrak{d}_+^{\mathrm{op}} \to (\mathfrak{C}/A)_\mathsf{f}$. This diagram is Reedy fibrant. With the construction of Section 8.6, we can define

$\widehat{\mathcal{N}} : (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \to (\mathfrak{C}/A)_{\mathsf{f}}$ to be the "fibrant diagram of natural transformations" from $\widehat{\mathcal{A}}$ to $\widehat{\mathcal{E}}$.

We can picture $\widehat{\mathcal{N}}$ on the subcategory $\left\{ c^j_{[\mathsf{m}]} \in (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \;\middle|\; m \leq 2 \right\}$ as shown in Figure 8.4. For readability, we use the homotopy equivalent representation of the values of $\mathcal{E}$ as shown in Figure 8.2. Further, we only write down the values of $F^{\widehat{\mathcal{E}}}$ (i.e. the fibres) instead of the full expression $\widehat{\mathcal{E}}(c^j_{[\mathsf{m}]}) \equiv \Sigma\left(t : M^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]})\right).\, F^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}, t)$. This means that the fibration $\mathsf{const}_f \twoheadrightarrow (f : A \to B)$ from Figure 8.4 stands for the projection $\Sigma\,(f : A \to B).\,\mathsf{const}_f \twoheadrightarrow (A \to B)$. The reader is invited to make a comparison with Proposition 8.1.3. Recall that, in the proof of Proposition 8.1.3, we have started with the component f_1. In the "expanding" part, we have added the pair of f and c_1, which (together) form a contractible type, as well as the pair of c and d_1, and c_2 and d_3. We have also used that the types of d and d_2 are, in the presence of the other components, contractible. Then, in the "retracting" part, we have used that the types of d_3 and d_1 are contractible, and that c_1 and d_2, as well as f_1 and c_2, form pairs of two other contractible types.

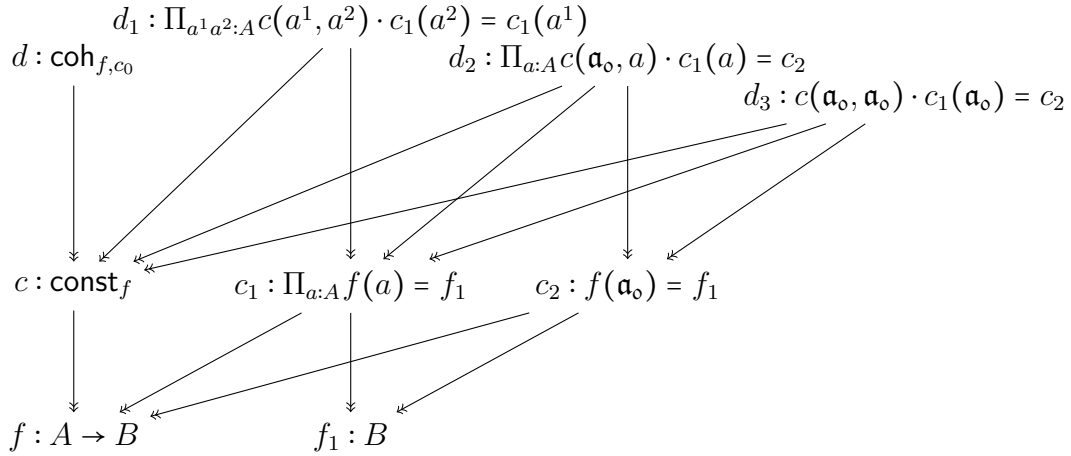


Figure 8.4: The diagram $\widehat{\mathcal{N}}$ in readable (homotopy equivalent) representation; only the three lowest levels (the images of $c^j_{[\mathsf{m}]}$ with $m \leq 2$) are drawn

To compare $\widehat{\mathcal{N}}$ with $\mathcal{N}$, first note that $\mathcal{N} : \Delta_+^{\mathrm{op}} \to \mathfrak{C}$ can be pulled back along $A \twoheadrightarrow \mathbf{1}$ pointwise and yields a diagram $\Delta_+^{\mathrm{op}} \to (\mathfrak{C}/A)_{\mathsf{f}}$. This diagram is identical (pointwise isomorphic) to the diagram that we get if we first pull back the diagrams $\mathcal{A}$ and $\mathcal{E}$, and then take the "fibrant diagram of natural transformations". Further, as "limits commute with limits", the limit of this diagram is, in $(\mathfrak{C}/A)_{\mathsf{f}}$, isomorphic to the pullback of $A \xrightarrow{\omega} B$ along $A \twoheadrightarrow \mathbf{1}$. It is thus irrelevant at which point in the construction we "add $(\mathfrak{a}_\mathsf{o} : A)$ to the context", i.e. at which point we switch from $\mathfrak{C}$ to the slice over A. This allows us to compare constructions in $(\mathfrak{C}/A)_{\mathsf{f}}$ and $\mathfrak{C}$, by implicitly pulling back the latter. As it is easy to see, $\widehat{\mathcal{N}}$ extends $\mathcal{N}$ in this sense (i.e. $\widehat{\mathcal{N}}(c^0_{[\mathsf{m}]})$ is the pullback of $\mathcal{N}_{[\mathsf{m}]}$ along $A \twoheadrightarrow \mathbf{1}$).

Recall that we have defined a cone $\gamma : B \to \mathcal{N}$ and an arrow $s : B \to (A \xrightarrow{\omega} B)$ in Definition 8.6.4. Exploiting that $\gamma_{[\mathsf{n}]}(b)$ was defined in a way that makes it completely independent of the "argument" $x : \mathcal{A}_{[\mathsf{n}]}$, and using Lemma 8.7.2, we can extend γ to a cone $\overline{\gamma} : B \to \widehat{\mathcal{N}}$, essentially by putting $\overline{\gamma}_{c^j_{[\mathsf{m}]}} :\equiv \gamma_{[\mathsf{m}]}$. This gives a morphism

$$\overline{s} : B \to \mathsf{lim}_{(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)}\widehat{\mathcal{N}} \tag{8.47}$$

which extends s, in the sense that (the pullback of) s is the composition

$$B \xrightarrow{\overline{s}} \mathsf{lim}_{(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)}\widehat{\mathcal{N}} \xrightarrow{\mathsf{pr}} \mathsf{lim}_{(\!|\Delta_+^{\mathrm{op}}|\!)}\mathcal{N}, \tag{8.48}$$

with pr coming from the embedding $(\!|\Delta_+^{\mathrm{op}}|\!) \hookrightarrow (\!|\mathfrak{d}_+^{\mathrm{op}}|\!)$ and the fact that the restriction of $\widehat{\mathcal{N}}$ to $\{c^0_{[\mathsf{m}]}\}$ is $\mathcal{N}$. Further, noting that $\widehat{\mathcal{N}}(c^1_{[0]})$ is canonically equivalent to B (as used in Figure 8.4), the composition

$$B \xrightarrow{\overline{s}} \mathsf{lim}_{(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)}\widehat{\mathcal{N}} \xrightarrow{\mathsf{pr}'} \widehat{\mathcal{N}}(c^1_{[0]}) \xrightarrow{\sim} B \tag{8.49}$$

is the identity on B.

8.8 The Main Theorem

The preparations of the previous sections allow us to formulate and prove our main result. We proceed analogously to our arguments for the special cases in Section 8.1: Lemma 8.8.1 and Corollary 8.8.2 show that certain fibrations are homotopy equivalences, i.e. that certain types are contractible. This is then used in Lemma 8.8.3 to perform the "expanding and contracting" argument, which shows that, if we assume a point in A, the function s from Definition 8.6.4 is an equivalence.

For the next statement, note that $\widehat{\mathcal{N}}(c^j_{[\mathsf{m}]})$ is isomorphic to $\mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}_+|\!) \;\middle|\; x \preceq c^j_{[\mathsf{m}]} \,\right\}}\widehat{\mathcal{N}}$.

Lemma 8.8.1. *The fibration*

$$\widehat{\mathcal{N}}(c^j_{[\mathsf{m}]}) \twoheadrightarrow \mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \;\middle|\; x \prec c^j_{[\mathsf{m}]}, x \neq c^{j-1}_{[\mathsf{m}-1]} \,\right\}}\widehat{\mathcal{N}} \tag{8.50}$$

is a homotopy equivalence for any m and j.

Proof. There is a single morphism in $\mathfrak{d}_+^{\mathrm{op}}\left(c^j_{[\mathsf{m}]}, c^{j-1}_{[\mathsf{m}-1]}\right)$. For the category $c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}}$ where this morphism is removed, we write $c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}} - c^{j-1}_{[\mathsf{m}-1]}$. By construction of $\widehat{\mathcal{N}}$, we have a natural transformation $v : (\widehat{\mathcal{N}} \circ |-|_{\mathfrak{d}_+^{\mathrm{op}}}) \times \widehat{\mathcal{A}} \to \widehat{\mathcal{E}}$, which gives rise to a morphism

$$w : \left(\mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \;\middle|\; x \prec c^j_{[\mathsf{m}]}, x \neq c^{j-1}_{[\mathsf{m}-1]} \,\right\}}\widehat{\mathcal{N}}\right) \times \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]}) \;\to\; \mathsf{lim}_{c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}} - c^{j-1}_{[\mathsf{m}-1]}}\, \widehat{\mathcal{E}} \circ U. \tag{8.51}$$

Consider the diagram shown in Figure 8.5, in which Q is defined to be the pullback. The right part (everything without the leftmost column) of that diagram comes

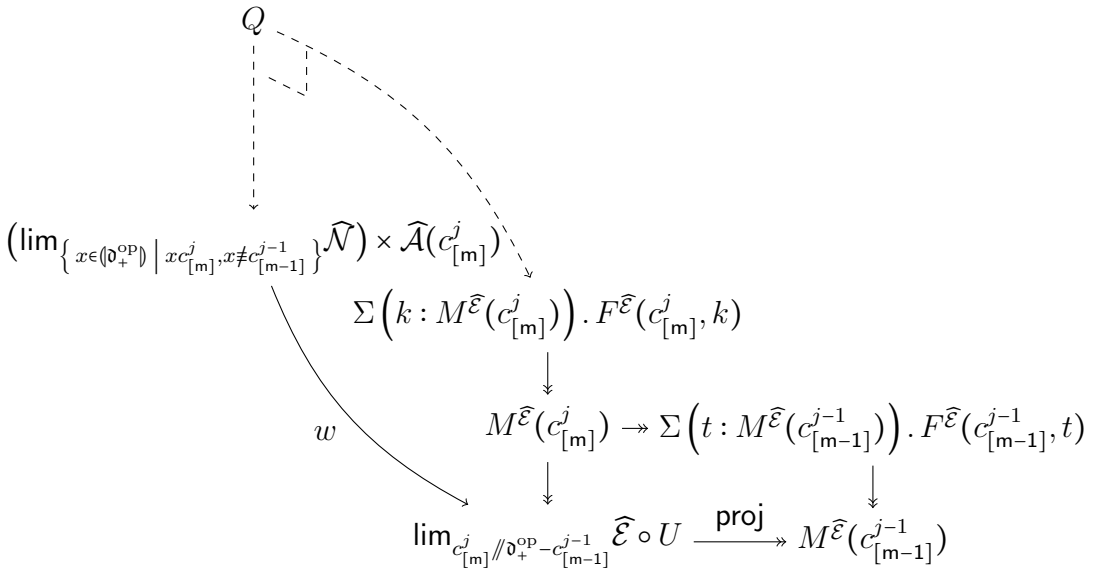


Figure 8.5: Derivation of a homotopy equivalence

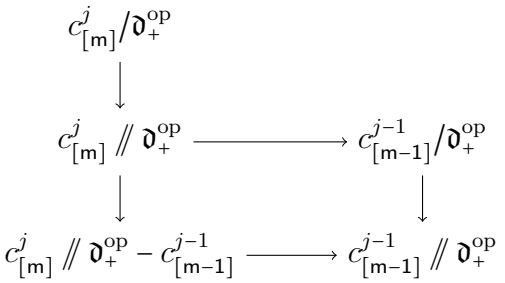


Figure 8.6: A small diagram in $\mathsf{Sub}(c^j_{[\mathsf{m}]}/\mathfrak{d}_+^{\mathrm{op}})$. This uses the principle that, in an inverse category $\mathfrak{I}$ with a morphism $k : x \to y$, the categories $k /\!\!/ (x /\!\!/ \mathfrak{I})$ and $y /\!\!/ \mathfrak{I}$ are isomorphic.

from applying the functor $\mathsf{lim}\_\widehat{\mathcal{E}}$ to the diagram in $\mathsf{Sub}(c^j_{[\mathsf{m}]}/\mathfrak{d}_+^{\mathrm{op}})$ that is shown in Figure 8.6.

In Figure 8.5, the fibration labelled proj comes of course from

$$\left(c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}} - c^{j-1}_{[\mathsf{m}-1]}\right) \supset \left(c^{j-1}_{[\mathsf{m}-1]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}}\right), \tag{8.52}$$

as shown in Figure 8.6. We give it a name solely to make referencing it easier. Our goal is to derive a representation of Q. As the right square is a pullback square by Lemma 8.3.1, we clearly must have

$$M^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}) \;\cong\; \Sigma\big(t : \mathsf{lim}_{c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}} - c^{j-1}_{[\mathsf{m}-1]}} \widehat{\mathcal{E}}\big).\, F^{\widehat{\mathcal{E}}}(c^{j-1}_{[\mathsf{m}-1]}, \mathsf{proj}(t)). \tag{8.53}$$

Doing so, we can write the top expression of the middle column as

$$\begin{aligned} & \Sigma\big(k : M^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]})\big).\, F^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}, k) \\ \simeq\ & \Sigma\big(t : \mathsf{lim}_{c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}^{\mathrm{op}}_+ - c^{j-1}_{[\mathsf{m}-1]}} \widehat{\mathcal{E}}\big).\, \Sigma\big(n : F^{\widehat{\mathcal{E}}}(c^{j-1}_{[\mathsf{m}-1]}, \mathsf{proj}(t))\big).\, F^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}, (t, n)). \end{aligned} \tag{8.54}$$

The pullback Q is thus

$$\begin{aligned} \Sigma\big(p : \mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}^{\mathrm{op}}_+|\!) \;\middle|\; x \prec c^j_{[\mathsf{m}]}, x \not\equiv c^{j-1}_{[\mathsf{m}-1]} \,\right\}} \widehat{\mathcal{N}}\big).\, \Sigma\big(a : \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]})\big). & \\ \Sigma\big(n : F^{\widehat{\mathcal{E}}}(c^{j-1}_{[\mathsf{m}-1]}, \mathsf{proj}(w(p,a)))\big). & \\ F^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}, (w(p,a), n)). & \end{aligned} \tag{8.55}$$

The composition of the two vertical fibrations in the middle column is a homotopy equivalence by Main Lemma 8.5.1 and Lemma 8.7.2. As acyclic fibrations are stable under pullback, the fibration

$$Q \twoheadrightarrow \big(\mathsf{lim}_{c^j_{[\mathsf{m}]} /\!\!/ (\!|\mathfrak{d}^{\mathrm{op}}_+|\!) - c^{j-1}_{[\mathsf{m}-1]}} N\big) \times \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]}) \tag{8.56}$$

is a homotopy equivalence as well. Function extensionality implies that a family of contractible types is contractible (i.e. that acyclic fibrations are preserved by Π), and we get that the first projection

$$\begin{array}{l} \Sigma\big(p : \mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}^{\mathrm{op}}_+|\!) \;\middle|\; x \prec c^j_{[\mathsf{m}]}, x \not\equiv c^{j-1}_{[\mathsf{m}-1]} \,\right\}} \widehat{\mathcal{N}}\big). \\ \Pi_{a : \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]})} \big(n : F^{\widehat{\mathcal{E}}}(c^{j-1}_{[\mathsf{m}-1]}, \mathsf{proj}(w(p,a)))\big) \times F^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}, (w(p,a), n)) \\ \qquad\qquad \big\downarrow \\ \qquad\qquad \mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}^{\mathrm{op}}_+|\!) \;\middle|\; x \prec c^j_{[\mathsf{m}]}, x \not\equiv c^{j-1}_{[\mathsf{m}-1]} \,\right\}} \widehat{\mathcal{N}} \end{array} \tag{8.57}$$

is an equivalence as well. The lemma is therefore shown if we can prove that the domain of the above fibration (8.57), a rather lengthy expression, is equivalent to $N(c^j_{[\mathsf{m}]})$. Our first step is to apply the distributivity law (Lemma 2.2.12) to transform this expression to

$$\begin{aligned} & \Sigma\big(p : \mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}^{\mathrm{op}}_+|\!) \;\middle|\; x \prec c^j_{[\mathsf{m}]}, x \not\equiv c^{j-1}_{[\mathsf{m}-1]} \,\right\}} \widehat{\mathcal{N}}\big). \\ & \Sigma\big(n : \Pi_{a : \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]})} F^{\widehat{\mathcal{E}}}(c^{j-1}_{[\mathsf{m}-1]}, \mathsf{proj}(w(p,a)))\big). \\ & \quad \Pi_{a : \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]})} F^{\widehat{\mathcal{E}}}(c^j_{[\mathsf{m}]}, (w(p,a), n(a))). \end{aligned} \tag{8.58}$$

When we look at the following square, in which w is the map (8.51), w' is induced by the natural transformation v in the same way as w, and proj, proj' come from

the restriction to subcategories,

$$\begin{array}{ccc} \left(\mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \;\middle|\; x < c^j_{[\mathsf{m}]}, x \neq c^{j-1}_{[\mathsf{m}-1]} \,\right\}} \widehat{\mathcal{N}}\right) \times \widehat{\mathcal{A}}(c^j_{[\mathsf{m}]}) & \xrightarrow{w} & \mathsf{lim}_{c^j_{[\mathsf{m}]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}} - c^{j-1}_{[\mathsf{m}-1]}} \widehat{\mathcal{E}} \circ U \\ \Big\downarrow \mathsf{proj}' & & \Big\downarrow \mathsf{proj} \\ \left(\mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \;\middle|\; x < c^{j-1}_{[\mathsf{m}-1]} \,\right\}} \widehat{\mathcal{N}}\right) \times \widehat{\mathcal{A}}(c^{j-1}_{[\mathsf{m}-1]}) & \xrightarrow{w'} & \mathsf{lim}_{c^{j-1}_{[\mathsf{m}-1]} /\!\!/ \mathfrak{d}_+^{\mathrm{op}}} \widehat{\mathcal{E}} \circ U \end{array} \tag{8.59}$$

we can see that it commutes due to the naturality of the natural transformation v. In particular, note that $\widehat{\mathcal{A}}$ maps the single morphism $c^j_{[\mathsf{m}]} \to c^{j-1}_{[\mathsf{m}-1]}$ to the identity on A^{i-1}. This is exactly what is needed to see that the second line of (8.58) corresponds to the "missing" component $\widehat{\mathcal{N}}(c^{j-1}_{[\mathsf{m}-1]})$ in the limit of the first line. Hence, the first and the second line can be "merged" and are equivalent to $\mathsf{lim}_{\left\{\, x \in (\!|\mathfrak{d}_+^{\mathrm{op}}|\!) \;\middle|\; x < c^j_{[\mathsf{m}]} \,\right\}} \widehat{\mathcal{N}}$, in other words, $M^{\widehat{\mathcal{N}}}(c^j_{[\mathsf{m}]})$. Comparing the third line of (8.58) with the definition of the "fibrant diagram of natural transformations" (see (8.34)), we see that (8.58) is indeed equivalent to $\widehat{\mathcal{N}}(c^j_{[\mathsf{m}]})$, as required. □

By pullback (Lemma 8.3.1 and preservation of homotopy equivalences along pullbacks), we immediately get:

Corollary 8.8.2. *Let D be a downwards closed subcategory of $\mathfrak{d}_+^{\mathrm{op}}$ which does not contain the objects $c^j_{[\mathsf{m}]}$ and $c^{j-1}_{[\mathsf{m}-1]}$, but all other predecessors of $c^j_{[\mathsf{m}]}$. The full subcategory of $\mathfrak{d}_+^{\mathrm{op}}$ which has all the objects of D and the objects $c^{j-1}_{[\mathsf{m}-1]}$, $c^j_{[\mathsf{m}]}$ (for which we write $D + c^{j-1}_{[\mathsf{m}-1]} + c^j_{[\mathsf{m}]}$) is also downwards closed and the fibration*

$$\mathsf{lim}_{D + c^{j-1}_{[\mathsf{m}-1]} + c^j_{[\mathsf{m}]}} \widehat{\mathcal{N}} \twoheadrightarrow \mathsf{lim}_D \widehat{\mathcal{N}} \tag{8.60}$$

is a homotopy equivalence. □

Corollary 8.8.2 is the crucial statement that summarises all of our efforts so far. We can use it to "add and remove" contractible Σ-components in the same way as we did it in the motivating examples (Section 8.1). More precisely, we exploit that we can group together components of $\mathfrak{d}_+^{\mathrm{op}}$ in two different ways. Our main lemma is the following:

Lemma 8.8.3. *Given types A, B, recall that we have defined $s : B \to (A \xrightarrow{\omega} B)$ in Definition 8.6.4. Assume further that we are given a point $\mathfrak{a}_\mathsf{o} : A$ (i.e. regard s as a morphism in $(\mathfrak{C}/A)_\mathsf{f}$ instead of $\mathfrak{C}$). Then, the function s is a homotopy equivalence.*

Proof. Using the point $\mathfrak{a}_\circ$, we define $\widehat{\mathcal{N}}$ and $\overline{s} : B \to \mathsf{lim}_{(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)}\widehat{\mathcal{N}}$ as before in (8.47), and consider the following:

$$\begin{array}{ccccccc} B & \xrightarrow{\overline{s}} & \mathsf{lim}_{(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)}\widehat{\mathcal{N}} & \xrightarrow{\mathsf{pr}'} & \widehat{\mathcal{N}}(c^1_{[0]}) & \xrightarrow{\sim} & B \\ & \searrow^{s} & \downarrow{\scriptstyle \mathsf{pr}} & & & & \\ & & A \xrightarrow{\omega} B & & & & \end{array} \tag{8.61}$$

The commutativity of the triangle on the left is given by (8.48). Our first goal is to show that the fibration pr' is a homotopy equivalence.

Consider the set $S :\equiv \{\, (m,j) \in \mathbb{N}^2 \mid j \text{ is even and } j \le m+1 \,\}$. A pair (m,j) is in S if and only if $c^j_{[\mathsf{m}]}$ is an object in an "odd column" of $\mathfrak{d}_+^{\mathrm{op}}$ in Figure 8.3 on page 138 (where we consider the leftmost column the "first"). Define a total order on S by letting (k,i) be smaller than (m,j) if either $k+i < m+j$ or ($k+i \equiv m+j$ and $i<j$). We represent this total order by an isomorphism $f : \mathbb{N}_+ \to S$ (where $\mathbb{N}_+$ are the positive natural numbers) which has the property that $f(n)$ is always smaller than $f(n+1)$. Write $f_1(n)$ and $f_2(n)$ for the first respectively the second component of $f(n)$.

