KU LEUVEN

FACULTY OF SCIENCE

Models of univalence in cubical sets

An introduction to the proof of univalence and a review of recent literature

Supervisor: Prof. Dr. D. Devriese
Vrije Universiteit Brussel,
Department of Computer Science,
Software Languages Lab

Co-supervisor: Prof. Dr. W. Castryck
Katholieke Universiteit Leuven,
Department of Mathematics,
Algebra Section

Mentor: A. Nuyts
Katholieke Universiteit Leuven,
Department of Computer Science,
IMEC-DistriNet

Reader: Prof. Dr. F. Piessens
Katholieke Universiteit Leuven,
Department of Computer Science,
IMEC-DistriNet

Willem VANHULLE

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Mathematics

Academic year 2018-2019

Preface

This topic for my master thesis came up when reading about homotopy type theory because I wanted to know if the implementation of the axiom of univalence in cubical sets can make the life of a mathematician easier or make writing, reading and exchanging proofs in obscure but still interesting mathematical domains more efficient.

This text was written with the help and support of many people. First of all, I would like to thank everyone who read drafts of this text thoroughly and gave very useful feedback on intermediate presentations: Andreas, Dominique, Frank and Wouter. These friendly people have a lot of experience and made me aware of the actual state of my drafts. I also had a lot of curious friends such as Jonathan, Alexander, Emily, Zhiyu, Michael, Dieter, Ben, Edward, Zhiquan. They listened to me when I tried to explain to them what I was reading or writing. Sometimes they asked questions that made me understand what I did not understand.

Leuven, May 2019

Summary

The univalence axiom states that equivalent spaces can be identified and has many interesting consequences such as the ability to do mathematics up to isomorphism as claimed by [VAC$^+$13], a statement that will be verified in section 3.5.1. If an axiom is consistent and simple enough, its addition to the theory is a compromise mathematicians are willing to make in exchange for beautiful results that depend on it. The consistency of the univalence axiom relative to ZFC was proven by giving a model in simplicial sets [KL12]. This was a first step in the recognition of the univalence axiom among mathematicians. However, this model in simplicial sets made use of the axiom of choice which made it unconstructive and unsuitable for a constructive implementation, see section 2.3.

Cubical type theory is an intuitionistic type theory that can model the univalence axiom, not as an axiom, but as a valid and provable theorem. This type theory will be explained in detail in chapter 3. Its proof in cubical theory has only recently been completed in [CCHM16]. Meanhwile, several updated proofs have been constructed in [SHC$^+$18, MV19] which will be discussed in section 3.4. By the constructiveness of cubical type theory, a proof gives an explicit map between given equalities. The map can be used to rewrite proofs of theorems in homotopy theory in the more basic constructive language of cubical sets. Understanding constructive proofs of univalence can also help to study or understand applications or consequences of the univalence axiom which will be surveyed in section 3.5. This text is mainly a literature review which means that as much recent literature will be covered and put into context as possible. In section 3.5.4 and section 3.6.3 open problems in the field will be discussed.

Glossary

$\Gamma \vdash \ldots, \Delta \vdash \ldots$ Contexts of judgments

$a : A$ A term a (lower case) of type A (capitalized)

$\mathtt{Glue}$ Font for custom types

$A = B$ Equality type of types

$X \equiv Y$ Equality by definition

$A \simeq B$ Equivalence type of types

$\mathcal{U}$ A universe of types

Set Examples of categories

$\mathcal{C}, \mathcal{D}$ Categories

$I, J, K \in \mathcal{C}$ Objects in a category $\mathcal{C}$

$\mathcal{F}, \mathcal{G}$ Functors of categories

i, j, k, x, y, z Dimension variables

r, s, t Constants in unit interval $\mathbb{I}$

(i/r) A substitution of variable i by r

Contents

Chapter 1

Introduction

1.1 Foundations of mathematics

Around the beginning of the twentieth century, mathematicians were working on foundations of mathematics and formalized the building blocks of mathematics: sets, propositions and proofs. Important examples of developments are the first inductive definition of the natural numbers which appeared in [Pea79], the concept of types invented around the time of [Rus03] and intuitionistic logic [Hey30]. Because these developments, especially intuitionism and type theory are important for the rest of this text and quite different from mathematics, this chapter will devote some time to explaining both.