Let us define a sequence $D_0 \subset D_1 \subset D_2 \subset D_3 \subset \ldots$ of full subcategories of $(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)$ by

$$D_0 :\equiv \{c^1_{[0]}\} \tag{8.62}$$

$$D_n :\equiv D_{n-1} + c^{f_2(n)}_{[\mathsf{f_1(n)}]} + c^{f_2(n)+1}_{[\mathsf{f_1(n)+1}]}. \tag{8.63}$$

It is easy to see that every object $c^j_{[\mathsf{m}]}$ is added exactly once, i.e. it is either $c^1_{[0]}$ or it is of the form $c^{f_2(n)}_{[\mathsf{f_1(n)}]}$ or of the form $c^{f_2(n)+1}_{[\mathsf{f_1(n)+1}]}$ for exactly one n. We have chosen the total order on S in such a way that every D_n is a downwards closed full subcategory of $(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)$. Applying Corollary 8.8.2, we get a sequence

$$\mathsf{lim}_{D_0}\widehat{\mathcal{N}} \xleftarrow{\sim} \mathsf{lim}_{D_1}\widehat{\mathcal{N}} \xleftarrow{\sim} \mathsf{lim}_{D_2}\widehat{\mathcal{N}} \xleftarrow{\sim} \mathsf{lim}_{D_3}\widehat{\mathcal{N}} \xleftarrow{\sim} \ldots \tag{8.64}$$

of acyclic fibrations. Lemma 8.2.5 then shows that the canonical map

$$\mathsf{lim}_{\mathfrak{d}_+^{\mathrm{op}}}\widehat{\mathcal{N}} \xrightarrow{\sim} \mathsf{lim}_{D_0}\widehat{\mathcal{N}} \tag{8.65}$$

is an acyclic fibration. As $\mathsf{lim}_{D_0}\widehat{\mathcal{N}}$ is simply $\widehat{\mathcal{N}}(c^1_{[0]})$, this proves that pr' is indeed a homotopy equivalence.

Next, we want to show the same about pr. We proceed very similarly. This time, we define $S' :\equiv \{\, (m,j) \in \mathbb{N}^2 \mid j \text{ is odd and } j \le m+1 \,\}$. A pair (m,j) is consequently in S' if and only if $c^j_{[\mathsf{m}]}$ is an object in an "even" column of Figure 8.4. As before, we define an isomorphism $f' : \mathbb{N}_+ \to S'$, and define a sequence $D'_0 \subset D'_1 \subset D'_2 \subset D'_3 \subset \ldots$ of full subcategories of $(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)$ by

$$D_0 :\equiv \{c^0_{[\mathsf{m}]}\} \qquad \text{(i.e. the full subcategory corresponding to } (\!|\Delta_+^{\mathrm{op}}|\!)) \tag{8.66}$$

$$D_n :\equiv D_{n-1} + c^{f_2(n)}_{[\mathsf{f_1(n)}]} + c^{f_2(n)+1}_{[\mathsf{f_1(n)+1}]}. \tag{8.67}$$

Again, every object $c^j_{[\mathsf{m}]}$ is added exactly once, and every D_n is downwards closed. Corollary 8.8.2 and Lemma 8.2.5 then tell us that $\mathsf{lim}_{(\!|\mathbb{o}^{\mathrm{op}}_+|\!)}\widehat{\mathcal{N}} \twoheadrightarrow \mathsf{lim}_{\{c^0_{[\mathsf{m}]}\}}\widehat{\mathcal{N}}$ is an acyclic fibration. Hence, pr is indeed a homotopy equivalence, as claimed.

We take another look at the diagram (8.61). The composition of the three horizontal arrows is the identity by (8.49). But homotopy equivalences satisfy "2-out-of-3", and we can conclude that $\overline{s}$ is an equivalence. Using "2-out-of-3" again, we see that s is an equivalence as well. □

This makes it easy to show the following:

Lemma 8.8.4. *If $\mathfrak{C}$ has propositional truncations, then the canonical function $s : B \to (A \overset{\omega}{\to} B)$, viewed as a morphism in $\mathfrak{C}$, is a homotopy equivalence assuming $\|A\|$. More precisely, we can construct a function*

$$\|A\| \to \mathsf{isequiv}(s) \tag{8.68}$$

in $\mathfrak{C}$.

Proof. We have shown in Lemma 8.8.3 that s is a homotopy equivalence in $(\mathfrak{C}/A)_{\mathsf{f}}$, i.e. if we pull back its domain and codomain along $A \twoheadrightarrow \mathbf{1}$. In $\mathfrak{C}$, this means that

$$\lambda(a,b).(a, s(b)) : A \times B \;\to\; A \times (A \overset{\omega}{\to} B) \tag{8.69}$$

is an equivalence, but this implies

$$A \to \mathsf{isequiv}(s). \tag{8.70}$$

The claim then follows from the ordinary universal property of the propositional truncation. □

This allows us to prove our main result:

Theorem 8.8.5 (General universal property of the propositional truncation)**.** *Let $\mathfrak{C}$ be a type theoretic fibration category that satisfies function extensionality, has propositional truncation, and Reedy ω^{op}-limits. Let A and B be two types, i.e. objects in $\mathfrak{C}$. Using the canonical function $s : B \to (A \overset{\omega}{\to} B)$ as defined in Definition 8.6.4, we can construct a function*

$$(\|A\| \to B) \;\to\; (A \overset{\omega}{\to} B), \tag{8.71}$$

and this function is a homotopy equivalence.

Proof. From Lemma 8.8.4 we can conclude, just as in the special cases in Section 8.1, that

$$(\|A\| \to B) \;\to\; \big(\|A\| \to (A \overset{\omega}{\to} B)\big) \tag{8.72}$$

$$f \;\mapsto\; \lambda x.s(f(x)) \tag{8.73}$$

is a homotopy equivalence.

This is not yet what we aim for. We need a statement corresponding to the infinite case of Lemma 8.1.1, i.e. we need to prove that $\|A\| \to (A \xrightarrow{\omega} B)$ is equivalent to $A \xrightarrow{\omega} B$. To do this, we consider the diagram $\mathcal{P} : (\!|\Delta_+^{\mathrm{op}}|\!) \to \mathfrak{C}$, defined on objects by

$$\mathcal{P}_{[\mathsf{k}]} :\equiv \|A\| \to \mathcal{N}_{[\mathsf{k}]}, \tag{8.74}$$

and on morphisms by

$$\mathcal{P}(g) :\equiv \lambda(h : \|A\| \to \mathcal{N}_{[\mathsf{k}]}).\lambda x.\mathcal{N}(g)(h(x)). \tag{8.75}$$

Paolo Capriotti has pointed out that $\mathcal{P}$ is Reedy fibrant, and this is a crucial observation. As $\mathcal{P}$ is defined over a poset, it is enough to show that (8.75) is a fibration for every g. Our argument is the following: The maps in both directions which are used to prove the distributivity law (Lemma 2.2.12) are *strict* inverses, i.e. their compositions (in both orders) are judgmentally equal to the identities. This means that every $\mathcal{P}_i$ is isomorphic to a Σ-type, where we "distribute" $\|A\|$ over the components. From this representation, it is clear that $\mathcal{P}(g)$ is always a fibration, as fibrations are closed under composition with isomorphisms.

Because of Lemma 8.1.1 (and the fact that the equivalence there can be defined uniformly), there is a natural transformation $\kappa : \mathcal{P} \to \mathcal{N}$ which is levelwise a homotopy equivalence. By the definition of $\mathfrak{C}$ having Reedy ω^{op}-limits, the resulting arrow between the two limits, that is

$$\mathsf{lim}_{(\!|\Delta_+^{\mathrm{op}}|\!)}(\kappa) : \big(\|A\| \to (A \xrightarrow{\omega} B)\big) \to (A \xrightarrow{\omega} B), \tag{8.76}$$

is a homotopy equivalence as well. To conclude, we simply compose (8.73) and (8.76). □

8.9 Finite Cases

If B is an n-type for some finite fixed number n, the higher coherence conditions should intuitively become trivial. This is obvious for the representation of $\mathcal{N}$ and $\mathcal{E}$ given in Figures 8.2 and 8.4, although admittedly not for our actual definition of $\mathcal{E}$ in Section 8.4 (and the corresponding definition of $\mathcal{N}$ and $\widehat{\mathcal{N}}$) where it requires a little more thought. This is our main goal for this section. After this, it will be easy to see that the universal properties of the propositional truncation with an n-type as codomain can be formulated and proved in (standard) homotopy type theory.

What we first need is an instance of a more general principle, and, as it often happens, the more general principle is easier to convince ourselves of. So, let us formulate the entirely straightforward following statement:

Lemma 8.9.1 ([Uni13, Theorem 4.7.7])**.** *Assume X is a type and $Y_1, Y_2 : X \to \mathcal{U}$ are two families. If there is an equivalence $f : \Sigma\,(x : X)\,.\,Y_1(x) \to \Sigma\,(x : X)\,.\,Y_2(x)$ such that f is the identity on the first component, then, for all $x : X$, the types $Y_1(x)$ and $Y_2(x)$ are equivalent.* □

This allows us to reverse the statement that "singletons are contractible":

Lemma 8.9.2. *Assume a type A, a family $B : A \to \mathcal{U}$, and a second family $C : (\Sigma\,(a : A)\,.\,B) \to \mathcal{U}$ are given. The following are logically equivalent:*

(1) For any $a : A$, there is a point $b_a : B(a)$ and an equivalence $C(a, b) \simeq (b = b_a)$.

(2) The canonical projection

$$\mathsf{fst} : \big(\Sigma\,(a : A)\,.\,\Sigma\,(b : B(a))\,.\,C(a, b)\big) \to A \tag{8.77}$$

is an equivalence.

Note that, in the special case that A is the unit type, this means that a type of the form $\Sigma\,(b : B)\,.\,C(b)$ is contractible if and only if there is some point $b_0 : B$ and an equivalence $C(b) \simeq (b = b_0)$.

Proof. The direction (1)⇒(2) is an obvious consequence from the contractibility of singletons. The other direction is (to the best of my knowledge) not very well-known. Thus, we assume (2). This implies that, for any $a : A$, the type $\Sigma\,(b : B(a))\,.\,C(a, b)$ is contractible; and this gives us the required family of b_a's.

From the assumption and contractibility of singletons, we further get

$$(\Sigma\,(a : A)\,.\,\Sigma\,(b : B(a))\,.\,C(a, b)) \;\simeq\; (\Sigma\,(a : A)\,.\,\Sigma\,(b : B(c))\,.\,(b = b_a))\,. \tag{8.78}$$

Let us use associativity of nested Σ-operators (which we have done implicitly before, but we do it explicitly here), and write this equivalence as

$$(\Sigma\,((a, b) : \Sigma\,(A)\,.\,B)\,.\,C(a, b)) \;\simeq\; (\Sigma\,((a, b) : \Sigma\,(A)\,.\,B)\,.\,(b = b_a))\,. \tag{8.79}$$

We set $X :\equiv \Sigma\,(a : A)\,.\,B(a)$ and $Y_1(x) :\equiv C(x)$ respectively $Y_2(a, b) :\equiv (b = b_a)$ and apply Lemma 8.9.1. The equivalence (8.79) preserves the first component: this is obvious for the A-part and easy to see for the $B(a)$-part, as $\Sigma\,(b : B(a))\,.\,C(a, b)$ and $\Sigma\,(b : B(a))\,.\,(b = b_a)$ are contractible. □

We are now ready to show that, in the case of n-types, the higher "fillers for complete boundaries" become homotopically simpler and simpler, and finally trivial.

Lemma 8.9.3. *Let $n \geq -2$ be a number and B be a type in $\mathfrak{C}$. Consider the equality semi-simplicial type $\mathcal{E} : \Delta_+^{\mathrm{op}} \to \mathfrak{C}$ of B. For an object $[\mathsf{k}]$ of Δ_+^{op}, we can consider the fibration $\mathcal{E}_{[\mathsf{k}]} \twoheadrightarrow M^{\mathcal{E}}_{[\mathsf{k}]}$. We know that, by definition, the fibre over $m : M^{\mathcal{E}}_{[\mathsf{k}]}$ is simply $\Sigma\,(x : B)\,.\,\tilde{\eta}_{[\mathsf{k}]}(x) = m$.*

If B is an n-type, then, for any object $[\mathsf{k}]$ of Δ_+^{op}, all these fibres are $(n - k)$-truncated (or contractible, if this difference is below -2).

Remark 8.9.4. The other direction of Lemma 8.9.3 should also hold, as $M^{\mathcal{E}}_{[\mathsf{k}]}$ should be equivalent to $\Sigma\,(b : B)\,.\,\Omega^k(B, b)$. We do neither prove nor require this direction here.

Proof of Lemma 8.9.3. The statement clearly holds for $[\mathsf{k}] \equiv [\mathsf{0}]$, as the matching object $M^{\mathcal{E}}_{[\mathsf{k}]}$ will in this case be the unit type. We assume that the statement holds for $[\mathsf{k}]$ and show it for $[\mathsf{k}+\mathsf{1}]$. First, consider the fibration

$$M^{\mathcal{E}}_{[\mathsf{k}+1]} \twoheadrightarrow \mathsf{lim}_{\overline{[\mathsf{k}+1]}_{-0}}(\mathcal{E} \circ U), \tag{8.80}$$

where the last expression is the 0-th $[\mathsf{k}]$-horn as in Main Lemma 8.5.1. This fibration can be completed to the following pullback square by Lemma 8.3.1:

$$\begin{array}{ccc} M^{\mathcal{E}}_{[\mathsf{k}+1]} & \twoheadrightarrow & \mathcal{E}_{[\mathsf{k}]} \\ \downarrow & & \downarrow \\ \mathsf{lim}_{\overline{[\mathsf{k}+1]}_{-0}}(\mathcal{E} \circ U) & \twoheadrightarrow & M^{\mathcal{E}}_{[\mathsf{k}]} \end{array}$$

By the induction hypothesis, the right vertical fibration is a $(n-k)$-truncated type. The left vertical fibration is therefore $(n-k)$-truncated as well (fibres on the left side are homotopy equivalent to fibres on on right side).

Consider the composition of fibrations

$$\mathcal{E}_{[\mathsf{k}+1]} \twoheadrightarrow M^{\mathcal{E}}_{[\mathsf{k}+1]} \twoheadrightarrow \mathsf{lim}_{\overline{[\mathsf{k}+1]}_{-0}}(\mathcal{E} \circ U). \tag{8.81}$$

Intuitively, the horn is a "tetrahedron with missing filler and one missing face", the matching object is the same plus one component which represents this face, and $\mathcal{E}_{[\mathsf{k}]}$ has, in addition to the face, also a filler of the whole boundary. The filler is really the statement that the "new" face equals the canonical one, and we can now make this intuition precise by applying Lemma 8.9.2. Let us check the conditions:

- Certainly, we can write the sequence in the form

  $$\Sigma\,(x : X)\,.\,\Sigma\,(x : Y(x))\,.\,Z(x, y) \twoheadrightarrow \Sigma\,(x : X)\,.\,Y(x) \twoheadrightarrow X \tag{8.82}$$

  (this is given by Lemma 8.3.2).

- The composition is a homotopy equivalence by Main Lemma 8.5.1.

Thus, we can assume that $Z(x, y)$ is equivalent to $y =_{Y(x)} y_x$ for some y_x, and thereby of a truncation level that is by one lower than $Y(x)$. But the latter is $(n-k)$ as we have seen above.[2] □

[2] On low levels, we can consider the situation in terms of the presentation in Figure 8.2. Here, y_x will be the "missing face" that one gets by gluing together the other faces.

As a corollary, we get the case for $[\mathsf{k}] \equiv [\mathsf{n}+2]$:

Corollary 8.9.5. *Let B be an n-type. Then, the fibration*

$$\mathcal{E}_{[\mathsf{n}+2]} \twoheadrightarrow M^{\mathcal{E}}_{[\mathsf{n}+2]} \tag{8.83}$$

is a homotopy equivalence. □

We are now in the position to formulate our result for n-types with finite n. Recall from Notation 8.6.3 that we write $A \xrightarrow{[\mathsf{n}]} B$ for $\mathcal{N}_n$.

Theorem 8.9.6 (Finite general universal property of the propositional truncation)**.** *Let n be a fixed number, $-2 \leq n < \infty$. In Martin-Löf type theory with propositional truncations and function extensionality we can, for any type A and any n-type B, derive a canonical function*

$$\big(\|A\| \to B\big) \to \big(A \xrightarrow{[\mathsf{n}+1]} B\big) \tag{8.84}$$

that is a homotopy equivalence.

Proof. Looking at Corollary 8.9.5 and at the definition of $\mathcal{N}$, as given in Section 8.6, we see immediately that each $\mathcal{N}_{[\mathsf{k}+1]} \twoheadrightarrow \mathcal{N}_{[\mathsf{k}]}$ with $k \geq n+1$ is a homotopy equivalence. Thus, using Lemma 8.2.5, the Reedy limit $\mathsf{lim}_{(\!|\Delta_+^{\mathrm{op}}|\!)}\mathcal{N}$ is equivalent to $\mathcal{N}_{[\mathsf{n}+1]}$, and these are $A \xrightarrow{\omega} B$ and $A \xrightarrow{[\mathsf{n}+1]} B$ by definition. Similarly, the limit $\mathsf{lim}_{(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)}\widehat{\mathcal{N}}$ (which we used in the proof of Lemma 8.8.3) is homotopy equivalent to the limit over $(\!|\mathfrak{d}_+^{\mathrm{op}}|\!)$ restricted to $\left\{ c^i_{[\mathsf{k}]} \;\middle|\; k \leq n+1 \right\}$. It is easy to see that the whole proof can be carried out using only finite parts of the infinite diagrams. But then, of course, all we need are finitely many nested Σ-types instead of Reedy ω^{op}-limits, and these automatically exist. Further, the only point where we crucially used the judgmental η-rule for Σ is the proof of Theorem 8.8.5. In the finite case, however, this is not necessary, as Lemma 8.1.1 is sufficient (similarly, the judgmental η-rule for Π-types is not necessary). Therefore, the whole proof can be carried out in the standard version of MLTT with propositional truncations. □

8.10 Elimination Principles for Higher Truncations

So far, we have seen that the universal property of the propositional truncation ($(A \to B) \simeq (\|A\| \to B)$ for a propositional B) allows us to characterise maps out of $\|A\|$ into general types, not only propositions. It suggests itself to ask whether statements analogous to Theorems 8.8.5 and 8.9.6 can be derived for higher truncations. Recall that we have $(\|A\|_n \to B) \simeq (A \to B)$ if B is an n-type, so what if B is not an n-type?

Together with Paolo Capriotti, I have worked on a solution to this question with higher inductive types. I believe that it is possible to prove a theorem similar to Theorem 8.9.6 which characterises $\|A\|_n \to B$ if B is m-truncated for some finite number m. However, even though we have ideas how we could in principle attack the general case, we can only solve the special case that m is $n+1$ in this thesis. We will see that even this case it quite involved with our approach, but we will also see that already this case has an application in the proof of Theorem 9.4.7.

After discussing the case $n \equiv 0$, $m \equiv 1$ with Andrea Vezzosi, he found an alternative argument which does not rely on higher inductive types. It can be generalised to the case $m \equiv n+1$ which I present here. While is is possible that Vezzosi's strategy could be used to find a solution for the general $n < m$ case (note that $n \geq m$ makes the problem trivial), this has to be expected to be significantly harder, for a reason that we will explain in Section 8.11.

We call the proof using higher inductive types the "HIT proof", and the one that builds upon Vezzosi's argument the "elementary proof".

If we want to get rid of all truncation conditions on B, i.e. derive a general result analogous to Theorem 8.8.5, I believe that the elementary approach is more promising. Of course, we would almost certainly still depend on the theory having Reedy ω^{op}-limits. It seems that the strategy with higher inductive types however would further require some form of "higher inductive types with infinitely many constructors". On the other hand, in the presence of Reedy ω^{op}-limits, this potential component of type theory might not even be absurd. For now, we need to leave all of this open for future work.

The current section of the thesis is largely based on many discussions with Paolo Capriotti, and Section 8.10.2 builds upon Vezzosi's argument. The main result here, together with its "elementary" proof, has been formalised by Capriotti, using a different library than the one that the electronic appendix of this thesis is based on. We do *not* mark the formalised statements with Ⓐ, as they can not be found in the electronic appendix. However, Theorem 8.10.2 with the proof in Section 8.10.2 is available in Capriotti's GitHub repository [Cap15].

8.10.1 Some Preparation and the Statement

To start, we need to clarify some simple constructions. If we have a type A and a pointed type (B, b), together with a function $f : A \to B$, we say that "f is null" if it is constantly b, that is,

$$\mathsf{isNull}(f) :\equiv \Pi_{x:A} b = f(x). \tag{8.85}$$

Recall that, given any two types A and B together with any function $f : A \to B$ and a point $a : A$, we have a function

$$\mathsf{ap}_{f,a} : \Omega(A, a) \to \Omega(B, f(a)). \tag{8.86}$$

In the same way, we have, for a number $n \geq 0$ in addition to A, B, f as before, the n-fold iterated ap-function

$$\mathsf{ap}^n_{f,a} : \Omega^n(A, a) \to \Omega^n(B, f(a)). \tag{8.87}$$

Here, we use the convention that $\mathsf{ap}^0_{f,a}$ is simply f.

Remark 8.10.1. Ω is really an endofunctor in some appropriate sense. Of course, $\mathsf{ap}_{f,a}$ is its action on the morphism f and could thus rightfully be called $\Omega(f, a)$.

Our result on higher truncation elimination can now be stated as follows:

Theorem 8.10.2. *Let $n \geq -1$ be a number, A a type, and B an $(n+1)$-type. Assume that $f : A \to B$ is a function. Then, f can be factored through the n-truncation, that is*

$$\Sigma\,(f' : \|A\|_n \to B)\,.\, f' \circ |-| = f, \tag{8.88}$$

if and only if $\mathsf{ap}^{n+1}_{f,a}$ is null for every a,

$$\Pi_{a:A} \mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a}), \tag{8.89}$$

and both of the types (8.88) *and* (8.89) *are propositional.*

An immediate corollary tells us how we can eliminate out of truncations:

Corollary 8.10.3. *Assume we have n, A and B as in Theorem 8.10.2. If we want to construct a function $\|A\|_n \to B$, it suffices to find a function $f : A \to B$ which satisfies $\Pi_{a:A} \mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$.*

Remark 8.10.4. Note that the function $\mathsf{ap}^{n+1}_{f,a}$ from the statements above is null if and only if it is weakly constant. Further, both types $\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ and $\mathsf{const}(\mathsf{ap}^{n+1}_{f,a})$ are propositional, as B is $(n+1)$-truncated by assumption. We could thus replace $\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ everywhere by $\mathsf{const}(\mathsf{ap}^{n+1}_{f,a})$; the reason why we prefer to talk about "being null" is explained in Section 8.11 below.

Before approaching a proof of Theorem 8.10.2, let us have a look at two special cases, namely the cases $n \equiv -1$ and $n \equiv 0$. We already know the first case, and it will serve as the base case for the two general proofs presented later. The second case is not strictly necessary, but serves to exemplify the techniques used in the "HIT proof" (Section 8.10.3).

The case $n \equiv -1$: The simplified statement of Theorem 8.10.2 reads in this case as follows: Assume we are given a type A and a 0-type B (often called a *set*). A function $f : A \to B$ factors (propositionally) through the propositional truncation,

$$\Sigma\,(f' : \|A\|_{-1} \to B)\,.\, f' \circ |-| = f, \tag{8.90}$$

if and only if it is *weakly constant*,

$$\Pi_{x,y:A} f(x) = f(y). \tag{8.91}$$

Of course, this is only a reformulation of Proposition 8.1.2. It is a pleasant surprise that "$\mathsf{ap}^0_{f,a}$ is null for all a", simply by unfolding our definitions, simplifies to "f is weakly constant". In the simplified formulation, we have omitted the part that the two logically equivalent types are propositional. This is easy to see here, and will in the general case be part of the proof.

The case n ≡ 0. Here, our result (Theorem 8.10.2) implies that, for any type A and 1-type B, a function $f : A \to B$ factors through $\|A\|_0$ if and only if, for all $a : A$ and $p : a = a$, we have that $\mathsf{ap}_{f,a}$ equals $\mathsf{refl}_{f(a)}$. As Shulman has remarked in an online discussion (in the comment section of a blog post [Cap14]), this follows from the *Rezk completion* [AKS15]: Let $\tilde{A}$ be the precategory with the type A of objects and $\mathsf{hom}(a_1, a_2) :\equiv \|a_1 =_A a_2\|_{-1}$, and let $\tilde{B}$ be the category with B as objects and $\mathsf{hom}(b_1, b_2) :\equiv (b_1 =_B b_2)$. Then, f with the condition $\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}_{f,a})$ gives (already using the case $n \equiv -1$) rise to a functor $\tilde{A} \to \tilde{B}$. Such a functor generates a functor between the Rezk completion of $\tilde{A}$ and the category $\tilde{B}$, and the former happens to be $\|A\|_0$.

In the remainder of the current section, we give a simple technical construction which essentially serves as a reformulation of Theorem 8.10.2. It is necessary for both the elementary and the HIT proof.

For types A and B, assume we are given a function $g : \|A\|_n \to B$. We can consider the composition

$$A \xrightarrow{|-|} \|A\|_n \xrightarrow{g} B. \tag{8.92}$$

For any $a : A$ we have, by functoriality of Ω^{n+1}, that the composition

$$\Omega^{n+1}(A, a) \xrightarrow{\mathsf{ap}^{n+1}_{|-|,a}} \Omega^{n+1}(\|A\|_n, |a|) \xrightarrow{\mathsf{ap}^{n+1}_{g,|a|}} \Omega^{n+1}(B, g(|a|)) \tag{8.93}$$

is equal to $\mathsf{ap}^{n+1}_{g\circ|-|,a}$. But $\Omega^{n+1}(\|A\|_n, |a|)$ is contractible ([Uni13, Theorem7.2.9]), and $\mathsf{ap}^{n+1}_{g,|a|}$ clearly maps its unique element to the basepoint of $\Omega^{n+1}(B, g(|a|))$. Therefore, $\mathsf{ap}^{n+1}_{g\circ|-|,a}$ is null. From this construction, we get a canonical function

$$\mathfrak{c}_n : (\|A\|_n \to B) \to \Sigma\,(f : A \to B)\,.\,\left(\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})\right). \tag{8.94}$$

We then claim the following:

Lemma 8.10.5 (Total space formulation of Theorem 8.10.2)**.** *For any $n \geq -1$, any type A and any $(n+1)$-type B, the types*

$$\|A\|_n \to B \tag{8.95}$$

and

$$\Sigma\,(f : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a}) \tag{8.96}$$

are equivalent, and the equivalence is given by the canonical function $\mathfrak{c}_n$.

It is easy to see that Lemma 8.10.5 does indeed imply, and is nearly immediately equivalent to, Theorem 8.10.2. Consider the triangle shown in Figure 8.7, where the top horizontal map is the canonical map $\mathfrak{c}_n$, the left one is composition with $|-|$, and the right one is simply the projection. The triangle clearly commutes (judgmentally) by construction. Let us fix some function $f : A \to B$. The fibre (or *inverse image*) over f is, in the case of $\_ \circ |-|$, exactly (8.88), i.e. the statement that f can be lifted. In the second case, the fibre is (8.89). Therefore, $\mathfrak{c}_n$ induces an equivalence of the two fibres, which implies that $\mathfrak{c}_n$ itself is an equivalence (see [Uni13, Theorem4.7.7]).

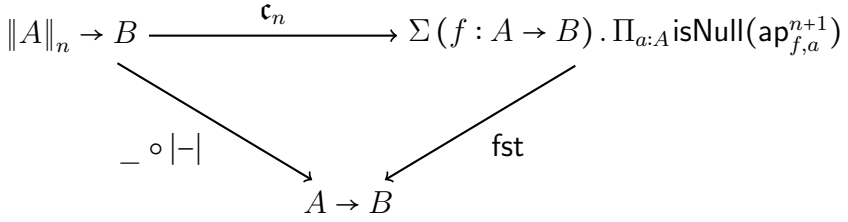


Figure 8.7: The canonical map $\mathfrak{c}_n$ as map between fibres

8.10.2 The "Elementary" Proof

In this section, we give our first proof of Lemma 8.10.5 (and thereby of Theorem 8.10.2). This does not need higher inductive types apart from truncations that already appear in the statement. The idea is to not prove the result for *any* type A first, but only for an n-connected one. Afterwards, we generalise this to arbitrary types, by splitting the type into its *connected components* and gluing together the constructions for the components. Note that the term *connected component* refers to the topological interpretation and has nothing to do with Σ-*components*, which are *syntactical* components of a type expression. The latter could rightfully be called *factors* of a Σ-type, but the terminology that we use is the one that has become accepted in type theory.

Lemma 8.10.6. *If $n \geq -1$ be a number, A an n-connected type, and B be an $(n+1)$-type, the canonical map $\mathfrak{c}_n$ is an equivalence.*

Proof. We do induction on n. As already discussed above, the case that n is -1 follows easily from Proposition 8.1.2.

Let now $n \geq 0$ be any given number. Note that, due to the assumption that $\|A\|_n$ is contractible, we have a unique element $x_0 : \|A\|_n$, the type $\|A\|_n \to B$ is actually equivalent to B, and any function $g : \|A\|_n \to B$ is uniquely specified by its value $g(x_0)$.

The claim of the lemma is propositional. Applying the eliminator of $\|A\|_n$, we may not only assume that we are given $x_0 : \|A\|_n$, but we can also assume a point

$a : A$. A potential inverse of $\mathfrak{c}_n$ is then given by

$$\mathfrak{d}_n : \left(\Sigma\,(f : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})\right) \to (\|A\|_n \to B) \tag{8.97}$$

$$\mathfrak{d}_n(f,p) :\equiv \lambda\_.f(a). \tag{8.98}$$

To show that $\mathfrak{c}_n$ and $\mathfrak{d}_n$ are inverses, we check that both compositions are the identities. One direction is easy: for any $g : \|A\|_n \to B$, we have

$$\mathfrak{d}_n(\mathfrak{c}_n(g))(x_0) \equiv g(|a|), \tag{8.99}$$

and the latter is equal to $g(x_0)$.

For the other direction, assume we have $f : A \to B$ together with a proof q. We need to show $(f,q) = \mathfrak{c}_n(\mathfrak{d}_n(f,q))$. Fortunately, the equality of the two second components is automatic thanks to the fact that $\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ is propositional, and we only need to prove the equality of f and $\mathsf{fst}(\mathfrak{c}_n(\mathfrak{d}_n(f,q)))$. We observe that the latter expression computes to $\lambda\_.f(a)$. Thus, our goal is to show that, for any $a' : A$, we have $f(a) = f(a')$.

We use the induction hypothesis with $(a = a')$ for A, and $f(a) = f(a')$ for B. By the connectedness assumption on A, the type $|a| = |a'|$ is contractible. Therefore, the type $\|a = a'\|_{n-1}$ is contractible ([Uni13, Theorem7.3.12]) or, put differently, $(a = a')$ is $(n-1)$-connected. As B is an $(n+1)$-type, we know that $f(a) = f(a')$ is n-truncated. By the induction hypothesis, it is hence enough to construct an element of

$$\Sigma\,(k : a = a' \to f(a) = f(a'))\,.\,\Pi_{p:a=a'}\mathsf{isNull}(\mathsf{ap}^n_{k,p}). \tag{8.100}$$

For k, we choose ap_f. By induction on p, we may assume that p is refl_a. Thus, we need to show that $\mathsf{ap}^n_{\mathsf{ap}_{f,a},\mathsf{refl}_a}$ is null. This term is equal to $\mathsf{ap}^{n+1}_{f,a}$. The condition that this function null is exactly what is given by $q(a')$. □

To move from n-connected to arbitrary types A, we simply split a type into n-connected components. This is very intuitive for $n \equiv 0$, in which case we use that any type (or space) can be viewed as the disjoint sum of its connected components. To be precise, an element of a component is a point of A together with a proof that it is in the component. For $n \equiv 0$, this proof is propositional. For higher n, it is not. This makes the general case less intuitive and hard to picture. In fact, the proof determines in which component the element is, which makes it seem circular. Fortunately, it is easier to write down the type-theoretic argument than picturing the topological intuition, as we will see in the following lemma.

Lemma 8.10.7. *For any type A and number n, we define the family of n-connected components,*

$$\mathsf{conn}_n : \|A\|_n \to \mathcal{U} \tag{8.101}$$

$$\mathsf{conn}_n(x) :\equiv \Sigma\,(a : A)\,.\,x =_{\|A\|_n} |a|. \tag{8.102}$$

Then, for any $x : \|A\|_n$, the type $\mathsf{conn}_n(x)$ is n-connected. Further, "choosing a connected component and then a point in this component" corresponds to "choosing a point", that is,

$$\Sigma\,(x : \|A\|_n)\,.\,\mathsf{conn}_n(x) \;\simeq\; A. \tag{8.103}$$

Proof. This is easy and standard. For the first part, we claim that the equivalence

$$\left\|\Sigma\,(a : A)\,.\,x =_{\|A\|_n} |a|\right\|_n \;\simeq\; \Sigma\,(y : \|A\|_n)\,.\,x =_{\|A\|_n} y \tag{8.104}$$

holds, where the left-hand type is $\|\mathsf{conn}_n(x)\|_n$ by definition, and the right-hand type is a singleton. For both directions of (8.104), we apply the dependent eliminator of $\|-\|_n$. From left to right, we map $|(a,p)|$ to $(|a|,p)$. From right to left, we map $(|a|,p)$ to $(|a,p|)$. For an alternative proof, see [Uni13, Corollary 7.5.8].

To see that the equivalence (8.103) holds, it is enough to unfold the definition of conn_n, and use that in $\Sigma\,(x : \|A\|_n)\,.\,\Sigma\,(a : A)\,.\,x =_{\|A\|_n} |a|$, the first and the third component form a singleton. □

Finally, we can complete the first proof of our main result:

"Elementary" proof of Lemma 8.10.5. Assume we have n, A, and B as in the statement. The preceding two lemmata tell us that, for any $x : \|A\|_n$, the canonical map

$$\mathfrak{c}_n^x : B \to \left(\Sigma\,(f_x : \mathsf{conn}_n(x) \to B)\,.\,\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f_x,y})\right) \tag{8.105}$$

is an equivalence (note that we have omitted the contractible type $\|\mathsf{conn}_n(x)\|_n$ in the domain of $\mathfrak{c}_n^x$). A family of equivalences gives rise to an equivalence of families, so that we get that the map

$$\tilde{\mathfrak{c}}_n : (\|A\|_n \to B) \;\to\; \left(\Pi_{x:\|A\|_n}\Sigma\,(g_x : \mathsf{conn}_n(x) \to B)\,.\,\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{g,y})\right) \tag{8.106}$$

$$\tilde{\mathfrak{c}}_n(k) :\equiv \lambda x.\mathfrak{c}_n^x(k(x)) \tag{8.107}$$

is also an equivalence.

All we need at this point is an equivalence from the codomain of the function (8.107) to the type stated in the theorem, i.e. $\Sigma\,(f : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$, and the composition of (8.107) and this equivalence has to be the canonical map $\mathfrak{c}_n$. We calculate:

$$\Pi_{x:\|A\|_n}\Sigma\,(g_x : \mathsf{conn}_n(x) \to B)\,.\,\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{g_x,y}) \tag{8.108}$$

(by the distributivity law)

$$\simeq\quad \Sigma\left(g : \Pi_{x:\|A\|_n}(\mathsf{conn}_n(x) \to B)\right)\,.\,\Pi_{x:\|A\|_n}\Pi_{y:\mathsf{conn}_n(x)}\mathsf{isNull}(\mathsf{ap}^{n+1}_{g(x),y}) \tag{8.109}$$

(by currying and using the canonical equivalence (8.103))

$$\simeq\quad \Sigma\,(h : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{\lambda y:\mathsf{conn}_n(|a|).h(\mathsf{fst}y),(a,\mathsf{refl}_{|a|})}) \tag{8.110}$$

Fortunately, the (pointed) types $\Omega^{n+1}(\mathsf{conn}_n(|a|), (a, \mathsf{refl}_{|a|}))$ and $\Omega^{n+1}(A, a)$ are equivalent, with the equivalence being $\mathsf{ap}^{n+1}_{\mathsf{fst}}$; this is an easy technical statement that follows from [KS15, Lemma 5.1]. If we compose $\mathsf{ap}^{n+1}_{\lambda y:\mathsf{conn}_n(|a|).h(\mathsf{fst}y),(a,\mathsf{refl}_{|a|})}$ with the inverse of this equivalence, functoriality of ap^{n+1} allows us to simplify the expression.

$$\simeq \quad \Sigma\,(h : A \to B)\,.\,\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{h,a}) \tag{8.111}$$

We need to check that the composition of $\tilde{\mathfrak{c}}_n$ with this equivalence is indeed the canonical function $\mathfrak{c}_n$. This is immediate because we only need to check that the first component (the map $A \to B$) turns out to be the correct function, as the second component is propositional. □

8.10.3 The "HIT Proof"

Our second proof is fairly technical. We construct a higher inductive type which is, in some appropriate sense, the *initial* type through which functions $f : A \to B$ with the property $\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ for all $a : A$ factor. This higher inductive type has exactly the right elimination principle (the one which we want to show for $\|A\|_n$), and we prove that it is indeed equivalent to $\|A\|_n$. For the following definition and for the rest of the subsection, we fix a type A and a number $n \geq -1$.

Definition 8.10.8. Define the higher inductive type H, which depends on A and n, as given by the constructors

$$\eta : A \to H \tag{8.112}$$
$$\epsilon : \Pi_{a,b:A}\,(\|a = b\|_{n-1} \to \eta(a) = \eta(b)) \tag{8.113}$$
$$\delta : \Pi_{a:A}\left(\mathsf{refl}_{\eta(a)} =_{\eta(a)=\eta(a)} \epsilon(a, a, |\mathsf{refl}_a|)\right) \tag{8.114}$$
$$t : \mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(H). \tag{8.115}$$

The definition of H is much more intuitive than it may look at first sight. η obviously says that we may always go from A to H, similarly as one always has $|-| : A \to \|A\|_n$. However, at the moment, we only know that H is an $(n+1)$-type from the constructor t. Note that, while t does not directly have the shape of a HIT constructor, $\mathsf{is}\text{-}(n+1)\text{-}\mathsf{type}(H)$ can for any (fixed) n be unfolded and brought into the required form.

If we have $(a = b)$, we of course always get a proof of $\eta(a) = \eta(b)$ using ap_η. The constructor ϵ says that $\|a = b\|_{n-1}$ is sufficient, while δ ensures that ϵ is really an extension of ap_η along $|-|_{n-1}$. This is because we could have used the expanded form

$$\delta' : \Pi_{a,b:A}\Pi_{p:a=b}\left(\mathsf{ap}_\eta(p) =_{\eta(a)=\eta(b)} \epsilon(a, b, |p|)\right), \tag{8.116}$$

instead of the constructor δ. By path induction on p, the type (8.116) it is easily seen to be equivalent to the original type (8.114). While (8.116) might look more regular next to (8.113), we choose (8.114) just for simplicity.

The recursion principle for H is straightforward to write down. Given some $(n+1)$-type B, we need a function $f : A \to B$, together with a function

$$k : \Pi_{a,b:A}(\|a = b\|_{n-1}) \to f(a) = f(b) \tag{8.117}$$

and a proof

$$h : \Pi_{a:A} \mathsf{refl}_{f(a)} =_{f(a)=f(a)} k(a, a, |\mathsf{refl}_{f(a)}|), \tag{8.118}$$

we get a function $H \to B$ with the expected properties. It is more involved, nevertheless not inherently difficult, to state the induction principle following the standard (intuitive) approach as used in [Uni13, Chapter 6]. Given an $(n+1)$-truncated family $P : H \to \mathcal{U}^{n+1}$, in order to prove $\Pi_{x:H} P(x)$, we need

$$\overline{\eta} : \Pi_{a:A} P(\eta(a)) \tag{8.119}$$

$$\overline{\epsilon} : \Pi_{a,b:A} \Pi_{q:\|a=b\|_{n-1}} \mathsf{transport}^P(\epsilon(a,b,q), \overline{\eta}(a)) =_{P(\eta(b))} \overline{\eta}(b) \tag{8.120}$$

$$\overline{\delta} : \Pi_{a:A} \left(\mathsf{transport}^{\lambda r.\mathsf{transport}^P(r,\overline{\eta}(a))=\overline{\eta}(a)} \left(\delta(a), \mathsf{refl}_{\overline{\eta}(a)}\right) = \overline{\epsilon}(a, a, |\mathsf{refl}_a|) \right). \tag{8.121}$$

The above type expressions look rather involved. Fortunately, we do not need to deal too much with them at all because we are only interested in the case that P is n-truncated (instead of, more generally, $(n+1)$-truncated), which enables us to use the following observation:

Lemma 8.10.9 (Restricted dep. universal property of H)**.** *Given A and $n \geq -1$ as above and a family of n-types, $P : H \to \mathcal{U}^n$, the canonical map*

$$\Pi_{x:H} P(x) \xrightarrow{-\circ\eta} \Pi_{a:A} P(\eta(a)) \tag{8.122}$$

is an equivalence.

Proof. As P is a family of n-types, the type of $\overline{\epsilon}$ is $(n-1)$-truncated. By the standard universal property of the $(n-1)$-truncation, we may thus assume that the q in the type (8.123) is of the form $|p|$ with $p : a = b$, and then do path induction on p. This shows that the type of $\overline{\epsilon}$ is equivalent to

$$\overline{\epsilon}'' : \Pi_{a:A} \mathsf{transport}^P(\epsilon(a, a, |\mathsf{refl}_a|), \overline{\eta}(a)) =_{P(\eta(a))} \overline{\eta}(a). \tag{8.123}$$

Under this equivalence, the type of $\overline{\delta}$ becomes

$$\overline{\delta}'' : \Pi_{a:A} \left(\mathsf{transport}^{\lambda r.\mathsf{transport}^P(r,\overline{\eta}(a))=\overline{\eta}(a)} \left(\delta(a), \mathsf{refl}_{\overline{\eta}(a)}\right) = \overline{\epsilon}''(a) \right). \tag{8.124}$$

We see that the dependent pair of (8.123) and (8.124) forms a family of singletons. Therefore, there is always a canonical and unique choice for $\overline{\epsilon}$ and $\overline{\delta}$. The induction principle can therefore be simplified to only (8.119); that is, for any function $\Pi_{a:A} P(\eta(a))$, the induction principle gives us $\Pi_{x:H} P(x)$. Let us write $\mathsf{rind} : \Pi_{a:A} P(\eta(a)) \to \Pi_{x:H} P(x)$ for this *restricted induction principle*.

The claim of the lemma should now really directly follow from a more general principle: it should be possible to derive statements saying that, if we replace types

of constructors of a HIT by equivalent types, then the new HIT is equivalent to the original one. However, such theorems have not been shown so far. Nevertheless, the low level approach of applying the induction principle is not tedious either. The map $\_ \circ \eta$ that we are supposed to prove invertible has the constructed restricted induction principle rind as an inverse:

- For any $f : \Pi_{a:A} P(\eta(a))$ and $a : A$, the expression $(\mathsf{rind}(f) \circ \eta)(a)$ can be reduced to $f(a)$.
- For any $g : \Pi_{x:H} P(x)$, assume $x : H$. We need to show $(\mathsf{rind}(g \circ \eta))(x) = g(x)$. Using the restricted induction principle, we may assume $x \equiv \eta(a)$, and the left side can be reduced to the right side of the equation. □

This allows us to conclude the following crucial property of H:

Lemma 8.10.10. *The type H is n-truncated.*

Proof. It suffices to show that $\Omega^{n+1}(H, x)$ is contractible for all $x : H$ ([Uni13, Lemma 7.2.9], or Theorem 3.2.1). The restricted induction principle of H tells us that, in order to show

$$P(x) :\equiv \mathsf{isContr}\left(\Omega^{n+1}(H, x)\right) \tag{8.125}$$

for all x, we only need to prove $P(\eta(a_0))$ for any $a_0 : A$. Let us define a type family

$$Q : H \to \mathcal{U}^{n-1} \tag{8.126}$$

using the restricted induction principle,

$$Q(\eta(a)) :\equiv \|a_0 = a\|_{n-1}. \tag{8.127}$$

This family is trivially inhabited at a_0. We want to show that Q implies local equality in the sense of

$$\Pi_{x:H} \left(Q(x) \to \eta(a_0) = x\right), \tag{8.128}$$

and as this type family is n-truncated, we apply the restricted induction principle again and the goal becomes

$$\Pi_{a:A} \left(Q(\eta(a)) \to \eta(a_0) = \eta(a)\right). \tag{8.129}$$

By definition of Q, this is exactly given by the constructor ϵ, applied on a_0 and a.