1.1.1 Constructivism

The foundational work around the end of the 19th century was very different from the mathematics in previous centuries. Mathematicians such as Cantor developped the concept of countability and ordinals but such results were very abstract and vague, even to other mathematicians. The terms *constructivism* and *intuitionism* stand for the philosophical view of the time in which mathematicians questioned weird abstract foundational work and was made popular by [Bro05]. According to [Bro05], mathematics should be the result of mental human activity rather than objective discovery. This philosophy was formalized into a formal system based on intuitionism, called intuitionistic (or constructive) logic [Hey30]. The difference between intuitionistic and traditional logic, is the lack of the following propositions which are extra (unneccesarry) assumptions or axioms according to intuitionistic logic:

- The *principle of excluded middle* states that for every proposition P, $P \vee \neg P$. It means that P holds or does not hold and is also called *decidability* of P. Decidability can hold for a large portion of mathematics and does not need to be taken as an axiom, see for example the decidability results in [Tar51].

- The *axiom of choice* states the existence of a choice function f on collections of sets X:
$$f : X \to \cup X, \quad \forall A \in X : f(A) \in A$$
and can be seen as a generalization of the principle of excluded middle. This axiom can lead to *abstract nonsense* but weaker versions are accepted in intuitionistic logic, see [VAC+13], section 3.8.

- Other axioms that involve others ways of implicit choice on arbitrary objects are not accepted unless a constructive motivation or model is given, see chapter 3 for a constructive model of the univalence axiom in ax. 1.3.2.

Example 1.1.1. *One example of a theorem that traditionally is given using the principle of excluded middle, is mentioned in most introduction to constructive mathematics and also in the introduction of [Pal14]:*

Theorem 1.1.2. *There are irrational numbers a and b such that a^b is a rational number.*

This theorem is not a valid theorem in intuitionistic logic anymore, unless a proof without the principle of excluded middle is given:

Proof. The number $a = \sqrt{2}$ is irrational by the constructive proof in [Ros84], p. 18 and $b = 2\log_2(3)$ is also constructively irrational but by computation $a^b = 3$. $\square$

This proof gives an explicit construction and can be directly verified by computation, opposed to the classical proof. It is however harder to give constructive proofs of irrationality than is suggested by [Bau09].

There are still theorems that do not have constructive proofs such as the Robertson-Symour theorem, see [BL95].

The philosophy of intuitionism is kept alive in proof assistants and type theory which are used by mathematicians to write more intuitive proofs with the help of computing power. Most proof assistants such as Coq, NuPRL, Agda support writing proofs in intuitionistic logic. On top of expressions and definitions in intuitionistic logic, a user of a proof assistant can formulate complex theorems and proofs. These formalized proofs can be used to investigate proofs of real-life theorems from mathematical domains such as algebra or topology that are hard to write down by hand such as the Odd Order Theorem:

Theorem 1.1.3. *If G is a finite group and there exists an $n \geq 0$ such that $|G| = 2n+1$, then G is solvable.*

This theorem was conjectured in 1911 and proven much later in [FT63]. It was one of the first proofs in group theory that was hundreds of pages long. The proof was formalized in the proof assistant Coq. The implementation of the formal proof ended up being at least (or only) 5 times as long, see [GAA$^+$13], section 6. The implementation of such a large proof required the implementation of new features and re-usable libraries in the proof assistant Coq. Subsequently, new developments should be easier. Proponents claim that proof assistants will become more widely used in the short term.

1.1.2 Type theory

Type theory, in general, is different from set theory. Set theory has two main layers: sets and propositions about elements of sets. In type theory, the analogues of these two layers are called *type*s and *term*s of types. The main difference between set theory and type theory is that terms are always accompanied by their type. Types can be empty, but when they have a term, they are called *inhabited*. Terms do not exist on their own. Without a type annotation, terms do not have any meaning, while in set theory it is perfectly possible to speak about a mathematical object on its own. For example, in type theory there is the type of groups denoted by `Group` and a group G would be denoted by the notation G : `Group`. Although this difference can seem very superficial, it implies that set membership is not a logical proposition anymore but becomes a judgment, see section 1.2.1.

Designers of formal systems and proof assistants based on type theory are confronted with questions about the properties of their systems. For example, certain expressions written in the system can or can not normalize or other expressions can be invalid in the system but not according to a mathematician using the system. These are the topics that this text will try to cover although some applications can be mentioned in section 3.5. An in-depth overview of the history of type theory can be found in [Coq13b] or [Con11] and [Con15].

1.2 Intuitionistic type theory

Now that some useful applications of type theory have been given, this section will explain in more detail what intuitionistic type theory [Ml75] is about.

1.2.1 Judgments and contexts

A *judgment* is a statement in (intuitionistic) type theory about types and terms. Judgments can be philosophically seen as a constructive act of knowledge but formally, they come in different forms:

- A (starting with upper case) is just the statement that A is a type.
- $A = B$ means that A and B are equal types, can also be seen as an identity type, see def. 1.2.6.
- $a : A$ (starting with lower case) means that a is a term of type A, also called the *membership judgment.*

Judgements are always accompanied by transformation rules that describe how several judgements can be combined into one new judgement. These rules, called *typing rules*, can be seen as rules of computation. This is an important difference with set theory because in set theory the only constructions available are not much more than sets, tuples and propositions. But intuitionistic type theory is rooted in intuitionism and constructivism (see section 1.1.1) which requires a constructive approach to computation.

Definition 1.2.1. *The following types of typing rules are required for every type:*

- *The introduction of a new type and how the type is referenced once it is introduced is denoted with a* formation rule. *These rules are important for the readability of the judgements.*
- *How terms of the type being introduced are constructed is denoted by a typing rule. For example, there are two ways to introduce a term of the type of natural numbers: the zero element or its successor. These rules for introducing terms are called* constructor*s.*

- *To compute with terms and types, other rules that describe how it is possible to apply terms and types to other terms and types. These rules are called* eliminator*s. The* computation rule*s determine how an eliminator can act on a constructor.*

This structural distinction between rules according to their constructive effect on the formal system is not really present in set theory. In set theory there is not much more than set comprehension. However, certain set-based theories such as category theory also have a more structural rule-based approach.

To make the bookkeeping of judgments and the above rules easier, *context*s are used. Contexts contain the judgments that are valid in the type theory according to the above rules but are deemed not important enough by the type-theoretical proof writer to mention explicitly. Their usage corresponds to the usage of scopes in programming languages. Traditionally a context is denoted by Γ or Δ and stands for a list

$$a_1 : A_1, ..., a_n : A_n$$

of judgments $a_i : A_i$ where a_i is a term of the type A_i. The empty context is denoted by (). In later chapters, contexts do not always take this form (see def. 3.2.2). Judgments can be added to or taken from contexts:

- Judgments $a : A$ can depend on the judgments that are present in a context which is denoted by the expression $\Gamma \vdash a : A$ and can be read as "$a : A$ can be derived from Γ". In formal type theories, all judgments are required to have a context. When a judgment does not depend on other judgments, this is denoted by $() \vdash a : A$, or in other words, $a : A$ depends on the empty context.

- A context Γ can be extended with any judgments $x : T$ if $\Gamma \vdash x : T$ is a valid expression in a process called *context extension*. Contexts can also be shortened by removing judgments and moving them towards the right. This is what happens for example in the introduction rule for the product type stating that if $\Gamma, x : A \vdash b : B$ is a valid judgment, then there is a term of the product type $\Gamma \vdash \lambda(x : A).b : \prod_{x:A} B$.

1.2.2 Informal type theory

The book [VAC+13] introduced a difference between two kinds of type theory: informal and formal. Informal type theory was introduced to make it easier for people accustomed to classical set-based mathematics to understand type theory. The usage of contexts in informal type theory is implicit and natural language is alternated with new judgements. This practice is clearly illustrated in the the first chapters of [VAC+13].

In *formal type theory*, for example as defined in [VAC+13], A.2 or in implementations in proofs assistants, context dependency is explicit because natural language is considered ambiguous. This was also one of the reasons type theory was introduced in the first place: to eliminate mathematical paradoxes formulated in natural language.