This allows us to conclude, by Theorem 3.2.1, $(3) \Rightarrow (1)$, that H is n-truncated, as claimed. □

It is straightforward and standard that an n-truncated type which satisfies the dependent eliminating principle of $\|A\|_n$ is necessarily equivalent to $\|A\|_n$, and we record:

Corollary 8.10.11. *The types H and $\|A\|_n$ are equivalent.*

At the same time, we have the following:

Lemma 8.10.12 (Universal property of H)**.** *For any $(n+1)$-type B, the type of functions $H \to B$ is equivalent to*

$$\begin{aligned}&\Sigma\,(f : A \to B)\,.\\ &\Sigma\,(e : \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b))\,.\\ &\quad (d : \Pi_{a:A}\mathsf{refl}_{f(a)} = e(a, a, |\mathsf{refl}_a|).\end{aligned} \tag{8.130}$$

Proof sketch. The proof of deriving this form of universal property from the induction principle is standard. The map from $H \to B$ into the stated type is more or less composition with the constructors; for any $k : H \to B$, we get

$$f :\equiv k \circ \eta :\ A \to B \tag{8.131}$$

$$e :\equiv \mathsf{ap}_k \circ \epsilon :\ \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b) \tag{8.132}$$

$$d :\equiv \lambda a.\mathsf{ap}_{\mathsf{ap}_k}(\delta(a)) :\ \Pi_{a:A}\mathsf{refl}_{f(a)} = e(a, a, |\mathsf{refl}_a|). \tag{8.133}$$

The map in the other direction is exactly the recursion principle of H. That they are mutually inverse corresponds to the computation (β) rule respectively the uniqueness (η) rule of H. □

Finally, we can complete the second proof of our main result:

"HIT proof" of Lemma 8.10.5. We do induction on n. The base case ($n \equiv -1$) is, as before, just what we have discussed in Section 8.10.1. For higher n, we have the following chain of equivalences:

$$\|A\|_n \to B \tag{8.134}$$

(by Corollary 8.10.11)

$$\simeq \quad H \to B \tag{8.135}$$

(by Lemma 8.10.12)

$$\begin{aligned}\simeq \quad &\Sigma\,(f : A \to B)\,.\,\Sigma\,(e : \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b))\,.\\ &\qquad \left(\Pi_{a:A}\mathsf{refl}_{f(a)} = e(a, a, |\mathsf{refl}_a|)\right)\end{aligned} \tag{8.136}$$

(by "inverse path induction")

$$\begin{aligned}\simeq \quad &\Sigma\,(f : A \to B)\,.\,\Sigma\,(e : \Pi_{a,b:A}\|a = b\|_{n-1} \to f(a) = f(b))\,.\\ &\qquad \left(\Pi_{a,b:A}\Pi_{p:a=b}\mathsf{ap}_f p = e(a, b, |p|)\right)\end{aligned} \tag{8.137}$$

(by the distributivity law)

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Pi_{a,b:A}\big(\Sigma\,(e' : \|a = b\|_{n-1} \to f(a) = f(b))\,. \\ \Pi_{p:a=b}\mathsf{ap}_f p = e'(|p|)\big) \tag{8.138}$$

Now we exchange e' by $(e_1, e_2) :\equiv \mathfrak{c}_{n-1}(e')$ using the induction hypothesis, and thus we need to apply $\mathfrak{c}_{n-1}^{-1}$ to that term in the last component. Fortunately, it follows from the definition of $\mathfrak{c}_{n-1}$ that $\_ \circ \mathfrak{c}_{n-1} \equiv \mathsf{fst} \circ |-|$, hence we can replace $e'(|p|)$ with simply $e_1(p)$:

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Pi_{a,b:A}\big(\Sigma\,(e_1 : a = b \to f(a) = f(b))\,.\,\Sigma\,\big(e_2 : \Pi_{p:a=b}\mathsf{isNull}(\mathsf{ap}^n_{e_1,p})\big)\,. \\ \big(\Pi_{p:a=b}\mathsf{ap}_f p = e_1(p)\big)\big) \tag{8.139}$$

The term e_1 and the very last (unnamed) component form a singleton and can be removed:

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Big(\Pi_{a,b:A}\Pi_{p:a=b}\mathsf{isNull}(\mathsf{ap}^n_{\mathsf{ap}_f,p})\Big) \tag{8.140}$$

(by path induction)

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\Big(\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^n_{\mathsf{ap}_f,\mathsf{refl}_{f(a)}})\Big) \tag{8.141}$$

(as $\mathsf{ap}^n_{\mathsf{ap}_f,\mathsf{refl}_a}$ is equal to $\mathsf{ap}^{n+1}_{f,a}$)

$$\simeq \quad \Sigma\,(f : A \to B)\,.\,\big(\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,\mathsf{refl}_a})\big)\,. \tag{8.142}$$

Finally, we need to check that the constructed equivalence is indeed the canonical function $\mathfrak{c}_n$. Fortunately, the second (and more involved) part $\Pi_{a:A}\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,\mathsf{refl}_a})$ is propositional. It is therefore enough to check that any map $g : \|A\|_n \to B$ gets, by the constructed equivalence, mapped to a pair in (8.142) of which *the first component* is $g \circ |-|$. But the first component is constructed in the very first step, where Lemma 8.10.12 is applied, and, looking at the proof of Lemma 8.10.12, it is indeed simply composition with $|-|$. □

8.11 The Big Picture: Solved and Unsolved Cases

Both Theorem 8.9.6 (or the more general Theorem 8.8.5) and Theorem 8.10.2 generalise Proposition 5.2.4, although in two different directions. In the first case, we characterise functions $\|A\|_n \to B$, but only for the case $n \equiv -1$, while the second case does not restrict n, but requires B to be an $(n+1)$-type. Of course, the general question is: What is the universal property of $\|A\|_n$ with respect to m-types, i.e. how can we construct a map $\|A\|_n \to B$ for some m-type B? Put differently, given a function $f : A \to B$, how can we (by only imposing conditions on f, not on A or

B) ensure that f factors through $\|A\|_n$? Figure 8.8 illustrates the current progress on this question. As indicated, the question is trivial if m is not greater than n, as this case is covered by the "ordinary" universal property and elimination principle. Non-trivial cases are solved by Proposition 5.2.4 and Theorems 8.8.5, 8.9.6 and 8.10.2. Note that only the most difficult case (Theorem 8.8.5) requires Reedy ω^{op}-limits and is not expected to be internalisable in the currently considered theory. The *family* of statements given by Theorem 8.9.6 is also not expected to be internalisable.

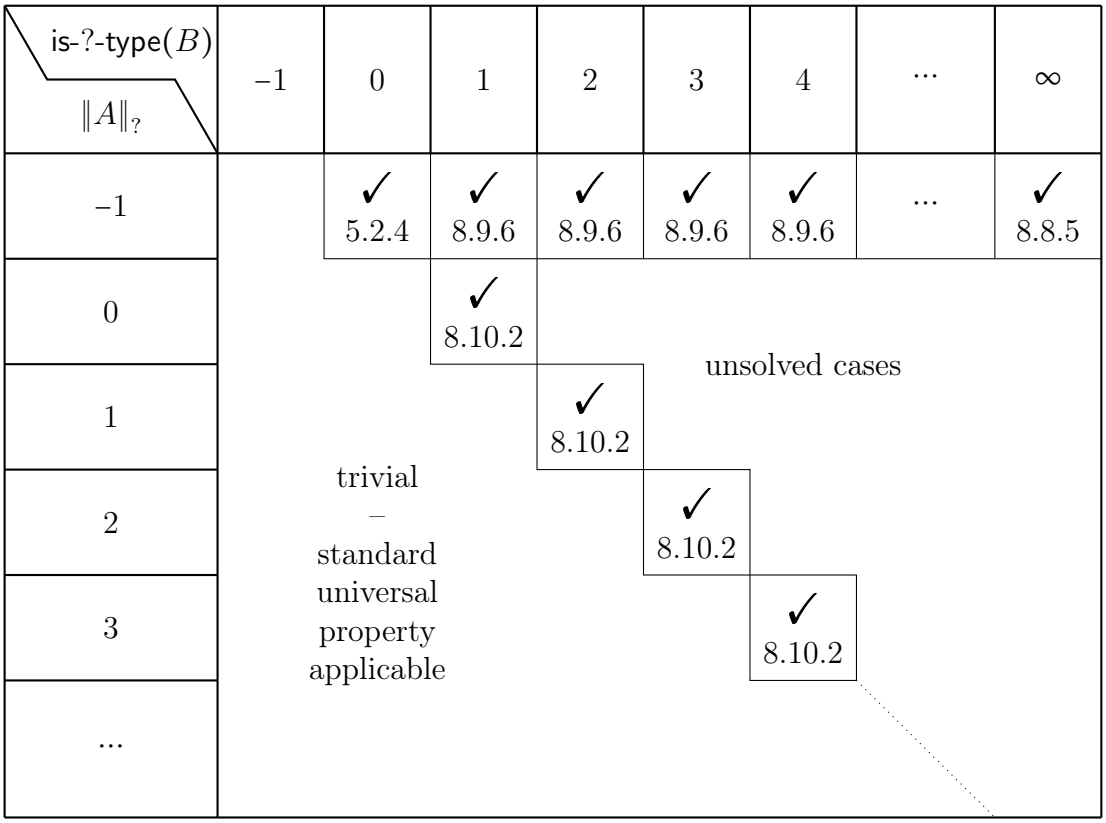


Figure 8.8: The universal property of $\|A\|_?$ with respect to ?-types: trivial, solved, and open cases

The (probably) simplest case that is left open is the case $n \equiv 0$, $m \equiv 2$, and we want to use this case to give some intuition on why the remaining cases are harder than the solved ones. So, consider a function $f : A \to B$, where B is 2-truncated. Which conditions do we have to impose on f to conclude that it factors through $\|A\|_0$? The one we gave in Theorem 8.10.2 (for general n) is equivalent to saying that f induces a trivial map on all $(n+1)$-st homotopy groups. One might therefore guess that, in the current case, it suffices to require f to induce trivial maps on all first and second homotopy groups. This would be nice as this condition is propositional, but, perhaps unsurprisingly, it does not suffice. One can also try

to replicate the coherence conditions that we have given for the case $n \equiv -1$, as discussed in Section 8.1. However, this does not address the main difficulty. In one aspect, the propositional truncation is a special case that is actually *harder* than the higher truncations, intuitively because loop spaces are always pointed.[3] In fact, in this "pointed" case, we can get all these coherences conditions (which make Theorem 8.8.5 hard) very easily, as we explain in and after the following proposition.

Proposition 8.11.1. *Let (A, a_0) be a pointed type and B be any type. Say that a function $f : A \to B$ is* coherently constant at the image of a_0 *if we have*

$$\Sigma\,(p : \Pi_{a:A} f(a) = f(a_0))\,.\,p(a_0) = \mathsf{refl}_{b_0}. \tag{8.143}$$

Then, the type of functions that are coherently constant at the image of a_0 is equivalent to B.

Proof. We can calculate

$$\Sigma\,(f : A \to B)\,.\,\Sigma\,(p : \Pi_{a:A} f(a) = f(a_0))\,.\,p(a_0) = \mathsf{refl}_{f(a_0)} \tag{8.144}$$

(by adding a singleton (b_0, q))

$$\begin{aligned} \simeq \quad & \Sigma\,(b_0 : B)\,.\,\Sigma\,(f : A \to B)\,.\,\Sigma\,(q : f(a_0) = b_0)\,. \\ & \qquad \Sigma\,(p : \Pi_{a:A} f(a) = f(a_0))\,.\,p(a_0) = \mathsf{refl}_{f(a_0)} \end{aligned} \tag{8.145}$$

(by substituting b_0 for $f(a_0)$ in the type of p)

$$\begin{aligned} \simeq \quad & \Sigma\,(b_0 : B)\,.\,\Sigma\,(f : A \to B)\,.\,\Sigma\,(q : f(a_0) = b_0)\,. \\ & \qquad \Sigma\,(p : \Pi_{a:A} f(a) = b_0)\,.\,p(a_0) = q \end{aligned} \tag{8.146}$$

(by removing the singleton consisting of q and the very last factor)

$$\simeq \quad \Sigma\,(b_0 : B)\,.\,\Sigma\,(f : A \to B)\,.\,\Pi_{a:A} f(a) = b_0 \tag{8.147}$$

(by using the distributivity law and removing a family of singletons)

$$\simeq \quad B \tag{8.148}$$

□

Of course, if we have $a_0 : A$, then B is also equivalent to $A \xrightarrow{\omega} B$ by Lemma 8.8.3. It is easy to check that, for any $f : A \to B$, the type (8.143) of proofs that f is coherently constant at the image of a_0 is equivalent to the type which states that f is ω-constant (the fibre over f in the fibration $(A \xrightarrow{\omega} B) \twoheadrightarrow (A \to B)$. But this

[3] This seems to correspond to the fact that the zeroth homotopy "group" is not a group, and does therefore not have a canonical element, which seems to occasionally make this special case harder in traditional topology as well.

means that, in the case of a *pointed* type (A, a_0), the infinite tower of conditions can be replaced by the very simple finite type (8.143). This result means that it is actually easy to state that $\mathsf{ap}^k_{f,a}$ is not only weakly constant, but ω-constant. In fact, this is our reason for preferring $\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$ over $\mathsf{const}(\mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a}))$, as discussed in Remark 8.10.4: Although in our case of interest both types are equivalent, isNull seems to be conceptually better. By only adding one single condition on a proof $p : \mathsf{isNull}(\mathsf{ap}^{n+1}_{f,a})$, namely the condition that $p(\mathsf{refl}^{n+1}_{a_0})$ equals $\mathsf{refl}^{n+1}_{f(a_0)}$, we can ensure that $\mathsf{ap}^{n+1}_{f,a}$ is ω-constant.

Thus, all the coherence conditions that were a major issue in the formulation and proof of our main result on propositional truncations, Theorem 8.8.5, can be managed easily for the higher truncations. What happens instead is that the higher groupoid structure of loop spaces induces a seemingly different sort of coherence problem. It certainly *is* necessary that, for any $a : A$ and $p : a = a$, there is a proof $c_{a,p} : \mathsf{ap}_{f,a}(p) = \mathsf{refl}_{f(a)}$. It seems plausible to also ask for a proof one level higher, ensuring that c_{a,refl_a} is reflexivity. However, this is not enough: By functoriality of $\mathsf{ap}_{a,f}$, we have that $\mathsf{ap}_{a,f}(p \cdot p)$ equals $\mathsf{ap}_{a,f}(p) \cdot \mathsf{ap}_{a,f}(p)$. Thus, $c_{a,p}$ allows us to construct a proof that $\mathsf{ap}_{a,f}(p \cdot p)$ equals $\mathsf{refl}_{f(a)}$. The family c can only be coherent if this proof is forced to be the same as $c_{a,p \cdot p}$.

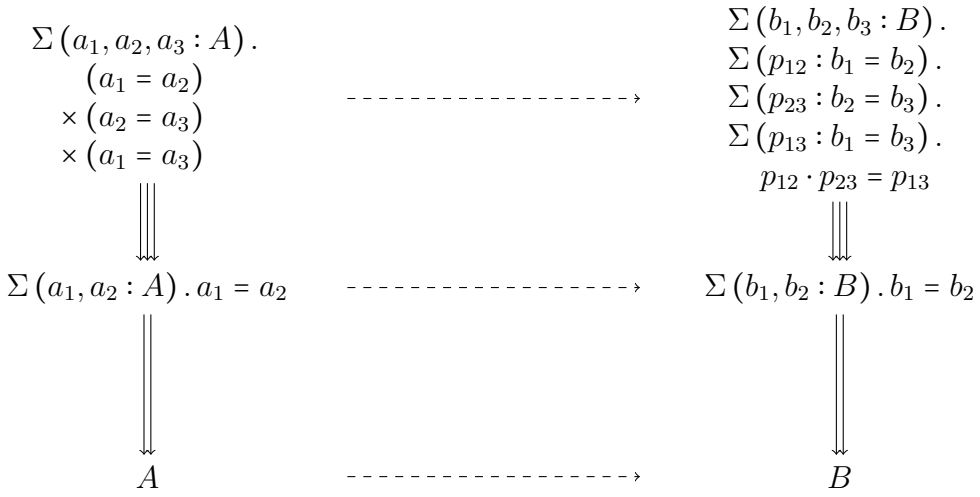


Figure 8.9: Natural transformations from the $[1]$-coskeleton of the equality semi-simplicial type on A to the equality semi-simplicial type on B are conjectured to correspond to functions $\|A\|_0 \to B$; only the components on the three lowest levels are drawn

As we have shown that natural transformations from the $[0]$-coskeleton of the constant diagram on A to the the equality semi-simplicial type on B correspond to maps from $\|A\|$ to B, it is natural to conjecture that, in the general case, we should consider natural transformations from the $[n+1]$-coskeleton of the constant diagram on A (or rather a fibrant replacement of it) to the equality semi-simplicial

type on B. In the case of the 0-truncation, this would amount to the situation illustrated in Figure 8.9. At the current time, we do not know whether this captures all the required coherences and whether the missing parts of Figure 8.8 can be filled in this way.

Chapter 9

Future Directions and Concluding Remarks

In this final part of the thesis, we describe approaches to formalisations of concepts that are commonly believed to be impossible in the current theory, and present some related results. First, we discuss the famous open problem of defining semi-simplicial types (see e.g. [Her15; Shu14]). We further show how to use that types already have the correct structure in order to define some form of weak ω-groupoids. We we them call *Yoneda-groupoids*; however, these do not seem to allow many interesting further constructions. More modestly, we turn to ordinary (1-) groupoids and discuss a question raised by Altenkirch, namely whether such a groupoid (a 1-type) can be represented by a set of objects and a family of sets of morphisms. Finally, additional notes on related work are given.

9.1 The Problem of Formalising Infinite Structures

Univalent foundations, so is the hope of many, have the potential to serve as a foundation for mathematics that make it possible to formalise (and thereby verify) huge parts of traditional mathematics. Many concepts from homotopy theory (originally constructed using set theory) can indeed be formalised in a very neat way. Probably even more important is the question what the current formulation of homotopy type theory allows us to formalise at all, not even necessarily in an elegant way. Results in that direction (especially negative ones) should certainly guide the discussion on what the "best" (in whichever sense, might it be philosophical or pragmatical) and "most convenient" theory to work in is.

At the moment, it seems that certain important infinite constructions can not be performed internally. Most importantly, it is believed that we can not construct a model of homotopy type theory in itself (without restricting the truncation levels of all types), something that would be desirable for many reasons.

First, such a construction might give answers to questions related to (homotopy) canonicity. The rough idea is that, if we use a constructive meta-theory, the "canonical" value of a given term can be obtained by checking what it evaluates to in the semantics. This approach has recently received a lot of attention, especially the model in cubical sets by Bezem, Coquand, and Huber [BCH14]. Their model does not use homotopy type theory as a meta-theory, and there is on-going on developing an appropriate syntax for "cubical type theory" by Altenkirch and Kaposi [AK14a],[AK14b], and by Brunerie and Licata [BL14]. On the other hand, Shulman [Shu14] has discussed the possibility to interpret homotopy type theory in itself, and it seems likely that this is not the case. Another reason why it would be desirable to do this, as Shulman argues, is of more philosophical nature: if we want a system to be a foundation for all of mathematics, it has to be able to serve as its own meta-theory. In terms of programming language theory, as Shulman mentions next to other motivations, it is natural to demand that "any general-purpose programming language must be able to implement its own compiler or interpreter".

Related (and maybe even to some extend equivalent) problems are the formalisation of weak ω-groupoids and semi-simplicial types, of which we discuss the latter below.

In this chapter, we assume the univalence axiom and higher inductive types, although the latter assumption can be dropped for most of what we do.

9.2 Semi-Simplicial Types

Defining semi-simplicial types internally in homotopy type theory is a challenge that, as far as I know, first came up at the special year program on univalent foundations in Princeton, 2013/14. From a meta-theoretical point of view, it is easy to describe them: they are Reedy fibrant type-valued presheaves over Δ_+, i.e. fibrant diagrams from Δ_+^{op} to, say, the first universe $\mathcal{U}_0$.[1] However, it seems to be hard to impossible to express this internally due to coherence issues. In (the currently considered version of) homotopy type theory, it is not possible to state a judgmental equality internally, which one would want to do to specify the functor laws. The only thing that would be left to do is to say that laws such as associativity hold "up to a higher paths". It would then be necessary to explain how these families of higher paths "fit together", in the same way as one requires such properties for weak n-categories (similar as, for example, *Mac Lane's pentagon* [Mac71] for monoidal categories). It seems very plausible that such an approach should in principle be possible for n-categories (with some fixed finite number n), with the main question being whether it can be done in a sufficiently clever way so that it is actually feasible. However, in the setting for an ω-category

[1] Recall that Δ_+ is the category of non-zero finite ordinals and strictly increasing functions (see the beginning of Section 8.5).

(or rather an ω-groupoid), each level of laws would analogously generate the necessity for new laws, and this leaves us with something which does not seem to be formalisable internally. On the other hand, the problem that arises seems to be similar to the reasons why it is hard (or impossible) to do what we discussed above, i.e. to model homotopy type theory in itself. We expect that, if we had the ability to formalise semi-simplicial types, many other important formalisation problems would become accessible. This is mostly speculation, as we do not know of any precise statement. However, there seems to be a connection as an interpreter for HoTT in HoTT would probably allow us to define semi-simplicial types; see Remark 9.2.2 below.

Of course, the discussion above does not only apply to semi-simplicial types. We have not yet made use of specific properties of Δ_+. If we could express what it means for a functor $\Delta_+ \to \mathcal{U}$ to be "fully coherent" in the way outlined, we could do this for any "type-valued presheaf". To give an example, we could formalise actual simplicial types, i.e. $\Delta^{\mathrm{op}} \to \mathcal{U}$. Semisimplicial types are a more modest aim. The reason why it is reasonable to expect that they are easier than the general case is that Δ_+ is an inverse category (in the sense of Section 8.2). We may thus try to define semi-simplicial types in the "Reedy way". Even more modestly, we restrict ourselves to *truncated* semi-simplicial types, which only have simplices at dimension less than a fixed given number. For example, a 0-truncated semi-simplicial type $X_{[0]}$ (consisting of "only points") is really only an ordinary type; that is, we can simply write

$$X_{[0]} : \mathcal{U}. \tag{9.1}$$

A 1-truncated semi-simplicial type has "points" and "directed lines" (0-simplices and 1-simplices). We want to use that we have just defined the type of "points". Given $X_{[0]}$, we have one "directed line" for every pair of "points":

$$X_{[1]} : X_{[0]} \to X_{[0]} \to \mathcal{U}. \tag{9.2}$$

Technically, the type of 1-truncated semi-simplicial types is then the Σ-type, consisting of both the components $X_{[0]}$ and $X_{[1]}$. For the case of a 2-truncated semi-simplicial type, what is new is that there might be "fillers for triangles",[2]

$$X_2 : (x_1, x_2, x_3 : X_{[0]}) \to X_{[1]}(x_1, x_2) \to X_{[1]}(x_2, x_3) \to X_{[2]}(x_1, x_3) \to \mathcal{U}. \tag{9.3}$$

Again, the type of 2-truncated semi-simplicial types is the (nested) Σ-type with three components, namely $X_{[0]}$, $X_{[1]}$, and $X_{[2]}$. As we have restricted ourselves to truncated semi-simplicial types, we are not really trying to define what corresponds to a functor $\Delta_+ \to \mathcal{U}$, but rather what corresponds to a functor $\Delta_{+,n}^{\mathrm{op}} \to \mathcal{U}$. Here, $\Delta_{+,n}$ is the full subcategory of Δ_+ with objects up to n. It thus should be clear how to continue from here, and it is easy (although tedious) to write down the next couple of definitions explicitly.