However, in all proof assistants there are ways to leave parts of the context implicit and force the proof assistant infer the missing parts of the context. This brings back some of the benefits of informal type theory and is called *implicit arguments* but can return the problems of ambiguity as in informal type theory, see for example the difficulties encountered in section 3.5.1.

1.2.3 Natural deduction

Throughout this text, formal type theory will be studied as much as possible. Because context dependency is explicit in formal type theory, it is possible to write very rigorous and explicit sequences of derivations in proofs that apply the typing rules of def. 1.2.1. In this text and many other texts, there is a special notation for derivations that is inspired by natural deduction. The method of *natural deduction* was originally a formal approach to logic that helped to state the rules of the proving game very succinctly. It was a reaction to the less informative Hilbert-style proofs and was defined for the first time in [Gen35]. Proofs in the style of natural deduction visually resemble how proofs are "naturally" constructed in the head of a mathematician, see ex. 1.2.2 and for an introduction to this style [GT90], chapter 2.

Understanding this notation is important for understanding the literature on the subject. Most texts on type theory introduce the typing rules at the beginning with the help of natural deduction. The notation allows for a very succint description and introduction to the theory. For example, the main reference text on cubical type theory introduces type theory on one page in fig. 1 of [CCHM16].

1.2.4 Propositions as types

Functions, which are terms of function types, are the most fundamental types and building blocks of type theory.

Example 1.2.2. *The typing rule for elimination or computation of function types, see def. 1.2.1, can be written with natural deduction, see section 1.2.3, as follows:*

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash b : B}$$

This typing rule does not make use of set-theoretic concepts such as relations but just reflects how functions are used or computed with.

Function types were historically the first types to have an interpretation in intuitionistic logic [Hey30] which was discovered in [Cur34]. More precisely, implications in intuitionistic logic correspond to terms of function types in type theory. This means that a constructive proof of the proposition $A \Rightarrow B$ corresponds to giving a function in type theory which is a term f of the function type $A \to B$, written with a membership judgment as $f : A \to B$.

Types that can be constructed with the function type are called *simple types* but other types also have parallels with logic:

Example 1.2.3. *The* sum type *of two types A and B, denoted by $A + B$, corresponds to the "or" operation, formally denoted by the symbol $\vee$ from intuitionistic logic. In proofs that make use of the the logical proposition $A \vee B$, there are two branches: one branch that proves the case were A is assumed and the other branch proves the case were B is assumed. The terms of a sum type $A + B$ are also constructed in two ways: either such a term is $\texttt{left}\ a : A + B$ for $a : A$ or $\texttt{right}\ b : A + B$ for $b : B$. A function $A + B \to C$ for C some type is defined by case analysis, stating a result for values of both injections.*

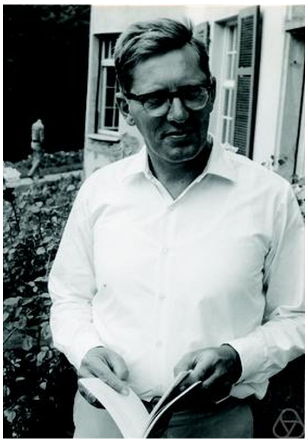

Figure 1.1: Nicolaas Govert de Bruijn (1918-2012) was a Dutch number theorist and the creator of one of the first proof assistants. Less-known, he started to treat proofs in full mathematical logic as objects with computational content. Picture taken in Oberwolfach and found on [Jac17].

This interpretation can be generalized somewhat to interpreting constructive propositions as types and their proofs to terms of types. But it remains just an approximate interpretation between two completely foundational theories, not an isomorphism or bijection. This interpretation can however be helpful to formalize constructive mathematics and was applied in the first proof assistant Automath [BG70] (see fig. 1.1). Multiple people worked (independently) on making this interpretation precise and it is often named after (some of) these people. Besides being called an interpretation, it is also called the propositions-as-types relation or *Curry-Howard correspondence*. There are several long introductions available to the history of this correspondence such as [Bib19].

Consequences of the correspondence

Certain versions of type theory do have good computational properties such as being strongly normalizing: computations terminate (see section 3.6.3). The main requirement for being strongly normalizing is not having axioms such as excluded middle and other versions of the axiom of choice. In other words, the type theory can only contain type theoretic interpretations of concepts of constructive mathematics, see section 1.1.1. The Curry-Howard correspondence then implies for such intuionistic type theories that terms double as constructive and computable proofs. Which means that these terms can be converted into programs, a process called *program extraction* and implemented in many proof assistants. Examples of such implementations are:

- The proof assistant Agda is a functional programming language based on intuitionistic type theory (see def. 1.2.10) in which programs double as mathematical proofs [ACD$^+$19]. Program extraction is possible for well-behaved parts of the theory but it is not the main purpose, see for an example of where proof extraction fails section 3.5.3.

- Idris is a programming language related to Agda and compiles to imperative languages, see [BCdMH18]. This programming language will not be further investi-

$$
\begin{aligned}
[\![P \Rightarrow Q]\!] &\equiv [\![P]\!] \to [\![Q]\!] \\
[\![P \wedge Q]\!] &\equiv [\![P]\!] \times [\![Q]\!] \\
[\![P \vee Q]\!] &\equiv [\![P]\!] + [\![Q]\!] \\
[\![\forall x : A.P]\!] &\equiv \prod_{x:A} [\![P(x)]\!] \\
[\![\exists x : A.P]\!] &\equiv \sum_{x:A} [\![P(x)]\!] \\
[\![\bot]\!] &\equiv 0 \text{ (empty type)} \\
[\![\top]\!] &\equiv 1 \text{ (singleton type)}
\end{aligned}
$$

Figure 1.2: The Curry-Howard correspondence can be illustrated by defining a map Ψ that takes an intuitionistic proposition P and extracts its evidence $[\![P]\!]$ as a type. Ψ can be defined for the full first-order fragment of constructive mathematics, based on [Alt17], p.3.

gated.

- The proof assistent Coq [ADH$^+$19] has program extraction as one of its main applications, see for example [PCZF$^+$18]. This particular extraction mechanism does have some limitations, for example in the context of the Theorem of Algebra [CFL05].

1.2.5 Universes

Type theory is an alternative foundational theory for mathematics based on intuitionism. Set theory does have self-referencing paradoxes such as the set of all sets that do not contain itself

$$R = \{x \mid x \notin x\}, \quad R \in R \Leftrightarrow R \notin R.$$

This paradox is often called the *Russel paradox* but was discovered first by Zermelo in 1899 and later eliminated by a hierarchy of classes of sets. Type theory has a similar self-referencing paradoxes, called the Burali-Forti or Girard paradox but is harder to state.

This paradox is solved in type theory by

- Reject the set comprehension principle which allows to form weird sets and replace it by types and typing rules as in def. 1.2.1.

- Forcing the existence of an upside-down tower of types such that each type consistently belongs in that tower.

The levels of the tower are *indexed* by ordinals or natural numbers For example, if the indexing set is the natural numbers, the levels are denoted by $\mathcal{U}_i$ for $i \in \mathbb{N}$. The levels are called *universe*s and are cumulative, $T : \mathcal{U}_i : \mathcal{U}_{i+1} \Rightarrow T : \mathcal{U}_{i+1}$.

In informal type theory, this tower is implicitly used and it is assumed that every type T belongs to some universe, denoted by $\mathcal{U}$ or $Type$. The universes in the tower

```
_ ≃ _ : ∀ (A : Set ℓ) (B : Set ℓ') → Set (ℓ-max ℓ ℓ')
A ≃ B = Σ[ f ∈ (A → B) ] (isEquiv f)
```

Figure 1.3: Equivalences will be introduced in def. 1.3.1. The definition of equivalences in [MV19] is defined by explicitly referencing the universe levels of the types A and B, respectively denoted by ℓ and ℓ'. The universe level of the type of equivalences is the maximum `ℓ-max ℓ ℓ'` to prevent paradoxes arising from using `A ≃ B` in the same context as other expressions using `A` or `B`.

can be considered as types, with types as terms. An important consequence of this is that every type is also a term of some universe.

In formal type theory implementations such as Agda, the tower of universe levels can be manipulated explicitly. This allows for formalizations of complex foundational mathematics. The explict referencing of universe levels is also necessary because the assignment of universe levels is a non-trivial task that cannot be automated with implicit arguments (see section 1.2.2) without introducing inconsistencies, see for example fig. 1.3.