[2]Note that we once more implicitly uncurry and write $X_{[0]}(x_1, x_2)$ instead of $X_{[0]}\, x_1\, x_2$.

The actual challenge consists of writing down a function $\mathbb{N} \to \mathcal{U}$ which, for each n, takes (up to equivalence) the type of n-truncated semi-simplicial type as value. Maybe surprisingly, it seems to be impossible to do this, at least in the theory that we are considering (the version of homotopy type theory outlined in Chapter 2). At the same time, it seems to be hard to characterise exactly why apparently promising approaches do not work. Multiple people have tried hard to perform the construction without succeeding. I am not aware of the full history of the problem, and I apologise in advance for the very fragmentary information that follows in the next paragraph. I am sure that I fail to mention everyone who would deserve to be acknowledged in the context of this problem.

Originally, the challenge was raised as an open problem at the special year program in Princeton by LeFanu Lumsdaine, after it had already been discussed for a while between members of the Carnegie-Mellon University and the Institute for Advanced Study. Voevodsky had started a formalisation in Coq [Voe12], experiencing coherence problems. The problem was a major motivation for him to develop HTS (Homotopy Type System) [Voe13a], a type system with two identity types, in which the definition of semi-simplicial types is actually possible. An implementation of such a system with two identity types, led by Andrej Bauer, is called *Andromeda*, and can be found in Bauer's github repository.[3] The challenge of defining semi-simplicial types was further examined extensively by Herbelin [Her15] and several other participants of the special year program. I myself have spent a considerable amount of time on the problem as well, mostly together with Nuo Li, but also with Thorsten Altenkirch and (later) with Paolo Capriotti, Ambrus Kaposi, and Christian Sattler. Later, semi-simplicial types have been discussed in the blog post by Shulman that we mentioned above [Shu14], which triggered a long discussion and made Oliveri work on the question [Oli14].

Unfortunately, I do not have a solution to the formalisation problem either. The different approaches that people tried do not differ too much from each other, and in particular, the attempts of Li and myself in Agda are very close to Voevodsky's ideas in Coq. Still, to the best of my knowledge, what we will describe below has some novel aspects. Firstly, we discuss the construction of diagrams over inverse categories instead of only semi-simplicial types, but this is only a straightforward generalisation. Secondly, we show how the constructions can benefit if the indexing category satisfies associativity strictly. This is in particular interesting because we also show how to implement Δ_+ such that this condition is met, in a theory with a very reasonable additional assumption, namely the judgmental η (or *uniqueness*) law for Σ-types. Thirdly, we draw the connection to *very dependent types* [Hic96], which has already been mentioned in the discussion following Shulman's blog post [Shu14]; however, all details are left for future work, and for the moment, we restrict the discussion to describing why it is reasonable that very dependent types would solve the problem.

As our basic idea does not make use of the specific structure of Δ_+, so let

[3] `https://github.com/andrejbauer/andromeda`

us describe it in higher generality. We intentionally do not address all details in the following description, but we keep it a bit vague as different possibilities all seem reasonable. We then discuss afterwards under which circumstances which assumptions should be made. Assume we have a category $\mathcal{C}$ internalised in type theory in a somewhat naive sense:

- a *set* $\mathcal{C} : \mathcal{U}^0$ of objects
- for any two objects a *set* of morphisms, that is, a family $\mathsf{Hom} : \mathcal{C} \times \mathcal{C} \to \mathcal{U}^0$
- identities $\mathsf{i} : \forall(A : \mathcal{C}).\, \mathsf{Hom}(A, A)$
- a composition operator

$$\_ \circ \_ : \forall(A, B, C : \mathcal{C}).\, \mathsf{Hom}(B, C) \times \mathsf{Hom}(A, B) \to \mathsf{Hom}(A, C) \tag{9.4}$$

- laws for
  - the identity:

$$\forall(A, B : \mathcal{C}).\, \forall(f : \mathsf{Hom}(A, B)).\, (f \circ \mathsf{i}_A = f) \times (\mathsf{i}_B \circ f = f) \tag{9.5}$$

  - associativity:

$$\begin{aligned} &\forall(A, B, C, D : \mathcal{C}).\\ &\forall(f : \mathsf{Hom}(A, B)).\\ &\forall(g : \mathsf{Hom}(B, C)).\\ &\forall(h : \mathsf{Hom}(C, D)).\, (h \circ g) \circ f = h \circ (g \circ f) \end{aligned} \tag{9.6}$$

What we have written down above is exactly the definition of a *precategory* in the sense of [Uni13, Definition 9.1.1] (respectively [AKS15]), with the additional assumption that the objects and the morphisms both form a set. We further want $\mathcal{C}$ to be an *inverse* category, that is, there must not be an infinite sequence $\cdot \to \cdot \to \cdot \to \cdots$ of composable nonidentity morphisms (see Section 8.2). This can be expressed internally or it can simply be a property that we "know" by construction, for example because we have defined $\mathcal{C}$ such that it "is" the category Δ_+ (in the latter case, we could prove the appropriate internal statement as well). Note that this (appropriately formulated) implies that the identities are the only isomorphisms, and $\mathcal{C}$ in fact has to be a *(gaunt) category* (see [Uni13], [AKS15]); for example, Δ_+ will automatically be a gaunt category. Whether it is necessary (or useful) to have the "inverse"-property as an internal statement depends on how we want to proceed, as we will see below.

Let us describe how we can define a ("Reedy fibrant") diagram $\mathcal{C} \to \mathcal{U}$ in the sense of what we have illustrated in (9.1-9.3). This means that we want to define a function $P : \mathcal{C} \to \mathcal{U}$ which is in fact a functor. In order to do this we define, for each object $A : \mathcal{C}$, the *matching object* $M^A : \mathcal{U}$. Similar as in (9.1-9.3), we prefer

to not define P directly, but we define F such that $F^X : M^X \to \mathcal{U}$. Then, F^X does (by the usual translation between fibrations and families) represent the fibration from $P(X)$ to the matching object M^X, just as in Chapter 8.

The structure of $\mathcal{C}$ as an inverse category induces a partial order on its set of objects $\mathcal{C}$. For $X : \mathcal{C}$, assume we have defined M^X and F^X for all objects that are smaller than X. M^X should then be a type with several components: for each Y smaller than X (written $Y \prec X$), we need some τ_Y^X of type (explanation below)

$$\Pi_{f:\mathsf{Hom}(X,Y)} F^Y(\{\lambda(g : \mathsf{Hom}(Y,Z)).\tau_Z^X(g \circ f)\}_{Z \prec Y}). \tag{9.7}$$

We did not specify what exactly we mean by "for each Y smaller than X": it could be a Π-type, quantifying over all such Y, or it could also be a nested Σ-type with one component for each τ_Z^X, or it could potentially be something else. For the moment, we treat it as a black box. Of course, this means that we also cannot specify what exactly the notation $\{\cdots\}_{Z \prec Y}$ means; it could be a function $(\lambda Z.\cdots)$, a pair with $(Z \prec Y)$-many components, or possibly something else. We will discuss this below. The crucial point is that, in (9.7), τ_Y^X makes use of τ_Z^X, but only for $Z \prec Y$, so it at least does not seem completely unreasonable.

While it might look confusing at first sight, (9.7) it is really just the "formalisation" of the (meta-theoretical) definition of matching objects given in Section 8.2. Let us unfold the above formula for the case that C is Δ_+ to hopefully get some clarity:

- $M^{[0]}$ is the unit type $\mathbf{1}$, as there are no $Y \prec [0]$.
- Consequently, $F^{[0]}$ has type $\mathbf{1} \to \mathcal{U}$. This is the type of points.
- To define $M^{[1]}$, note that there is exactly one $Y \prec [1]$, namely $[0]$. $M^{[1]}$ therefore has one component which, for every $f : [0] \overset{+}{\to} [1]$, gives something in $F^{[0]}(\star)$. The last argument is $\star$ simply because there is no $Z \prec [0]$. If we simplify, we see that $M^{[1]}$ is a pair of two points.
- Therefore, $F^{[1]}$ has to be a family of types, indexed over pairs of points.
- It now becomes more interesting. $M^{[2]}$ has two components, as there are two objects less than $[2]$:
  - $\tau_{[0]} : \Pi_{f:[0]\overset{+}{\to}[2]} F^{[0]}(\star)$, a set of three points
  - $\tau_{[1]} : \Pi_{f:[1]\overset{+}{\to}[2]} F^{[1]}\{\lambda(g : Z \overset{+}{\to} [1]).\tau_Z(g \circ f)\}_{Z \prec [1]}$. Of course, Z here has to be $[0]$. Intuitively, the map f selects two points, and the map g then selects one of these two.
- We now see that $F^{[2]}$ is a family of types, indexed over "unfilled triangles" just as expected.
- One more step will certainly not hurt. $M^{[3]}$ has three components:

- $\tau_{[0]} : \Pi_{f:[0]\overset{+}{\to}[3]} F^{[0]}(\star)$, a set of four points
- $\tau_{[1]} : \Pi_{f:[1]\overset{+}{\to}[3]} F^{[1]}\left\{\lambda(g : Z \overset{+}{\twoheadrightarrow} [1]).\tau_Z(g \circ f)\right\}_{Z \prec [1]}$. Again, Z has to be $[0]$.
- $\tau_{[2]} : \Pi_{f:[2]\overset{+}{\to}[3]} F^{[2]}\left\{\lambda(Z \prec [2]).\lambda(g : Z \overset{+}{\twoheadrightarrow} [2]).\tau_Z(g \circ f)\right\}_{Z \prec [2]}$. Now, Z can take both the values $[0]$ and $[1]$. We could write write the type of $\tau_{[2]}$ as

$$\Pi_{f:[2]\overset{+}{\to}[3]} F^{[2]}\left(\begin{array}{l}[0] \mapsto \lambda(g : [0] \overset{+}{\twoheadrightarrow} [2]).\tau_{[0]}(g \circ f) \\ [1] \mapsto \lambda(g : [1] \overset{+}{\twoheadrightarrow} [2]).\tau_{[1]}(g \circ f)\end{array}\right), \tag{9.8}$$

  where we also intentionally leave unclear what the notation formally means.

The first serious question that we need to address is whether (9.7) type-checks. Note that, by definition, F^Y will have type $M^Y \to \mathcal{U}$. Its argument

$$\left\{\lambda(g : \mathsf{Hom}(Y, Z)).\tau_Z^X(g \circ f)\right\}_{Z \prec Y} \tag{9.9}$$

therefore has to be of type M^Y, meaning that it has to have components τ_Z^Y (for $Z \prec Y$). To see whether this is the case, we need to check that (for a given Z) the expression

$$\lambda(g : \mathsf{Hom}(Y, Z)).\tau_Z^X(g \circ f) \tag{9.10}$$

can serve as τ_Z^Y (a component of M^Y and not of M^X!). Looking at (9.7), the type of this τ_Z^Y would be

$$\Pi_{h:\mathsf{Hom}(Y,Z)} F^Z(\left\{\lambda(k : \mathsf{Hom}(Z, W)).\tau_W^Y(k \circ h)\right\}_{W \prec Z}). \tag{9.11}$$

By induction, we know that τ_W^Y (for all $W \prec Z$) will be the term corresponding to (9.10), with Z replaced by W. Making this substitution, we see that (9.11) becomes

$$\Pi_{h:\mathsf{Hom}(Y,Z)} F^Z(\left\{\lambda(k : \mathsf{Hom}(Z, W)).(\lambda(g : \mathsf{Hom}(Y, W)).\tau_W^X(g \circ f))(k \circ h)\right\}_{W \prec Z}), \tag{9.12}$$

and, after β-reducing,

$$\Pi_{h:\mathsf{Hom}(Y,Z)} F^Z(\left\{\lambda(k : \mathsf{Hom}(Z, W)).\tau_W^X((k \circ h) \circ f)\right\}_{W \prec Z}). \tag{9.13}$$

Let us go back to (9.10) which, after α-renaming, is $\lambda(h : \mathsf{Hom}(Y, Z)).\tau_Z^X(h \circ f)$. From (9.7), we conclude that the type of this expression is

$$\Pi_{h:\mathsf{Hom}(X,Z)} F^Z(\left\{\lambda(k : \mathsf{Hom}(Z, W)).\tau_W^X(k \circ (h \circ f))\right\}_{W \prec Z}). \tag{9.14}$$

Comparing (9.13) with (9.14), we see that the type we have matches the type we need if associativity in $\mathcal{C}$ is strict, as we need the equality $((k \circ h) \circ f) \equiv (k \circ (h \circ f))$ to hold judgmentally. At the same time, we did not make use of the identity

morphisms at all. We technically do not require them to be part of the structure of the indexing "category", and in particular, we do not need any judgmental laws for the identities.

Strictness of associativity is a rather strong requirement. The interesting observation here is that is can be satisfied for our main case of interest, namely the category Δ_+. We also get strictness of the identity laws "for free".

Ⓐ **Proposition 9.2.1.** *In MLTT with judgmental η-rule for dependent pair and function types, that is $x \equiv (\mathsf{fst}(x), \mathsf{snd}(x))$ for $x : \Sigma\,(a : A)\,.\,B(a)$, the category Δ_+ can be constructed in such a way that associativity and identity laws all hold strictly (and the same is true for the category Δ).*

Proof. We first restrict ourselves to the case of Δ_+. The construction is the canonical one with strictly increasing functions between finite sets. We only need to ensure that we define "strictly increasing" correctly, and check that associativity holds indeed strictly.

We write $\mathsf{Fin} : \mathbb{N} \to \mathcal{U}$ for the family of finite types: for any natural number k, the type $\mathsf{Fin}(k)$ is the type with exactly k elements [Uni13, Chapter 1.3 and Exercise 1.9]. Clearly, we can (for every k) define a function $>^k \colon \mathsf{Fin}(k) \to \mathsf{Fin}(k) \to \mathcal{U}$, usually used infix, such that $a >^k b$ is the proposition that a is greater than b. Now, we define (for any $h, k : \mathbb{N}$) the predicate $\mathsf{isIncr}_{h,k}$ by

$$\mathsf{isIncr}_{h,k} : (\mathsf{Fin}(h) \to \mathsf{Fin}(k)) \to \mathcal{U} \tag{9.15}$$

$$\mathsf{isIncr}_{h,k}(f) :\equiv \Pi_{i,j:\mathsf{Fin}(h)}(j >^h i) \to (f(j) >^k f(i)). \tag{9.16}$$

Of course, $\mathsf{isIncr}_{h,k}$ is a family of propositions as well.

This is all we need to construct Δ_+ with the required properties. Define the set of objects of Δ_+ to be $\mathbb{N}$. Morphisms from m to n are strictly increasing functions from $\mathsf{Fin}(n+1)$ to $\mathsf{Fin}(m+1)$:

$$\mathsf{Hom}(m, n) :\equiv \Sigma\,(f : \mathsf{Fin}(n+1) \to \mathsf{Fin}(m+1))\,.\,\mathsf{isIncr}_{n+1,m+1}(f). \tag{9.17}$$

The identities are the obvious ones: given $m : \mathbb{N}$, we have

$$(\mathsf{id}_{\mathsf{Fin}(m+1)}, \lambda i, j.\mathsf{id}_{j >^{m+1} i}) : \mathsf{Hom}(m, m). \tag{9.18}$$

Composition is defined pointwise. For all natural numbers m, n, o and morphisms $a : \mathsf{Hom}(m, n)$, $b : \mathsf{Hom}(n, o)$, we may assume that a is of the form (f, p) with $f : \mathsf{Fin}(n+1) \to \mathsf{Fin}(m+1)$ and $p : \mathsf{isIncr}_{n+1,m+1}(f)$, and similarly that b is of the form (g, q). We then simply set

$$(f, p) \circ (g, q) :\equiv (f \circ g, p \circ q), \tag{9.19}$$

where the composition on the right is the usual composition of functions. The slick part is that this does not only work for the function parts, but also for the proof components $(p \circ q)$.

Strictness of the associativity and identity laws now simply follows from the fact that they hold for ordinary functions with the canonical definition of composition, together with the assumed judgmental η-rule for dependent pair types.

For the category Δ, we simply replace $>^k$ by $\geq^k$, defined in an appropriate way. □

The problematic part with the definition of the matching objects as consisting of "components" τ_Y^X as given in (9.7) is of course how this collection of components is formed, and further how the different "levels" are put together (for example, $F^{[2]}$ is not a valid expression on its own, but it relies on $F[\mathsf{0}]$ and $F[\mathsf{1}]$). Voevodsky's [Voe12] and Herbelin's [Her15] approaches use Σ-types. Unfortunately, this works neither with our nor with their definitions (unless the theory is modified to support more judgmental equalities in one way or the other). The reason is that we cannot refer to a component τ_Y^X directly, but need to extract it using a bunch of projections. Consequently, the whole construction does then not type-check when defined for variable objects of $\mathcal{C}$.

For an (externally) fixed number n, however, this strategy does allow us to define the type of n-truncated semi-simplicial types. I have written a Haskell program which takes a number n and returns the definition of n-truncated semi-simplicial types as (valid[4]) Agda code [Kra14a].

Remark 9.2.2. Shulman [Shu14] made the following point: If there is such a program in Haskell, then it could certainly be implemented in type theory, as a function from $\mathbb{N}$ to some sort of expressions. If we had an interpreter of HoTT in itself, taking the expressions to types and terms, this would give us an internal definition of semi-simplicial types. Thus, the problem of defining such an interpreter seems to be at least as hard as defining semi-simplicial types. (At that time, Shulman referred to a Haskell program by Lumsdaine. At another occurrence, Lumsdaine stated that the program he had written did not generate the expressions for semi-simplicial types, but the semi-simplicial equality type instead. However, there was never any doubt that a programs as ours exists and can be written with a reasonable small amount of effort.)

Remark 9.2.3. One potential issue is that, in our construction, the matching objects are in general not *strictly* the limits they are supposed to be. This means that the diagram that we construct will not be Reedy fibrant. However, it will still be equivalent to the "correct" diagram, which, in many cases, should be sufficient.

The algorithm of our Haskell program uses Proposition 9.2.1. On the other hand, as everything in that setting is finite, we could easily avoid needing strict associativity. All we would have to do is "unfolding the τ_Y^X". In the definition above (9.7), a component τ_Y^X encodes *all* Y-faces (or Y-subdiagrams) at the same time: for *each* morphism $\mathsf{Hom}(X, Y)$, we have something in the corresponding

[4]This was tested for a range of possible n. However, our Haskell program does not produce very readable code, and can at most serve as a "proof of concept".

fibre $F^Y(\ldots)$. Instead, we could construct (an even larger) Σ-type which has one component for each such face. This is the approach in Herbelin's construction. It also does not type-check in the general case of a variable (not externally fixed) n.

What we could try doing is interpreting the "for every X and $Y \prec X$, a τ^X_Y" as a Π-type instead of a Σ-type. That is, we could try viewing τ^X as a function $\Pi_{Y \prec X}(\ldots)$, and actually even see τ as a function $\Pi_{X:\mathcal{C}_0}\Pi_{Y \prec X}(\ldots)$. In the same way, we could try to see the matching objects M as a function $M : \mathcal{C} \to \mathcal{U}$. The advantage is that, intuitively, functions (seem to) be easier to access than Σ-components that are "hidden" somewhere deep inside a long term. The problem is that $M(X)$ makes use of $M(Y)$ for all $Y \prec X$. This seems like it is not a big issue as all terms are defined in a well-founded manner, but it is something that functions can not do. The idea of having function types with this power, i.e. with types depending on the function's value at lower arguments, has been examined in the system NuPRL by Hickey, *Formal Objects in Type Theory Using Very Dependent Types* [Hic96]. Their "very dependent types" seem to be exactly what we would need here. Although it is currently not clear whether they can be introduced into HoTT appropriately, and a suitable semantics is yet to be found, we conjecture that they would solve the coherence problems with semi-simplicial types:

Conjecture 9.2.4 (Capriotti and Kraus)**.** It is possible to augment the current version of homotopy type theory with a suitable formulated version of very dependent types such that type-checking remains decidable, and in this system semi-simplicial types are definable. We further expect that all functors from a large class of inverse categories into $\mathcal{U}$ would be definable at least as long as (but maybe not only if) the inverse category is fixed (externally).

Remark 9.2.5. Given a type B, we can use our construction to define the *semi-simplicial equality type* that played a crucial role in Chapter 8 (see especially Remark and Corollary 8.5.2). Recall that we have defined the functor $\mathcal{E} : \Delta_+^{\mathrm{op}} \to \mathfrak{C}$ in Section 8.4. For an externally fixed number n, we can now easily define a type that is (homotopy) equivalent to $\mathcal{E}_{[\mathsf{n}]}$.

As in Remark 9.2.3, the diagram will not be Reedy fibrant. This would be the case if $\mathsf{Fin}(n) \to X$ was *strictly* isomorphic to $X \times \ldots \times X$ for numerals n, which it is not. Of course, simply copying the construction from Section 8.4 would not come with this issue, but would lead to a far more involved definition that would be difficult to write down in type theory.

For an object $[\mathsf{n}]$ of Δ_+^{op}, we define, by induction on the object $[\mathsf{i}]$,

- $\mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}]} : \mathcal{U}$, the $[\mathsf{i}]$-skeleton of an $[\mathsf{n}]$-simplex, i.e. a type that corresponds to $\mathcal{E}_{[\mathsf{n}]}$ where the cells on level $[\mathsf{i}+1], \ldots, [\mathsf{n}]$ are removed
- $\eta^{[\mathsf{n}]}_{[\mathsf{i}]} : B \to \mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}]}$, a canonical inhabitant of $\mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}]}$ for every $b : B$
- $\mathsf{F}_{[\mathsf{i}]} : \mathsf{Sk}^{[\mathsf{i}]}_{[\mathsf{i}-1]} \to \mathcal{U}$, the fibres.

Note that the matching object $M^{\mathcal{E}}_{[\mathsf{n}]}$ (as used in Chapter 8) is the $[\mathsf{n}-1]$-skeleton of $\mathcal{E}_{[\mathsf{n}]}$, and $\mathcal{E}_{[\mathsf{n}]}$ itself is $\mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{n}]}$. The cones $\tilde{\eta}_{[\mathsf{n}]}$ and $\eta_{[\mathsf{n}]}$ (as introduced in Section 8.4) can both be expressed as η here, using the appropriate indices, namely as $\eta^{[\mathsf{n}]}_{[\mathsf{n}-1]}$ and $\eta^{[\mathsf{n}]}_{[\mathsf{n}]}$.