1.2.6 Dependent types

In traditional logic, propositions state properties about elements of sets. Such a property could for example state: given a group, there is a group element g with infinite order. One could say that this property depends on the presence of an element g, and in a sense it is a *dependent property* depending on a group element g. Properties can be translated to type theory through the Curry-Howard correspondence as types such that types, being interpreted as propositions, can depend on terms of other types. In this case, they are called *dependent types*. The property about group element becomes for example a type called `hasC`$_\infty$ with terms that correspond to proofs of the property.

Definition 1.2.4 (Dependent types)**.** *Let A and B be types belonging to the hierarchy of universes in a consistent way, this means that their presence does not introduce any paradoxes. Formally speaking, it is enough to require that A and B belong to the same universe type: $A, B : \mathcal{U}$. Next to the simple types, the following types are basic building blocks for intuitionistic type theory:*

- *Given a type B that depends on the choice of a term $a : A$, the* dependent product *is the type denoted by*
$$\prod_{a:A} B(a)$$
which has as terms the generalized functions $f : (a : A) \to B(a)$. The dependent product can be viewed as a generalization of the function type and the typing rules (see def. 1.2.1) for the dependent product are formally stated in [VAC$^+$13], A.2.4.

- *The dual type of the dependent product over a type A and a family of types B depending on terms $a : A$ is the* dependent sum *which is denoted by $\sum A\ B$ or*
$$\sum_{a:A} B(a).$$

```
notZero :  ℕ → Set
notZero n = Σ ℕ (λ m → (n ≡ (suc m)))

ℕ₀ : Set _
ℕ₀ = Σ ℕ (λ n → notZero n)
```

Figure 1.4: The type of non-zero natural numbers $\mathbb{N}_0$ can be defined in Agda using a dependent sum type indexed by the natural numbers. Terms of the type ℕ₀ are tuples of a natural number and a proof that it is non-zero in the form of a dependent function.

This type has as terms tuples (a, b) where $a : A$ is a term and $b : B(a)$ is the value of a generalized function applied to this term. The formal typing rules for the dependent sum are in [VAC+13], A.2.5. When the second term b does not depend on the first term a, the dependent sum becomes a finite product which is denoted by $A \times B$. A funite sum type has as terms the tuples of the form (a, b) where $a \in A$ and $b : B$.

When the indexing type A only has two terms, both the dependent product and the dependent sum are equivalent with the finite product type $B \times B$. Using these two dependent types it is possible to build other dependent types. See fig. 1.4 for an example of a custom dependent types used in practice. When working in set-theoretic models, the definition of a dependent type can be stated very precise in terms of sets, see def. 2.4.8.

1.2.7 Equality

Equality in type theory is completely different from set theory and called the intensional identity type. Its definition is such that type theory still good computational properties such as the *normalizability* of terms and types, see section 3.6.3. This implies it is suitable for implementation of proof assistants and this contributed to its popularity.

In traditional mathematics, an equality between two sets is an equivalence relation: a transitive, symmetric and reflexive relation. When an equivalence relation is not explicitly given, it assumed that the equivalence is one of the following:

- Equality by definition: can be seen as an equality up to substitution of definitions. It denoted in this text by $\equiv$ and by $=$ in most programming languages.
- Equality by isomorphism: between objects of a category.

In the absence of any of such intepretations, it also possible to define equality based on whether properties are the same or different between objects. This way of identifying objects is called *extensional* because it is based on the behavior of "black-box" objects which can not be well-behaved. This equality is called *judgmental equality*. Judgmental equality is sometimes also called *propositional equality* to emphasize that the properties of equal objects (at least according to this equality) have the same properties.

Example 1.2.5 (Leibniz's function extensionality axiom)**.** *Two arbitrary sect functions $f, g : X \to Y$ can be identified whether they take the same values:*

$$f = g \Leftrightarrow f(x) = g(x), \forall x$$

Figure 1.5: Per-Martin Löf (1942-) is a Swedish logician and statistician. He was the first to give a good definition of equality in type theory with his intensional identity type [Ml75]. Picture taken from [Eur13].

This is an example of a propositional equality between two different objects. This principle is in type theory also called function extensionality axiom *because it does not hold in standard type theory. The function extensionality axiom does hold however in certain extensions such as cubical type theory or univalent type theory, see [VAC⁺13] for a proof. A generalization of this axoim is called* indiscernability of identicals.

The *intensional identity type* is not defined as a relation using set-theoretic concepts such as tuples, opposed to the traditional equivalence relations of mathematics. The intensional identity type is a type with its own typing rules which will be stated explicitly in def. 1.2.6 and was only fairly recently added to type theory. Its definition, see def. 1.2.6, was given inductively, similar to the inductive definition of the natural numbers (see fig. 1.5).

Definition 1.2.6 (Formation rule)**.** *Given a type X and terms $a, b : X$, the* intensional identity type *between a and b is written as $a = b$, $a =_X b$ (or sometimes $\mathtt{Id}_X(a, b)$). The intensional identity type satisfies props. 1.2.7 and 1.2.8.*

When $a =_X b$, the terms a and b satisfy the same properties, so it is possible to treat the intensional identity type as the propositional equality from mathematics which explains the notation "=".

Property 1.2.7 (Introduction rule)**.** *There is only one constructor called* reflexitivity *or in short* `refl`*. Given a term $a : X$ it returns the term $\mathtt{refl}(a)$ of the intensional identity type $a = a$.*

The rule prop. 1.2.7 makes the intensional identity type reflexive by construction.

The definition of the eliminator for the intensional identity type is one of the most complex expressions in type theory because it looks so self-explanatory or simple but is not. This is because the proof of equality is now a term of the intensional identity type and there can be multiple terms (or equalities).

Property 1.2.8 (Elimination rule)**.** *The eliminator is also called* path induction *or J: given a dependent product type, also called a family*

$$C : \prod_{x,y:A} (x =_A y) \to \mathcal{U}$$

representing a predicate depending on terms (and proofs of identity), a function

$$c : \prod_{x:A} C(x, x, \mathtt{refl}_x)$$

that can be considered as the base step in an inductive construction, there is a function

$$f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$$

such that

$$f(x, x, \mathtt{refl}_x) \equiv c(x).$$

It tells that that to prove a property for all terms x, y and equalities $p : x = y$ between them, it suffices to consider all the cases where x is definitionally equal to y and where the term of the intensional identity type under consideration is $\mathtt{refl}_x : x = x$.

Using the definition of the eliminator, it is possible to prove the remaining properties of the usual identity [VAC$^+$13], chapter 2.

Lemma 1.2.9. *For every type A and every $x, y, z : A$:*

- *The intensional identity type is symmetric. There is a function $(x = y) \to (y = x)$ denoted by $p \mapsto p^{-1}$, such that $\mathtt{refl}_x \equiv \mathtt{refl}_x$ for each $x : A$. The term p^{-1} is called the inverse of p.*

- *The intensional identity type is also transitive. This means that there is a function $(x = y) \to (y = z) \to (x = z)$ written $p \mapsto q \mapsto p \star q$ such that $\mathtt{refl}_x \star \mathtt{refl}_x \equiv \mathtt{refl}_x$. The expression $p \star q$ is called the* concatenation *of p and q.*

- *The intensional identity type supports* indiscernability of identicals*, see ex. 1.2.5 in the following way: if P is a type family over A or in other words, there is a dependent function $P : A \to \mathcal{U}$ and suppose there is a term $p : x =_A y$, then there is a function, called the transport $p_\star : P(x) \to P(y)$. When the family P is important, $p_\star$ is sometimes also denoted by $\mathrm{transport}^P(p, -) : P(x) \to P(y)$.*

These properties are not the defining properties of the intensional identity type but consequences of prop. 1.2.8. This is what makes the intensional identity type special compared to the traditional notion of equality, defined with relations over sets.

As mentioned earlier, the introduction of the intensional identity type was an important step in the establishment of (Per-Martin Löf) type theory. Having defined the intensional identity type in def. 1.2.6, it is possible to summarize the previous sections in a definition. Because there are many definitions of type theory, it is also a good occasion to fix on a particular definition.