For $\mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}]}$ and $\eta^{[\mathsf{n}]}_{[\mathsf{i}]}$, we allow $i \equiv [-1]$, which simplifies the definition (Herbelin [Her15] does the same in his construction of semi-simplicial types). We set

$$\mathsf{Sk}^{[\mathsf{n}]}_{[-1]} :\equiv \mathbf{1} \tag{9.20}$$

$$\eta^{[\mathsf{n}]}_{[-1]}(b) :\equiv \star \tag{9.21}$$

and, assuming that we have already defined $\eta^{[\mathsf{n}]}_{[\mathsf{i}-1]}$ and $\mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}-1]}$,

$$\mathsf{F}_{[\mathsf{i}]}(m) :\equiv \Sigma\,(x : B)\,.\,\eta^{[\mathsf{i}]}_{[\mathsf{i}-1]}(x) = m \tag{9.22}$$

$$\mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}]} :\equiv \Sigma\Big((\_, \tau_{[0]}, \ldots, \tau_{[\mathsf{i}-1]}) : \mathsf{Sk}^{[\mathsf{n}]}_{[\mathsf{i}-1]}\Big)\,.\,\Pi_{f:[\mathsf{i}]\overset{+}{\to}[\mathsf{n}]}\mathsf{F}_{[\mathsf{i}]}\left(\_, \overline{\tau_{[0]}}, \overline{\tau_{[1]}}, \ldots, \overline{\tau_{[\mathsf{i}-1]}}\right) \tag{9.23}$$

where we set

$$\overline{\tau_{[\mathsf{k}]}} :\equiv \lambda(g : [\mathsf{k}] \overset{+}{\to} [\mathsf{i}]).\tau_{[\mathsf{k}]}(f \circ g). \tag{9.24}$$

Finally, we define

$$\eta^{[\mathsf{n}]}_{[\mathsf{i}]}(b) :\equiv \left(\eta^{[\mathsf{n}]}_{[\mathsf{i}-1]}(b), \lambda\_.\left(b, \mathsf{refl}_{\eta^{[\mathsf{i}]}_{[\mathsf{i}-1]}(b)}\right)\right). \tag{9.25}$$

Note that this type-checks for *externally fixed* values of i in standard MLTT, while n may be a variable.

As said above, the diagram that we defined here is not fibrant. In particular, given $f : [\mathsf{k}] \overset{+}{\to} [\mathsf{m}]$, the function of type $\mathsf{Sk}^{[\mathsf{m}]}_{[\mathsf{m}]} \to \mathsf{Sk}^{[\mathsf{k}]}_{[\mathsf{k}]}$ that corresponds to $\mathcal{E}(f)$ is not simply a projection, and we thus need to show how to construct it. We define, more generally for any $j \geq i$, a function

$$\mathsf{Sk}^{[\mathsf{m}],[\mathsf{k}]}_{[\mathsf{j}],[\mathsf{i}]}(f) : \mathsf{Sk}^{[\mathsf{m}]}_{[\mathsf{j}]} \to \mathsf{Sk}^{[\mathsf{k}]}_{[\mathsf{i}]}. \tag{9.26}$$

There is a projection $\mathsf{Sk}^{[\mathsf{m}]}_{[\mathsf{j}]} \twoheadrightarrow \mathsf{Sk}^{[\mathsf{m}]}_{[\mathsf{i}]}$, and it is therefore enough to define the function $\mathsf{Sk}^{[\mathsf{m}],[\mathsf{k}]}_{[\mathsf{i}],[\mathsf{i}]}(f) : \mathsf{Sk}^{[\mathsf{m}]}_{[\mathsf{i}]} \to \mathsf{Sk}^{[\mathsf{k}]}_{[\mathsf{i}]}$. For $i \equiv [-1]$, this is trivial, and otherwise, by recursion on $[\mathsf{i}]$,

$$\mathsf{Sk}^{[\mathsf{m}],[\mathsf{k}]}_{[\mathsf{i}],[\mathsf{i}]}(f) :\equiv \lambda(s, \tau).\left((\mathsf{Sk}^{[\mathsf{m}],[\mathsf{k}]}_{[\mathsf{i}-1],[\mathsf{i}-1]}(f))\ ,\ \lambda(g : [\mathsf{i}] \overset{+}{\to} [\mathsf{k}]).\tau(f \circ g)\right). \tag{9.27}$$

Again, this type-checks as long as i is an externally fixed numeral.

9.3 Yoneda Groupoids

In this section, we want to use the natural ω-groupoid structure of types (especially of universes) to approach the problem of defining weak ω-groupoids. Our approach does not come close to capturing all ω-groupoids hat one might be interested in, but something that satisfies our definition of a *Yoneda groupoid* should be (or give rise to) a weak ω-groupoid for any reasonable definition. This section has in a different form been part of my first year report [Kra11, Section 2.4]. It is the only content from my first year report that I have chosen to include in this thesis. Similar ideas have been discovered and explored by at least two other researchers independently. Martín Escardó has started a private discussion in late 2012 about a similar concept that he called *univalent relations*. James Cranch has done work on concrete categories, which essentially use the same idea as we do here [Cra13].

Let us start with the motivation of this development. Around 2011, Altenkirch raised the problem of specifying when a (non-propositional) relation gives rise to an ω-groupoid. Of course, this is another instance of a formalisation problem that seems to require an infinite tower of coherence conditions. It is clear how to start: we need conditions which ensure that the relation is reflexive, transitive, and symmetric. But these conditions have to behave well when combined, and this is where the usual coherence problems begin.

Question 9.3.1 (Altenkirch, around 2011)**.** *Given a type $A : \mathcal{U}$ with a relation $R : A \times A \to \mathcal{U}$, and terms*

$$\mathrm{r} : \forall a.\, R(a,a) \qquad \textit{(“reflexivity”),} \tag{9.28}$$
$$\mathrm{s} : \forall (a,b).\, R(a,b) \to R(b,a) \qquad \textit{(“symmetry”),} \tag{9.29}$$
$$\mathrm{t} : \forall (a,b,c).\, R(a,b) \to R(b,c) \to R(a,c) \qquad \textit{(“transitivity”),} \tag{9.30}$$

2 how can we formalise (in type theory) the statement that they give A the structure of a weak ω-groupoid?

Admittedly, the question does not specify what it really asks for: what does it mean to “give A the structure of a weak ω-groupoid”? Given a complete answer what the question means is already non-trivial. It is possible to do this without already answering the question. For example, for the definition of semi-simplicial types, one could say that an acceptable definition is a function $\mathbb{N} \to \mathcal{U}$ together with a (meta-theoretical) proof that, for any fixed n, it gives a type that is equivalent to the type that (for example) our Haskell reference implementation gives. We choose to not clarify Question 9.3.1 completely and hope that the mentioned coherence properties and the section on semi-simplicial types together give sufficient intuition for the question to be understandable.

If we start to write down which conditions we want to impose on $\mathrm{r}, \mathrm{s}, \mathrm{t}$, we come up with the following list. We omit arguments which are (for the understanding) unimportant or inferable (the style of “implicit arguments”), and we omit quantifications over all used variables.

1. $\mathrm{s}(\mathrm{r}) = \mathrm{r}$
2. $\mathrm{s}(\mathrm{s}(p)) = p$
3. $\mathrm{t}(\mathrm{r}, p) = p$
4. $\mathrm{t}(p, \mathrm{r}) = p$
5. $\mathrm{t}(p, \mathrm{s}(p)) = \mathrm{r}$
6. $\mathrm{t}(\mathrm{s}(p), p) = \mathrm{r}$
7. ...

Some of the above terms are redundant as they are constructable from other terms. Unfortunately, these terms give to new coherence conditions. For example, for $p \equiv \mathrm{r}$, the third and the fourth both inhabit the type $\mathrm{t}(\mathrm{r}, \mathrm{r}) = \mathrm{r}$, and we certainly want them to be equal. Each new set of conditions gives rise to even more conditions that we need to impose. We could try to find *all* the conditions, formulate them uniformly, and try to write them down as a type-theoretical expression. I believe that this would be (at least) as hard as defining semi-simplicial types. Instead of specifying all the coherences, we attempt to reuse a coherent structure that we already have: the identities on types.

Definition 9.3.2 (Yoneda groupoid)**.** Let $A : \mathcal{U}$ be some type with a relation $R : A \times A \to \mathcal{U}$. For some type $\mathbf{U}$, we say that R is an $\mathbf{U}$-*Yoneda groupoid* if there is a function mapping every $a : A$ to an $X : \mathbf{U}$ such that X represents the structure of a's "equivalence class",

$$\mathsf{isYonedaGrp}_{\mathbf{U}}(R) :\equiv \Sigma\,(F : A \to \mathbf{U})\,.\,\Pi_{a,b:A} R(a,b) \simeq (F(a) = F(b)). \tag{9.31}$$

$\mathbf{U}$ could be any type, especially one in a universe above $\mathcal{U}$. The point is that $\mathbf{U}$ has a natural ω-groupoid structure which we want to use; therefore, using an $\mathbf{U}$ which involves a universe in some form seems to be particularly interesting if we work in a type theory with univalence.

However, simply picking the univalent universe $\mathcal{U}$ for $\mathbf{U}$ bears a serious problem. Let $n : \mathbb{N}$ be some number and consider the discrete set $A :\equiv \mathsf{Fin}(n)$ with the discrete relation $R(x, y) :\equiv (x = y)$, which is for all pairs either $\mathbf{1}$ or $\mathbf{0}$. We can try to define $F : \mathsf{Fin}(n) \to \mathcal{U}$ with the desired property. For distinguished $x, y : \mathsf{Fin}(n)$ we need to use two non-equivalent types $F(x)$, $F(y)$. This means that we need to find a family of n different types all of which have a trivial automorphism group. This is a difficult exercise which, to the best of my knowledge, was solved by Peter Lumsdaine and Michael Shulman in a private discussion. Later, they explained the construction in a mailing list discussion on rigid types, which was started by Martín Escardó [Hmail]. For us, a much easier and generalisable solution is to choose $\mathbf{U} :\equiv \mathsf{Fin}(n) \times \mathcal{U}$, and set $F(x) :\equiv (x, \mathbf{1})$. Then, it is easy to see that $F(x) = F(y)$ is indeed contractible if $(x = y)$ and empty otherwise. In this

example, $\mathsf{Fin}(n)$ acts as a "labelling set" to produce multiple distinguishable copies of a specific type (here $\mathbf{1}$). In general (and in particular here), one can alternatively very often simply use $\mathbb{N}$ as a set of labels, i.e. use $\mathbf{U} :\equiv \mathbb{N} \times \mathcal{U}$. This will become clearer in the examples below (see Example 9.3.4).

Definition 9.3.2 is inspired by two different formalisations of equivalence relations in the proof-irrelevant case and can actually be understood as a combination of those. The first is what we could call the *Yoneda characterisation* of equivalence relations; that is, for some propositional relation $S : A \times A \to \mathcal{U}^{-1}$, we can say that S is an equivalence relation if and only if we have, for all $a, b : A$,

$$S(a,b) \leftrightarrow (\Pi_{x:A} S(a,x) \leftrightarrow S(b,x)) \,. \tag{9.32}$$

Unfortunately, it is not possible to generalise this in the straightforward way to higher relations. One might think that replacing $\leftrightarrow$ by $\simeq$ could suffice, but unfortunately, there would be an unwanted level shift. Let again R be some (not necessarily propositional) relation R. Say that, for all $a, b : A$, we require

$$R(a,b) \simeq (\Pi_{x:A} R(a,x) \simeq R(b,x)) \,. \tag{9.33}$$

This does not give R a satisfactory higher groupoid structure for at least two reasons. First, in the right-hand type, we quantify over all x. If multiple distinguished elements of A are in the same "equivalence class" (i.e. related by R), then the right-hand type will necessarily be larger than we want. Even more problematic is the second issue, the unwanted level shift, which makes even very simple examples fail. Consider $A :\equiv \mathbf{1}$, choose some $n : \mathbb{N}$, and set $R(\star, \star) :\equiv \mathsf{Fin}(n)$. There is (for $n \geq 1$) certainly an associated groupoid structure; for example, we could have the cyclic group $\mathbb{Z}/(n)$. However, there is (unless n is 1 or 2) no equivalence between the sets $\mathsf{Fin}(n)$ and $(\mathsf{Fin}(n) \simeq \mathsf{Fin}(n))$, the latter of which is equivalent to $\mathsf{Fin}(n!)$.

The second source of inspiration for Definition 9.3.2 has been the definition of an equivalence class by Voevodsky [Voe10b; Voe13b] which we have already explained in Example 5.2.5 and briefly repeat here. Recall that, for $P : A \to \mathcal{U}^{-1}$, we may define the statement that P *is an equivalence class* of the (propositional) *equivalence* relation $S : A \times A \to \mathcal{U}^{-1}$ by

$$\mathsf{isEquivClass}(P) :\equiv \|\Sigma\,(a : A)\,.\,\forall(b : A).\, S(a,b) \leftrightarrow P(b)\|. \tag{9.34}$$

Then, the quotient would be defined as the collection of equivalence classes. In the theory we consider, it would necessarily live in a higher universe than A.

Remark 9.3.3. The first definition for Yoneda groupoids that I came up with used equivalence classes, which made it very similar to the construction of quotients by Voevodsky. I have later simplified it and so that it now uses a single $F : A \to \mathcal{U}$ which also adds labels from the set $\mathbb{N}$ to distinguish classes that are equivalent but distinct.

As the relation R of a Yoneda groupoid is by definition exactly given by the equality of some type, it is clear that we can construct the terms $\mathrm{r}, \mathrm{s}, \mathrm{t}$ and prove all the coherence laws listed on page 178. In particular, it one came up with some definition of a weak ω-groupoid, it would certainly be desirable that every Yoneda groupoid is (or gives rise to) an ω-groupoid in that sense.

If we are given a relation R, it may be proved to be a Yoneda groupoid in more than one way. This is to be expected: For example, with $A :\equiv \mathbf{1}$ and $R(\star, \star) :\equiv \mathsf{Fin}(6)$, two distinguishable groupoid structures would be $\mathbb{Z}/(6)$ and the symmetric group S_3. Put differently, $\mathsf{isYonedaGrp}_{\mathbf{U}}(R)$ is not necessarily propositional. As so often, it is the proof that makes the choice of the precise structure.

We can easily form some kind of quotient of a Yoneda groupoid if the theory supports propositional truncations. However, the quotient that we define will not live inside the same universe. This is to be expected, as we (without higher inductive types) can only get types that do not have some fixed truncation level by going up the universe hierarchy.

Let A be a type with a relation R that is a Yoneda groupoid with the proof (F, p), where $F : A \to \mathbf{U}$ and $p : \Pi_{a,b:A} R(a,b) \simeq (F(a) = F(b))$, Define the "carrier" of the quotient by

$$Q :\equiv \Sigma\,(x : \mathbf{U})\,.\,\|\Sigma\,(a : A)\,.\,F(a) = x\| \tag{9.35}$$

and the projection into the carrier to be

$$q : A \to Q \tag{9.36}$$

$$q(a) :\equiv (F(a), \left|a, \mathsf{refl}_{F(a)}\right|). \tag{9.37}$$

For any $a, b : A$, we then have

$$(q(a) = q(b)) \;\simeq\; (F(a) = F(b)) \;\simeq\; R(a,b), \tag{9.38}$$

which can be understood as a form of *soundness and exactness*.

However, it is not clear what the induction (elimination) principle of that quotient should be. Of course, it is *not* enough to have a map $f : A \to B$ with a proof $\Pi_{a,b:A}\,(R(a,b) \to f(a) = f(b))$ if we want to construct a map $Q \to B$. This will in general only suffice if B is a set. The absence of a reasonable induction (or even recursion) principle is certainly a serious drawback of the construction.

Some examples of relations that can be equipped with the Yoneda groupoid structure are the following:

Example 9.3.4. (i) For any type A, we have $\mathsf{Id}\omega$, the *identity ω-groupoid* as consider by other authors [ALR14]. As we know, A with its equality has the structure of such an ω-groupoid, and indeed, it can be turned into a Yoneda groupoid. Even more, it is as trivial as it can be expected to be.[5] We simply choose $\mathbf{U}$ to be A, and F to be the identity on A.

[5] This is not the case in the work by Altenkirch, Li and Rypacek [ALR14] from which we have the example $\mathsf{Id}\omega$ from.

(ii) For any $n : \mathbb{N}$, we have a Yoneda groupoid with $A :\equiv \mathbf{1}$ and $R(\star, \star) :\equiv \mathsf{Fin}(n!)$, where $n!$ denotes the factorial: We simply define $F(\star) :\equiv \mathsf{Fin}(n)$ and use the equivalence of the automorphism group of $\mathsf{Fin}(n)$ and $\mathsf{Fin}(n!)$. There are many other group structures, such as $\mathbb{Z}/(n)$, but unfortunately, we do not get them with our construction. Note that this example is somewhat boring as, for (fixed) finite truncation levels, groupoid structures could be defined directly.

(iii) We can freely combine types and relations that come with a Yoneda groupoid structure. If we have $A, B : \mathcal{U}$ and $R : A \times A \to \mathcal{U}$ as well as $S : B \times B \to \mathcal{U}$, we can define the canonical relations on $A + B$, namely

$$(R + S)(\mathsf{inl}(a_1), \mathsf{inl}(a_2)) :\equiv R(a_1, a_2) \tag{9.39}$$
$$(R + S)(\mathsf{inl}(a), \mathsf{inr}(b)) :\equiv \mathbf{0} \tag{9.40}$$
$$(R + S)(\mathsf{inr}(b), \mathsf{inl}(a)) :\equiv \mathbf{0} \tag{9.41}$$
$$(R + S)(\mathsf{inr}(b_1), \mathsf{inl}(b_2)) :\equiv S(b_1, b_2) \tag{9.42}$$
$$\tag{9.43}$$

and also on $A \times B$,

$$(R \times S)((a_1, b_2), (a_2, b_2)) :\equiv R(a_1, a_2) \times S(b_1, b_2). \tag{9.44}$$

If we have $F : A \to \mathbf{U}_1$ and $G : B \to \mathbf{U}_2$ with the corresponding proofs, then $(F+G) : (A+B) \to (\mathbf{U}_1+\mathbf{U}_2)$ and, respectively, $(F \times G) : (A \times B) \to (\mathbf{U}_1 \times \mathbf{U}_2)$ give the proofs that the sum and product of Yoneda groupoids are again Yoneda groupoids.

Our definitions immediately give rise to the question: when does the pair $(\mathbf{U}, F)$ exist? If we consider the case that $\mathbf{U}$ is a universe, our question amounts to asking for which types the operator $\Omega(\mathcal{U}, \_)$ has a local inverse, i.e. for which types X there is a type Y with $(Y = Y) \simeq X$. In the example $X \equiv \mathsf{Fin}(n!)$, a solution exists, namely $Y \equiv \mathsf{Fin}(n)$ However, can we find an appropriate structure if X is not a discrete set where the number of terms equals a factorial? Christian Sattler has sketched a proof that such a group does exist indeed for $X \equiv \mathbf{3}$, which can (probably) be generalised. Sattler further conjectured that for any n-groupoid there always is an $n+1$-groupoid with the required property, but even if this is the case, it is not clear whether it can be done in type theory. I do not know whether there are solutions or partial solutions to the corresponding problem of ω-groupoids (or ω-categories, (ω, n)-categories, ...), although the question seems to be very famous for ordinary groups. For example, by a result of Guohua Qian, groups with n elements (with n odd and not having a divisor p^4 for any prime p) cannot arise as the automorphism group of a finite group [Qia03]. On the other hand, all groups are outer automorphism groups of simple groups, see Droste, Giraudet, Rüdiger [DMG01]. The point is that, in our setting, we are not restricted to groups of the same "dimension" as a given group; that is, even if a group G is not the

automorphism group of another group, it may still happen to be the automorphism 2-group of some 2-group.

9.4 Set-Based Groupoids

In homotopy type theory, we have (at least) two plausible notions of (1)-"groupoid". We can consider a set of objects, and for any pair of objects, a set of morphisms between them, together with all the data needed to make it a groupoid. We call this a *set-based* representation. Another possibility is to simply consider 1-types as groupoids.

From a groupoid in set-based representation, it seems to be fairly simple to get a 1-type. We do not discuss this here; it is a simple higher inductive type, and in particular should follow from the Rezk completion [AKS15]. A question of Altenkirch is whether a 1-type can be transformed into a set-based groupoid as well. One advantage of representing types of higher truncation levels in the set-based way could be that it simplifies the handling of coherence issues, in particular when trying to model type theory [ALR14].

Here, we show that we can derive the set-based representation for a 1-type (in the sense of Altenkirch) if and only if the task is trivial, i.e. for 1-types that are actually sets. However, for a given 1-type with braided loop spaces ("weakly abelian" loop spaces), we can construct a weak form of the second representation which only includes all loop spaces instead of all path spaces.

9.4.1 Negative Results

Let us begin with a precise definition.

Definition 9.4.1 (Altenkirch, around 2012)**.** Given a type A, we say that A is *set-based representable* if there is

$$h : \|A\|_0 \times \|A\|_0 \to \mathcal{U} \tag{9.45}$$

such that there is a proof that, for any $a, b : A$, we have

$$h(|a|_0, |b|_0) \simeq (a =_A b). \tag{9.46}$$

Note that the above definition does not require A to be 1-truncated. Altenkirch wanted to know whether all types are set-based representable. Capriotti observed that, if A is set-based representable, then

$$|-| : A \to \|A\|_0 \tag{9.47}$$

has a section. The other direction of this statement is wrong. A section does allow us to construct a reasonable h, but we will in general not get the family of proofs of equivalence. This would require a retraction (which would immediately imply that A is a set). We can show a strong improvement of this observation:

Proposition 9.4.2. *A type is set-based representable if and only if it is a set.*

Proof. The first direction ("if") is trivial.

For the other direction ("only if"), let us start with a simple observation:

Observation. *Given types A, B, functions $j : A \to B$ and $k : B \to \mathcal{U}$ as well as $a_1, a_2 : A$ and $p : a_1 = a_2$. Then, there are three functions $k(j(a_1)) \to k(j(a_2))$, namely the ones coming from*

- *transporting along p*
- *transporting along $\mathsf{ap}_j p$*
- *transporting along $\mathsf{ap}_{k \circ j} p$.*

These three functions are all equal (trivial by induction on p).

Assume now that we have a type A, together with

$$h : \|A\|_0 \times \|A\|_0 \to \mathcal{U} \tag{9.48}$$

and

$$c : \Pi_{a,b:A} \left(h(|a|_0, |b|_0) \simeq (a =_A b) \right). \tag{9.49}$$

We need to show that A is a set. First, assume $a, b_1, b_2 : A$ and $p : b_1 = b_2$. Consider the following square, in which the horizontal maps are given by the function part of the equivalences $c(a, b_1)$ and $c(a, b_2)$:

$$\begin{array}{ccc} h(|a|_0, |b_1|_0) & \xrightarrow{\quad c(a,b_1) \quad} & a =_A b_1 \\ \Big\downarrow{\scriptstyle \mathsf{transport}^{\lambda b:A.h(|a|_0,|b|_0)}(p, \_)} & & \Big\downarrow{\scriptstyle \_ \cdot p} \\ h(|a|_0, |b_2|_0) & \xrightarrow{\quad c(a,b_2) \quad} & a =_A b_2 \end{array}$$

By path induction, the square commutes (up to homotopy, of course). But, by the observation stated above, the functions

$$\mathsf{transport}^{\lambda b:A.h(|a|_0,|b|_0)}(p, \_) \tag{9.50}$$

and

$$\mathsf{transport}^{\lambda x:\|A\|_0.h(|a|_0,x)}(\mathsf{ap}_{|-|_0}(p), \_) \tag{9.51}$$

are equal.But (9.51) is independent of p, as $\mathsf{ap}_{|-|_0}(p)$ equals $\mathsf{ap}_{|-|_0}(q)$ for any $q : b_1 = b_2$. Looking at the right vertical arrow, this means that composition with p equals composition with q, which is only possible if p and q are equal. □

9.4.2 Positive Results

The notion of set-based representability of Definition 9.4.1 is too strong. Instead, we suggest:

Definition 9.4.3. We call a type A *reduced set-based representable* if there is a single-indexed family

$$g : \|A\|_0 \to \mathcal{U} \tag{9.52}$$

of types which, for all $a : A$, satisfies

$$g(|a|_0) \simeq (a =_A a). \tag{9.53}$$

Alternatively (and logically equivalently), A is reduced set-based representable if the map $\lambda a.(a = a) : A \to \mathcal{U}$ factors through $\|A\|_0$.

We consider this a reasonable alternative to Definition 9.4.1. The higher structure of a type is fully specified by its higher loop spaces (Lemma 2.2.17). Consequently, in Definition 9.4.1, "too much" information is given; the HIT-construction can be done with the reduced data given in Definition 9.4.3.

Remark 9.4.4. Assume A is reduced-representable by (g, d) (where d is the family of proofs, as in Definition 9.4.3). We will see that A does not need to be a set. One approach to construct a "full" representation (h, c) (as in Definition 9.4.1) would be to set

$$h(x, y) :\equiv (x = y) \times g(x). \tag{9.54}$$

Intuitively, this is correct as $x = y$ is always propositional: $(a = b)$ is isomorphic to $(a = a)$ if $(a = b)$, and empty if $\neg(a = b)$. However, we will not be able to construct the general c. For any $a, b : A$, the best we could get would be

$$\|h(|a|_0, |b|_0) \simeq (a =_A b)\|_{-1}, \tag{9.55}$$

but even this would require LEM_{-1}.

We now work towards proving a "positive" result.

Definition 9.4.5. We say that a type A has *braided* loop spaces if, for all $a : A$, the loop space $a = a$ is commutative ($p \cdot q = q \cdot p$ for all $p, q : a = a$).

Caveat: Having braided loop spaces is a much stronger statement than saying that the fundamental group at every basepoint is abelian. Of course, if A is 1-truncated, they become the same.

The certainly most obvious examples of types with braided loop spaces are sets (for which it is trivial). More interesting are loop spaces themselves:

Lemma 9.4.6. *For any type A, and any element $a : A$, the type $\Omega(A, a)$ has braided loop spaces.*

Proof. For given A and a, and for any $p : a = a$, and any $s, t : p = p$, we need to show $s \cdot t = t \cdot s$. The Eckmann-Hilton argument, see [Uni13, Theorem 2.1.6], shows this equality in the case that p is reflexivity. To complete the argument, we consider $s \cdot \mathsf{refl}_{p^{-1}}$ and $t \cdot \mathsf{refl}_{p^{-1}}$ instead, where, in this case, $\cdot$ denotes 2-composition. Those commute, and canceling the trivial proofs, we get what we need. This essentially is the Eckmann-Hilton argument. □

The property of having braided loop spaces is interesting for the following reason:

Theorem 9.4.7. *Every* 1*-type with braided loop spaces is reduced-representable.*

Proof. We want to show that $f : A \to \mathcal{U}$, defined by $f \equiv \lambda a : A.(a = a)$, factors through $\|A\|_0$. By Theorem 8.10.2 with $n \equiv 1$, we need to show that, for all a in A, the function $\mathsf{ap}_f : (A = A) \to ((a = a) = (a = a))$ is weakly constant.

Consider any points $a, b : A$ and any proof $p, q : a = b$. We then have

$$\mathsf{ap}_f(p) : (a = a) = (b = b) \tag{9.56}$$

$$\mathsf{ap}_f(q) : (a = a) = (b = b). \tag{9.57}$$

Clearly, we are done if we can show $\mathsf{ap}_f(p) = \mathsf{ap}_f(q)$. Recall that we have a function $\mathsf{idtoeqv}$, mapping equalities of types to equivalences. The univalence axiom says exactly that this function is an equivalence. In particular, it is enough to show that the induced functions $(a = a) \to (b = b)$ are equal. But now we see that the function induced by (9.56) is

$$\lambda(s : a = a).p^{-1} \cdot s \cdot p \tag{9.58}$$

and the function induced by eq. (9.57) is

$$\lambda(s : a = a).q^{-1} \cdot s \cdot q; \tag{9.59}$$

this follows immediately by path induction. In the case $a \equiv b$, we have by assumption

$$(p^{-1} \cdot s \cdot p) = (p^{-1} \cdot p \cdot s) = s = (q^{-1} \cdot q \cdot s) = (q^{-1} \cdot s \cdot q). \tag{9.60}$$

□

9.5 Further Notes on Related Work and Conclusions

Chapter 3 The beginning of our thesis contains simple results that are related to Hedberg's Theorem. The *Generalised Local Hedberg Argument* (Theorem 3.2.1) itself is very easy to prove and certainly not surprising: it essentially shows that a reflexive n-truncated relation which implies equality on a type is

enough to conclude that the type is $(n + 1)$-truncated. It is a straightforward generalisation of a collection of theorems that are listed in the textbook on HoTT, such as [Uni13, Theorem 7.2.2], [Uni13, Theorem 7.2.9], and Hedberg's Theorem ([Uni13, Theorem 7.2.5]). Although simple, I believe that this result is useful. It is sometimes involved to show that a type has some truncation level, and Theorem 3.2.1 can turn out to be a valuable technical tool for streamlining such proofs; after all, already Hedberg's Theorem itself (which seems much more restricted) is sometimes used to show that a type is a set. We have seen evidence for this claim: the already technically involved argument in Theorem 8.10.2 would probably be significantly longer if we did not have this tool at hand.

Chapter 4 Considering "Hedberg-style" arguments has naturally led us to examining the propositional truncation more carefully. We have compared it with alternative notions of anonymous existence. In particular, we have defined *populatedness*: A type is populated (in our sense) if every weakly constant endofunction on it has a fixed point. It is a main result of the section that this property is propositional. Therefore, populatedness is a propositional notion of existence that is definable in MLTT. Unfortunately, the concept seems to be less useful than propositional truncation.

Chapter 5 While we have shown that a type has a constant endofunction if and only if it has split support, and (without any additional effort) we can factor such a weakly constant endomap through the propositional truncation, it seems to be impossible to do this in general if we drop the condition that domain and codomain of the function coincide. For example, a constant map from the sum of two propositions factors through the propositional truncation, meaning that the join of propositions exists even in the absence of higher inductive types. Retrospectively, it is not surprising that the factorisation is possible in this case as the constancy can be expressed in a way that does not have any coherence problems. More precisely, we can "coherify" it by replacing the "problematic" parts of the constancy proof by the canonical ones, something that is not possible for general weakly constant functions $(A \to B)$. An open question that especially Martín Escardó and I are interested in is whether one can derive a (constructive or homotopical) taboo from the assumption that every weakly constant function factors through the propositional truncation. More precisely, does the assumption

$$\forall (X\, Y : \mathcal{U}).\, \forall (f : X \to Y).\, \mathsf{const}_f \to \|X\| \to Y \tag{9.61}$$

allow us to derive a "suspicious" statement, such as UIP for all types, some form of the axiom of choice, or some form of excluded middle? For the assumption that every type is collapsible, we have established a corresponding result in Section 4.3.1: It implies that all equalities are decidable.

One example of a weakly constant function for which we do not know the status of its factorisability is the function (6.71) which maps, for a given transitive type

X, any point $x : X$ to $(X, x) : \mathcal{U}_\bullet$. Being able to factor it amounts to knowing an inhabitant of

$$\forall (X : \mathcal{U}).\, (\|X\| \times \mathsf{isTransitive} X) \to \Sigma\, (A : \mathcal{U})\, .\, A \times (X \to A =_{\mathcal{U}} X). \tag{9.62}$$

We can use the functoriality of $\|-\|$ to see that (9.62) implies

$$\forall (X : \mathcal{U}).\, (\|X\| \times \mathsf{isTransitive} X) \to \Sigma\, (A : \mathcal{U})\, .\, A \times \|A =_{\mathcal{U}} X\|. \tag{9.63}$$

The latter statement can be read as: Given a transitive merely inhabited type, can we find an inhabited type that is merely equal to the first? Finally, we can ask one more short question if we drop the transitivity condition and arrive at

$$\forall (X : \mathcal{U}).\, \|X\| \to \Sigma\, (A : \mathcal{U})\, .\, A \times \|A =_{\mathcal{U}} X\| : \tag{9.64}$$

If we have exact knowledge of a type and mere knowledge about an inhabitant, can we "trade" it for mere knowledge of (the structure of) the type and exact knowledge about an inhabitant? We do not expect that (9.62), (9.63) or (9.64) is derivable, but does any of these three assumptions allow us to conclude a taboo? These problems are open.

Chapter 6 When I have presented the "myst puzzle" (Theorem 6.4.6) first (in private discussions and on the HoTT blog [Kra13b]), it has caused a lot of amazement, in fact far more amazement than I had expected. The possibility to recover $n : \mathbb{N}$ from a term $x : \|\mathbb{N}\|$ that was (secretly) defined to be $|n|$ seems very counter-intuitive at first sight. On the other hand, if we are given x, it is clear that we can find out at least *something* about it because we can observe how functions behave when applied on x. The two facts that are not immediately obvious are thus: First, the "tests" that we can do can indeed reveal *everything* about x; and second, the "tests" can be summarised and formulated as a single internally defined function term. We have seen that this does not only work for $\mathbb{N}$ (in fact, $\mathbb{N}$ is just the example that I considered best), but for all transitive types.[6] However, even this is only a requirement if we want to define the complete term myst. The proof of homogeneity is irrelevant for the actual computation. Thus, we can ("from the outside", e.g. as an Agda programmer who does not want to open or search some file which contains important definitions) recover a for *any* $x : \|A\|$ that was (secretly) defined to be $|a|$. We do not have to provide a concrete proof of $\mathsf{transitive}(A)$; working with the *assumption* $\mathsf{transitive}(A)$ (or simply leaving a "hole" in Agda) will work.

Cohen, Coquand, Huber, and Mörtberg have implemented an experimental simple type-checker with an evaluator for closed terms (including terms that use univalence) [CCHM14], based on the cubical set model of type theory by Bezem,

[6] The term "transitive" was suggested by Christian Sattler, and I find it better for the concept than "homogeneous", which I had used in the blog post.

Coquand, and Huber [BCH14]. As an example, they have implemented the function myst.[7] As Simon Huber told me in a private conversation, it took "forever" to type-check (but, giving it enough time, everything worked fine). This is not surprising as the term myst becomes very long if we try to unfold it. In fact, already its type is difficult to unfold!

Chapter 7 The construction of strict n-types from univalence alone is tricky (it has been an open problem of Princeton's special year for a reason), but it is probably not surprising that it is possible. What is nice is that, with our first construction where we show that $\mathcal{U}_n^n$ is a strict $(n+1)$-type, we do not loose any universe level (see Remark 7.1.2(ii)). This was maybe not to be expected, especially since the "more straightforward" approach of "wrapping" the type of booleans (see Section 7.5) will necessarily loose a level.

The "local-global looping" lemma (Main Lemma 7.4.2), saying that a loop in the universe ("global") corresponds to a family of ("local") loops in the type it is based at, is essentially a not very difficult consequence of strong function extensionality. I had presented it in my talk in Princeton in April 2013. However, at that time, its name was rather boring (something like "Lemma 1"); calling it "local-global" is inspired by *Hasse's local-global principle*, a famous concept from algebraic number theory. The local-global looping lemma was later reused (or rediscovered) by Licata and Brunerie [LB13].

Section 7.6, the part about connectedness, is a bit away from the main development. The motivation was to see whether we can trivialise the lower homotopy groups, not only make sure that the n-th group is non-trivial. The construction is somewhat straightforward and it is not surprising that it works, but it was (at least for Christian Sattler and myself) also not obvious that it would really work before we did it. We had to use the impredicative encoding very carefully to make sure that we quantify over the correct universes, and we think that, in principle, it is for this sort of constructions possible to run into cyclic dependencies. For example, if we need some operator T on types which can be defined but only in a way that increases the universe level, and we need to apply T on $T(X)$ somewhere (i.e. compose T with itself), this clearly is problematic. The main difficulty of that part was the formalisation in Agda, which benefits from a lot of effort by Christian Sattler. In some way, the formalisation of Section 7.6 could probably serve as a case study for formalising with only a minimum of judgmental computation rules.

The article "Higher Homotopies in a Hierarchy of Univalent Universes", jointly written with Sattler, is to appear in *Transactions on Computational Logic*, containing essentially the results of Sections 7.2 to 7.4.

Chapter 8 It always feels a bit unsatisfying that, in order to define a function $(\|A\|_{-1} \to B)$, one needs to come up with some ad-hoc auxiliary proposition P

[7] https://github.com/simhu/cubical/blob/master/examples/Kraus.cub

such that $A \to P$ and $P \to B$. One can often construct P by "describing a particular unique element of B" (see e.g. [Uni13, Exercise 3.19]). With Theorem 8.8.5, we have described what a map $(\|A\|_{-1} \to B)$ "is" in general, although it is only applicable if we have Reedy ω^{op}-limits. It should also work with other forms of the "infinite Σ-types" concept. In particular, very dependent types should be sufficient if one can formulate a version of MLTT which includes them; see Conjecture 9.2.4.

On the other hand, the result that is more likely to be usable "in practice" is the finite special case Theorem 8.9.6, where B is an n-type for some fixed finite n. Unfortunately, it is (as we believe) impossible to internalise Theorem 8.9.6 for a *variable* n in the system that we mainly consider in this thesis (and we are convinced that it is impossible to formalise Theorem 8.9.6 in, say, Agda). To be precise, it already seems to be impossible to formalise the semi-simplicial equality types. As for semi-simplicial types, it should be possible to write a program which takes n as input and outputs the correct Agda source code which formalises Theorem 8.9.6 for n.

It is a somewhat obvious question to ask whether we can generalise Theorem 8.8.5 from $\|-\|_{-1}$ to $\|-\|_n$, and we have discussed this in Section 8.10; however, we have only solved one special case, and leave the rest as future work.

Another tool for eliminating truncations was given by Escardó and Xu [EX15, Lemma 3]. Their "exiting [propositional] truncations" principle says that, if A is a family of propositions indexed over $\mathbb{N}$ such that $A(n)$ implies the decidability of $A(m)$ for all $m < n$, then

$$\|\Sigma\,(n : \mathbb{N})\,.\,A(n)\|_{-1} \to \Sigma\,(n : \mathbb{N})\,.\,A(n). \tag{9.65}$$

This is very useful in their situation as it allows them to get a proof of uniform continuity for functions $\mathbf{2}^{\mathbb{N}} \to \mathbf{2}$ from the truncated version of such a proof. As $\Sigma\,(n : \mathbb{N})\,.\,A(n)$ is a set, we know that a function of type (9.65) corresponds exactly to a weakly constant endofunction on $\Sigma\,(n : \mathbb{N})\,.\,A(n)$ by Proposition 8.1.2; and indeed, such a weakly constant endofunction is constructed easily: given $(n, a) : \Sigma\,(n : \mathbb{N})\,.\,A(n)$, we know that $A(0), A(1), \ldots, A(n)$ are all decidable, and we simply return the smallest n' for which there is an element of $A(n')$. The authors of [EX15] do not assume that the propositional truncation exists, so they show that the the "type in the middle" (see the beginning of Chapter 8), but disregarding this, their proof is essentially the unfolded version of the proof sketched above.

The work presented in the discussed chapter gives rise to the following question, related to the discussion of semi-simplicial types above. The number n in Theorem 8.9.6 really has to be an externally fixed constant or our proof will not go through. One could therefore ask whether we can do the proof if n is a variable. This would amount to internalising a family of proofs, indexed over $\mathbb{N}$. It should then be possible to actually construct what is intuitively an "infinite Σ-type", and we could reasonably hope that Theorem 8.8.5 can be proved in HoTT without any further assumptions. I believe that the answer is negative. I do already not believe

that the equality semi-simplicial type can be defined internally, i.e. I believe that $\mathcal{E} : \mathbb{N}^+ \to \mathcal{U}$ does not exist as a term. This seems to be very similar to the problem of defining semi-simplicial types internally (see Section 9.2), but, as far as I can see, there is neither a reason why it should be easier nor a reason why it should be harder. Having said that, I still expect that if one could solve one problem (possibly by extending the theory in a reasonable way), one could also solve the other. This means that, although I can prove Theorem 8.9.6 internally if n is instantiated with any number (Theorem 8.9.6), I conjecture that it is impossible to prove it for a variable $n$in the standard version of homotopy type theory. What I think is certainly possible is to write a program in any standard programming language that takes a number n as input and prints out the formalised statement of Theorem 8.9.6 (in the syntax of a proof assistant such as Coq or Agda) together with a proof. I suspect that everything we did on propositional truncations would be formalisable in Voevodsky's HTS [Voe13a] or another type system with two layers, where the inner layer corresponds to HoTT and the outer layer has a strict identity type, as Altenkirch, Capriotti and myself are currently considering [ACK]. This would be conjectures that could be checked in further investigations.

On Infinite Structures It would be feasible to have a type theory which can deal with "infinite Σ-types" for many reasons. We could define semi-simplicial types, and we could reasonably hope for an internal definition of weak ω-groupoids, and an internal model of univalent type theory. As we have discussed in Section 9.2, Hickey's *very dependent types* could potentially make it possible to achieve this without changing the currently commonly used theory much. However, we need to be careful: even if we find a nice extension of the currently considered theory which allows the construction of an internal model of univalent type theory, it is not clear whether we could equip the model itself with the same sort of "infinite Σ-types"; thus, it is unclear whether we could get an interpretation of the theory in itself. This means it would be preferable to perform the construction within the currently considered type theory, but unfortunately, it is to be expected that this is impossible.

One earlier attempt to define weak ω-groupoids syntactically was made by Altenkirch and Rypacek [AR12], who suggested a very involved construction. Originally, a definition was given by Grothendieck [Gro83], which was later simplified, and used as inspiration for a new definition, by Maltsiniotis[Mal10]. An overview was given by Shulman [Shu10]. There is also a type-theoretic version by Brunerie[Bru13], on which an implementation in Agda by Altenkirch, Li and Rypacek is based [ALR14].

The idea of using the structure of univalent universes to define well-behaved higher relations, or weak ω-groupoids, is something that I initially found interesting but quickly dropped when I noticed the restrictions that we saw in Definition 9.3.2. The work by Cranch [Cra13] sheds light on this approach from a slightly different angle which makes it appear not as disappointing as I had thought it was.

In Section 9.4, we have analysed whether a type A always has a set-based representation, in the sense that there is a set A_0 and a family $h : A_0 \times A_0 \to \mathcal{U}$ which gives us the identity types of A. If we are honest, we have to admit that our formulation was already biased from the beginning. Already from the beginning, we had pretended that the only possible choice for A_0 was $\|A\|_0$. This is not unreasonable, it probably would have led to a "minimalistic" representation if it had worked; but in principle, two distinguishable elements of A_0 could still be "identified" by the family h. The proof that such a representation is only possible if A is a set goes through even if we "relax" the definition of "set-based representable" and allow any A_0 (with a map $A \to A_0$) instead of forcing A_0 to be $\|A\|_0$. For precisely this reason, our alternative definition of *reduced set-based representability* seems reasonable: if we do not allow that h contains "arrows" between points that are distinguishable in A_0, then A_0 necessarily has to be $\|A\|_0$.

Bibliography

[AAL11] Thorsten Altenkirch, Thomas Anberrée and Nuo Li. *Definable quotients in type theory*. Draft paper. 2011 (cited on pp. 6, 70).

[AB04] Steve Awodey and Andrej Bauer. *Propositions as [types]*. *Journal of Logic and Computation* 14.4, 2004, pp. 447–471 (cited on p. 8).

[ACK] Thorsten Altenkirch, Paolo Capriotti and Nicolai Kraus. *Infinite structures in type theory: problems and approaches*. Presented at TYPES'15, Tallinn, Estonia, 20 May 2015. Work in progress, working title (cited on p. 191).

[AGS12] Steve Awodey, Nicola Gambino and Kristina Sojakova. *Inductive types in homotopy type theory*. *Logic in Computer Science (LICS)*. IEEE Computer Society, 2012, pp. 95–104. ISBN: 978-0-7695-4769-5 (cited on pp. 39, 106).

[AK11] Peter Arndt and Krzysztof Kapulkin. *Homotopy-theoretic models of type theory*. *Typed Lambda Calculi and Applications (TLCA)*. Springer-Verlag, 2011, pp. 45–60. ISBN: 978-3-642-21690-9 (cited on p. 5).

[AK14a] Thorsten Altenkirch and Ambrus Kaposi. *A syntax for cubical type theory*. Talk in Oxford. 8th November 2014 (cited on p. 168).

[AK14b] Thorsten Altenkirch and Ambrus Kaposi. *A syntax for cubical type theory*. Draft paper. Available at the first-named author's webpage. August 2014 (cited on p. 168).

[AKS15] Benedikt Ahrens, Krzysztof Kapulkin and Michael Shulman. *Univalent categories and the Rezk completion*. *Mathematical Structures in Computer Science (MSCS)*, January 2015, pp. 1–30. ISSN: 1469-8072 (cited on pp. 11, 118, 153, 171, 183).

[ALR14] Thorsten Altenkirch, Nuo Li and Ondrej Rypacek. *Some constructions on ω-groupoids*. *Logical Frameworks and Meta-languages: Theory and Practice (LFMTP)*. 2014 (cited on pp. 181, 183, 191).

[Alt99] Thorsten Altenkirch. *Extensional equality in intensional type theory*. *Logic in Computer Science (LICS)*. IEEE Computer Society Press, 1999, pp. 412–420 (cited on pp. 6, 28).

[AM06] Thorsten Altenkirch and Conor McBride. *Towards observational type theory*. Draft paper. February 2006 (cited on p. 6).

[AMS07] Thorsten Altenkirch, Conor McBride and Wouter Swierstra. *Observational equality, now! Programming languages meets program verification (PLPV)*. ACM, 2007, pp. 57–68. ISBN: 978-1-59593-677-6 (cited on p. 6).

[APW13] Steve Awodey, Álvaro Pelayo and Michael A. Warren. *Voevodsky's univalence axiom in homotopy type theory. Notices of the American Mathematical Society*, 2013 (cited on p. 2).

[AR12] Thorsten Altenkirch and Ondrej Rypacek. *A syntactical approach to weak ω-groupoids. Computer Science Logic (CSL)*. 2012, pp. 16–30 (cited on p. 191).

[AW09] Steve Awodey and Michael A. Warren. *Homotopy theoretic models of identity types. Mathematical Proceedings of the Cambridge Philosophical Society* 146, January 2009, pp. 45–55. ISSN: 1469-8064 (cited on pp. 4, 23).

[Awo+11] Steve Awodey, Richard Garner, Per Martin-Löf and Vladimir Voevodsky. *Mini-workshop: the homotopy interpretation of constructive type theory. Oberwolfach Reports* 8.1, 2011, pp. 609–638 (cited on p. 6).

[Awo12] Steve Awodey. *Type theory and homotopy. Epistemology versus Ontology*. Ed. by Peter Dybjer, Sten Lindström, Erik Palmgren and Goran Sundholm. Vol. 27. Logic, Epistemology, and the Unity of Science. Springer Netherlands, 2012, pp. 183–201. ISBN: 978-94-007-4434-9 (cited on p. 2).

[Awo14] Steve Awodey. *Natural models of homotopy type theory*. June 2014 (cited on pp. 5, 126).

[Bat98] Michael Batanin. *Monoidal globular categories as a natural environment for the theory of weak n-categories. Advances in Mathematics* 136.1, 1998, pp. 39–103 (cited on p. 133).

[BCH14] Marc Bezem, Thierry Coquand and Simon Huber. *A model of type theory in cubical sets. Types for Proofs and Programs (TYPES)*. Ed. by Ralph Matthes and Aleksy Schubert. Vol. 26. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, March 2014, pp. 107–128. ISBN: 978-3-939897-72-9 (cited on pp. 5, 168, 189, 195).

[BG11] Benno van den Berg and Richard Garner. *Types are weak ω-groupoids. Proceedings of the London Mathematical Society* 102.2, 2011, pp. 370–394 (cited on pp. 4, 24, 133).

[BG12] Benno van den Berg and Richard Garner. *Topological and simplicial models of identity types. ACM Transactions on Computational Logic (TOCL)* 13.1, 2012, p. 3 (cited on p. 5).

[BL14] Guillaume Brunerie and Daniel Licata. *Cubical infinite-dimensional type theory.* Talk in Oxford. 9th November 2014 (cited on p. 168).

[Bru13] Guillaume Brunerie. *Syntactic grothendieck weak ∞-groupoids.* Unpublished note, available at `uf-ias-2012.wikispaces.com`. January 2013 (cited on p. 191).

[Cap14] Paolo Capriotti. *Higher lenses.* Blog post at homotopytypetheory.org. 29th April 2014 (cited on p. 153).

[Cap15] Paolo Capriotti. *Agda formalisations of results on eliminating out of higher truncations.* Available at `github.com/pcapriotti/agda-base/tree/trunc`. April 2015 (cited on pp. 117, 151).

[CCHM14] Cyril Cohen, Thierry Coquand, Simon Huber and Anders Mörtberg. *Cubical.* Project code available at `github.com/simhu/cubical`. 2014 (cited on p. 188).

[CDP14] Jesper Cockx, Dominique Devriese and Frank Piessens. *Pattern matching without K.* Talk at the *Types for Proofs and Programs* workshop (TYPES) in Paris. 13th May 2014 (cited on p. 14).

[CH88] Thierry Coquand and Gerard Huet. *The calculus of constructions. Inf. Comput.* 76.2-3, February 1988, pp. 95–120. ISSN: 0890-5401 (cited on p. 2).

[Con+86] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki and S. F. Smith. *Implementing mathematics with the NuPRL proof development system.* NJ: Prentice-Hall, 1986 (cited on pp. 8, 75, 117).

[Coq13] Thierry Coquand. *A property of contractible types / a remark on contractible families of types.* Unpublished note, complementing [BCH14]. 9th December 2013 (cited on p. 5).

[Coq14] Thierry Coquand. *Variation on cubical sets.* Unpublished note, 3rd version. 8th November 2014 (cited on p. 5).

[Cra13] James Cranch. *Concrete categories in homotopy type theory. ArXiv e-prints*, November 2013 (cited on pp. 178, 191).

[Dia75] Radu Diaconescu. *Axiom of choice and complementation. Proc. Amer. Math. Soc.* 51, 1975, pp. 176–178 (cited on p. 60).

[DMG01] Manfred Droste, Giraudet Michèle and Rüdiger Göbel. *All groups are outer automorphism groups of simple groups. Journal of the London Mathematical Society* 64(3). December 2001, pp. 565–575. ISSN: 1469-7750 (cited on p. 182).

[EO10] Martín Escardó and Paulo Oliva. *Searchable sets, Dubuc-Penon compactness, omniscience principles, and the drinker paradox. Computability in Europe 2010, Abstract and Handout Booklet.* 2010, pp. 168–177 (cited on p. 5).

[Esc15a] Martín Escardó. *Constructive decidability of classical continuity. Mathematical Structures in Computer Science* FirstView, February 2015, pp. 1–12. ISSN: 1469-8072 (cited on p. 5).

[Esc15b] Martín Escardó. *The intrinsic topology of a Martin-Löf universe.* Draft paper. March 2015 (cited on p. 5).

[EX15] Martín Escardó and Chuangjie Xu. *The inconsistency of a brouwerian continuity principle with the curry–howard interpretation.* Draft paper. 2015 (cited on p. 190).

[FŠ82] M. P. Fourman and A. Ščedrov. *The "world's simplest axiom of choice" fails. Manuscripta Math.* 38.3, 1982, pp. 325–332. ISSN: 0025-2611 (cited on p. 66).

[GG08] Nicola Gambino and Richard Garner. *The identity type weak factorisation system. Theor. Comput. Sci.* 409, 1 December 2008, pp. 94–109. ISSN: 0304-3975 (cited on p. 126).

[Gon+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi and Laurent Thery. *A machine-checked proof of the odd order theorem. Interactive Theorem Proving.* 2013 (cited on p. 2).

[Gon08] Georges Gonthier. *Formal proof – the four-color theorem. Notices of the American Mathematical Society* 55, 2008, pp. 1382–1393 (cited on p. 2).

[Gro83] Alexander Grothendieck. *Pursuing stacks.* Unpublished Manuscript. 1983 (cited on p. 191).

[Hagda] The HoTT and UF community. *Homotopy type theory.* Agda library, available online at `github.com/HoTT/HoTT-Agda`. Since 2012 (cited on p. 14).

[Hat01] Allen Hatcher. *Algebraic Topology.* First edition. Cambridge University Press, 3rd December 2001. ISBN: 0521795400 (cited on p. 23).

[Hcoq] The HoTT and UF community. *HoTT GitHub repository.* Since 2011 (cited on p. 45).

[Hed98] Michael Hedberg. *A coherence theorem for Martin-Löf's type theory. Journal of Functional Programming* 8.4, 1998, pp. 413–436 (cited on p. 44).

[Her15] Hugo Herbelin. *A dependently-typed construction of semi-simplicial types. Mathematical Structures in Computer Science*, March 2015, pp. 1–16. ISSN: 1469-8072 (cited on pp. 117, 167, 170, 175, 177).

[Hic96] Jason J. Hickey. *Formal objects in type theory using very dependent types. Foundations of Object Oriented Languages 3*. 1996 (cited on pp. 117, 170, 176).

[Hmail] The HoTT and UF community. *Homotopy type theory mailing list*. Google Groups. Since 2011 (cited on pp. 72, 77, 117, 179).

[Hof95] Martin Hofmann. *Extensional concepts in intensional type theory*. PhD thesis. University of Edinburgh, July 1995 (cited on pp. 6, 70, 75).

[Hof97] Martin Hofmann. *Syntax and semantics of dependent types. Semantics and Logics of Computation*. Cambridge University Press, 1997, pp. 79–130 (cited on p. 17).

[Hov07] Mark Hovey. *Model categories*. Mathematical surveys and monographs 63. American Mathematical Society, 2007. ISBN: 9780821843611 (cited on p. 130).

[How80] William A. Howard. *The formulas-as-types notion of construction. To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Ed. by J. P. Seldin and J. R. Hindley. Academic Press, 1980, pp. 479–490 (cited on pp. 2, 4, 18).

[HS96] Martin Hofmann and Thomas Streicher. *The groupoid interpretation of type theory. In Venice Festschrift*. Oxford University Press, 1996, pp. 83–111 (cited on pp. 3, 5, 6, 24).

[KECA13] Nicolai Kraus, Martín Escardó, Thierry Coquand and Thorsten Altenkirch. *Generalizations of Hedberg's theorem. Typed Lambda Calculus and Applications (TLCA)*. Ed. by Masahito Hasegawa. Vol. 7941. Lecture Notes in Computer Science. Springer-Verlag, 2013, pp. 173–188 (cited on pp. 15, 25, 43, 46, 51).

[KECA14] Nicolai Kraus, Martín Escardó, Thierry Coquand and Thorsten Altenkirch. *Notions of anonymous existence in Martin-Löf type theory*. Submitted to the special issue of TLCA'13. 2014 (cited on pp. 15, 43, 51, 69, 80).

[KLV12a] Krzysztof Kapulkin, Peter LeFanu Lumsdaine and Vladimir Voevodsky. *The simplicial model of univalent foundations. ArXiv e-prints*, November 2012 (cited on p. 5).

[KLV12b] Krzysztof Kapulkin, Peter LeFanu Lumsdaine and Vladimir Voevodsky. *Univalence in simplicial sets. ArXiv e-prints*, March 2012 (cited on p. 5).

[Kol32] Andrey Kolmogorov. *Zur Deutung der intuitionistischen Logik. Mathematische Zeitschrift 35*, 1932, pp. 58–65 (cited on p. 18).

[Kra11] Nicolai Kraus. *Higher dimensional type theory and other aspects of mathematics in computer science*. First year progress report, extended version. 2011 (cited on p. 178).

[Kra12] Nicolai Kraus. *A direct proof of Hedberg's theorem*. Blog post at homotopytypetheory.org. 30th March 2012 (cited on pp. 45, 48).

[Kra13a] Nicolai Kraus. *The hierarchy of univalent universes and a type of strictly homotopy level n*. Talk in Princeton at the UF program, notes available at `uf-ias-2012.wikispaces.com`. 8th April 2013 (cited on p. 91).

[Kra13b] Nicolai Kraus. *The truncation map $|-| : N \to \|N\|$ is nearly invertible*. Blog post at homotopytypetheory.org. 28th October 2013 (cited on pp. 86, 88, 188).

[Kra14a] Nicolai Kraus. *A haskell script to generate the type of n-truncated semi-simplicial types*. A proof of concept. 2014 (cited on pp. 13, 175).

[Kra14b] Nicolai Kraus. *The general universal property of the propositional truncation*. *ArXiv e-prints*, November 2014. To appear in the post-proceedings of *Types for Proofs and Programs* (TYPES) 2014 (cited on pp. 15, 117, 132).

[KS15] Nicolai Kraus and Christian Sattler. *Higher homotopies in a hierarchy of univalent universes*. *ACM Transactions on Computational Logic (TOCL)* 16.2, April 2015, 18:1–18:12 (cited on pp. 15, 89, 157).

[LB13] Daniel Licata and Guillaume Brunerie. *$\pi_n(S^n)$ in homotopy type theory*. Vol. 8307. LNCS. Springer, 2013, pp. 1–16 (cited on p. 189).

[Lei02] Tom Leinster. *A survey of definitions of n-category*. *Theory and applications of Categories* 10.1, 2002, pp. 1–70 (cited on p. 133).

[LF14] Daniel Licata and Eric Finster. *Eilenberg-MacLane spaces in homotopy type theory*. *Logic in Computer Science (LICS)*. ACM. 2014, pp. 66–74 (cited on p. 113).

[Li15] Nuo Li. *Quotient types in type theory*. PhD thesis. School of Computer Science, University of Nottingham, 2015 (cited on pp. 70, 75).

[Lic12] Daniel Licata. *Homotopy type theory*. Agda library, available at `github.com/dlicata335`. Since 2012 (cited on p. 14).

[Lic13] Daniel Licata. *Homotopy theory in type theory: progress report*. Blog post at homotopytypetheory.org. 20th May 2013 (cited on p. 104).

[Lum09] Peter LeFanu Lumsdaine. *Weak omega-categories from intensional type theory*. *Typed Lambda Calculi and Applications (TLCA)*. Springer-Verlag, 2009, pp. 172–187. ISBN: 978-3-642-02272-2 (cited on pp. 4, 133).

[Lum10] Peter LeFanu Lumsdaine. *Higher categories from type theories*. PhD thesis. Carnegie Mellon University, 2010 (cited on p. 24).

[Lur09] Jacob Lurie. *Higher topos theory*. Vol. 170. Annals of Mathematics Studies. Princeton, NJ: Princeton University Press, 2009. ISBN: 978-0-691-14049-0; 0-691-14049-9 (cited on p. 117).

[LW15] Peter LeFanu Lumsdaine and Michael A. Warren. *The local universes model: an overlooked coherence construction for dependent type theories. ACM Transactions on Computational Logic (TOCL)* 16.3, 2015. To appear (cited on p. 126).

[Mac71] Saunders Mac Lane. *Categories for the working mathematician*. New York: Springer-Verlag, 1971 (cited on p. 168).

[Mal10] Georges Maltsiniotis. *Grothendieck ∞-groupoids, and still another definition of ∞-categories. ArXiv e-prints*, 2010 (cited on p. 191).

[McC13] James E. McClure. *On semisimplicial sets satisfying the kan condition. Homology, Homotopy and Applications* 15.1, 2013, pp. 73–82 (cited on p. 133).

[Men90] Nax Paul Mendler. *Quotient types via coequalizers in Martin-Löf type theory. Proceedings of the Logical Frameworks Workshop*. 1990, pp. 349–361 (cited on p. 6).

[ML75] Per Martin-Löf. *An intuitionistic theory of types: predicative part. Logic Colloquium '73, Proceedings of the Logic Colloquium*. Ed. by H.E. Rose and J.C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. North-Holland, 1975, pp. 73–118 (cited on p. 2).

[ML82] Per Martin-Löf. *Constructive mathematics and computer programming. Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979*. Ed. by L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer and Klaus-Peter Podewski. Vol. 104. Studies in Logic and the Foundations of Mathematics. North-Holland, 1982, pp. 153–175 (cited on p. 2).

[ML84] Per Martin-Löf. *Intuitionistic type theory*. Vol. 1. Studies in Proof Theory. Naples: Bibliopolis, 1984 (cited on p. 2).

[ML98] Per Martin-Löf. *An intuitionistic theory of types. Twenty-five years of constructive type theory (Venice, 1995)*. Ed. by Giovanni Sambin and Jan M. Smith. Vol. 36. Oxford Logic Guides. Oxford University Press, 1998, pp. 127–172 (cited on p. 2).

[Nor07] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis. Department of Computer Science, Engineering, Chalmers University of Technology and Göteborg University, 2007 (cited on pp. 2, 13).

[Oli14] Matt Oliveri. *A formalized interpreter*. Blog post at homotopytypetheory.org. 19th August 2014 (cited on pp. 117, 170).

[Pal12] Erik Palmgren. *Proof-relevance of families of setoids and identity in type theory. Arch. Math. Log.* 51.1-2, 2012, pp. 35–47 (cited on p. 48).

[PM93] Christine Paulin-Mohring. *Inductive definitions in the system Coq - rules and properties. Typed Lambda Calculi and Applications (TLCA)*. Ed. by Marc Bezem and Jan Friso Groote. Lecture Notes in Computer Science 664. 1993 (cited on p. 24).

[PW12] Álvaro Pelayo and Michael A. Warren. *Homotopy type theory and Voevodsky's univalent foundations. CoRR* abs/1210.5658, 2012 (cited on pp. 2, 4, 18).

[Qia03] Guohua Qian. *On automorphism groups of some finite groups. Science in China Series A: Mathematics* 46.4, 2003, pp. 450–458. ISSN: 1006-9283 (cited on p. 182).

[RS14] Egbert Rijke and Bas Spitters. *Sets in homotopy type theory. MSCS, special issue: From type theory and homotopy theory to univalent foundations*, 2014 (cited on p. 43).

[RS71] Colin Patrick Rourke and Brian Joseph Sanderson. Δ*-sets I: homotopy theory. The Quarterly Journal of Mathematics* 22.3, 1971, pp. 321–338 (cited on p. 133).

[Rus03] Bertrand Russell. *The principles of mathematics*. First edition. Cambridge University Press, 1903 (cited on pp. 4, 18).

[Sat15] Christian Sattler. *On the complexities of polymorphic stream equation systems, isomorphism of finitary inductive types, and higher homotopies in univalent universes*. PhD thesis. School of Computer Science, University of Nottingham, December 2015 (cited on p. 91).

[Sco72] Dana Scott. *Continuous lattices. Toposes, Algebraic Geometry, and Logic*. Ed. by F. W. Lawvere. Lecture Notes in Mathematics 274. Springer-Verlag, 1972, pp. 97–136 (cited on p. 4).

[Seg68] Graeme Segal. *Classifying spaces and spectral sequences. Publications Mathématiques de l'Institut des Hautes Études Scientifiques* 34.1, 1968, pp. 105–112. ISSN: 0073-8301 (cited on p. 135).

[Shu10] Michael Shulman. *Grothendieck-Maltsiniotis* ∞*-categories*. Blog post at The n-Category Café. 15th September 2010 (cited on p. 191).

[Shu11] Michael Shulman. *An interval type implies function extensionality*. Blog post at homotopytypetheory.org. 4th April 2011 (cited on p. 83).

[Shu14] Michael Shulman. *Homotopy type theory should eat itself (but so far, it's too big to swallow)*. Blog post at homotopytypetheory.org. 3rd March 2014 (cited on pp. 117, 167, 168, 170, 175).

[Shu15] Michael Shulman. *Univalence for inverse diagrams and homotopy canonicity. Mathematical Structures in Computer Science*, January 2015, pp. 1–75. ISSN: 1469-8072 (cited on pp. 10, 116, 118, 124–128, 133, 135).

[Soj15] Kristina Sojakova. *Higher inductive types as homotopy-initial algebras. Principles of Programming Languages (POPL)*. ACM, 2015, pp. 31–42. ISBN: 978-1-4503-3300-9 (cited on pp. 39, 106).

[Str11] Thomas Streicher. *A model of type theory in simplicial sets – a brief introduction to Voevodsky's homotopy type theory*. Unpublished note. September 2011 (cited on p. 5).

[Str87] Ross Street. *The algebra of oriented simplexes. Journal of Pure and Applied Algebra* 49.3, 1987, pp. 283–335. ISSN: 0022-4049 (cited on pp. 124, 132).

[Uni13] The Univalent Foundations Program. *Homotopy type theory: univalent foundations of mathematics*. First edition. Institute for Advanced Study: homotopytypetheory.org/book, 2013 (cited on pp. 1, 6, 7, 9, 10, 14, 17, 20, 22, 24–41, 43, 45, 46, 49, 60, 61, 63, 71, 72, 75, 77, 79–81, 87, 92, 93, 99, 102, 103, 105, 109, 110, 112, 113, 119, 148, 153–156, 158, 159, 171, 174, 186, 187, 190).

[Vic01] Steven Vickers. *Strongly algebraic = SFP (topically). Mathematical Structures in Computer Science* 11, 2001, pp. 717–742. ISSN: 0960-1295 (cited on p. 4).

[Vic05] Steven Vickers. *Some constructive roads to Tychonoff. From Sets and Types to Topology and Analysis: Towards Practicable Foundations for Constructive Mathematics*. Ed. by Laura Crosilla and Peter Schuster. Oxford Logic Guides 48. Oxford University Press, 2005, pp. 223–238. ISBN: 0-19-856651-4 (cited on p. 4).

[Vic96] Steven Vickers. *Topology via logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 13th September 1996. ISBN: 0521576512 (cited on p. 4).

[Vic99] Steven Vickers. *Topology via constructive logic. Logic, Language and Computation Vol II*. Ed. by Moss, Ginzburg and de Rijke. CSLI Publications, 1999, pp. 336–345. ISBN: 1575861801, 157586181X (cited on p. 4).

[Voe06] Vladimir Voevodsky. *A very short note on homotopy λ-calculus*. Unpublished note. September 2006 (cited on p. 4).

[Voe10a] Vladimir Voevodsky. *Univalent foundations project*. A modified version of an NSF grant application. 1st October 2010 (cited on pp. 1, 4, 5, 7, 23).

[Voe10b] Vladimir Voevodsky. *Foundations*. Coq Library based on Univalent Foundations, available at the author's institutional webpage. Since 2010 (cited on pp. 7, 38, 75, 180).

[Voe12] Vladimir Voevodsky. *Semi-simplicial types*. Formalisation attempt in Coq, available at `uf-ias-2012.wikispaces.com`. 2012 (cited on pp. 170, 175).

[Voe13a] Vladimir Voevodsky. *A simple type system with two identity types*. Unpublished note. 2013 (cited on pp. 170, 191).

[Voe13b] Vladimir Voevodsky. *Experimental library of univalent formalization of mathematics*. Available at `arxiv.org/abs/1401.0053`. December 2013 (cited on pp. 7, 38, 75, 180).

[War08] Michael A. Warren. *Homotopy theoretic aspects of constructive type theory*. PhD thesis. Carnegie Mellon University, 2008 (cited on p. 4).

[War11] Michael A. Warren. *The strict ω-groupoid interpretation of type theory. Models, Logics, and Higher-dimensional Categories: A Tribute to the Work of Mihály Makkai* 53, 2011, pp. 291–340 (cited on p. 4).

[XE13] Chuangjie Xu and Martín Escardó. *A constructive model of uniform continuity. Typed Lambda Calculi and Applications (TLCA)*. Ed. by Masahito Hasegawa. Vol. 7941. Lecture Notes in Computer Science. Springer-Verlag, 2013, pp. 236–249. ISBN: 978-3-642-38945-0 (cited on p. 5).