Definition 1.2.10. *A theory about types is called a* type theory *if it includes typing rules as in def. 1.2.1 for the following types:*

- *The empty and unit type, the type of booleans.*
- *The inductive type* $\mathbb{N}$ *of natural numbers.*
- *The function, dependent product, sum types and their derivatives introduced in def. 1.2.4.*
- *The intensional identity type from def. 1.2.6.*
- *The universes from section 1.2.5.*

This definition is however very simple and is often extended with more complicated types such higher inductive types, see for an introduction to type theory [Pal14] or for higher inductive types [VAC+13]. In this text, only aspects of type theory relevant to (applications of) the proof the univalence theorem will be mentioned. The type theory def. 1.2.10 is also called MLTT and is very close to the implementation of proof assistant Agda [ACD+19] and variants of MLTT such as CIC, used in proof assistant Coq [ADH+19]. For now, for understanding the intensional identity type and the univalence axiom, the differences are however not that important. There are for example formalizations of the univalence axiom and applications of it in both assistants. For simplicity, this text not cover the differences between MLTT, CIC and type theory as in def. 1.2.10 and all code examples and implementations will be in Agda.

Implications of equality as a type

With equality having been defined an implemented, it is possible to translate the definitions of mathematical objects that use equality relations between elements of sets to the language of type theory. An example is the is the type theoretic definition of a group. One could try to define a group as a dependent sum. Take G to be the base type, e the neutral element, i the inversion and m the multiplication map. A first try at a definition of a group would then be $\sum_{G,e,i} m : (G \times G \to G)$, but this definition is not yet complete. The maps i and m have to satisfy associativity and other properties. These properties have to be encoded as types or terms. Let's say there is a proof that these properties hold, called α, then the first definition of a group can be extended to contain this proof. However, the following quesions turns up: if two groups G and G' have different multiplication maps, they are definitely not equal. But what if they have different proofs of associativity, α and α'? This can happen because the identity between terms of G can have more than one term. Are they still equal? They are not, at least according to the way the base type G is chosen. By the definition of the intensional identity type on G it was possible to have different terms of the intensional identity type and accordingly different proofs of associativity α and α'.

Of course this an undesirable consequence of the generality of the definition of the intensional identity type. In practice, it is undesirable to have more than one term or proof of equality between certain terms representing group elements. The difference in proofs is irrelevant to the structure and properties of the group. This motivates the following definitions:

Definition 1.2.11.

- *An* h-prop X *is a type such that any two terms* $x, y : X$ *are equal and can be formally stated as*

$$\mathtt{isProp}(X) \equiv \prod_{x,y:X} (x = y).$$

- *When the intensional identity type between any two terms of a type X is an h-prop, the type is called an* h-set *which is formally denoted by*

$$\mathtt{isSet}(X) \equiv \prod_{x,y:A} \prod_{p,q:(x=y)} (p = q).$$

It can be proven that $isSet(X)$ *is always a proposition.*

Example 1.2.12. *All types that have a decidable equality are h-sets, the types are sometimes called* discrete types*. More specifically, the natural numbers* $\mathbb{N}$ *are an h-set. A proof can be found in the library [MV19].*

The terminology of these concepts refers to the fact that they serve the same purpose as propositions and sets in intuitionistic logic. It can be proven that they have the same properties. In type theory the prefix is omitted, for example the type of all types that are h-props is denoted by `Prop` and the type of all types that are h-sets is denoted by `Set`, but for clarity, this text will always try to use the "h" prefix.

In the previous example, the type G can be chosen to be an h-set, that is $G : \mathtt{Set}$. Then the possibility of having multiple proofs of associativity is eliminated because the intensional identity type is always an h-prop. The definition of groups can be changed such that it only allows for G to be an h-set. This gives an intuitive type-theoretical alternative to the traditional mathematical definition of groups. This means that all existing constructive proofs can be studied again in type theory. Although this can seem like a stupid idea to do everything again, it has proven to be useful in the context of big theorems in algebra that needed a lot of so-called proof-engineering [GAA+13].

1.3 The univalence axiom

The univalence axiom states that the type of equivalences between two types is equivalent with the intensional identity type between those types. Its addition to type theory gives a new theory, called *univalent type theory*, see def. 1.3.3, a new domain in mathematics that was considered very promising and still is. See for example [Shu18]. In this domain, concepts from topology, category theory and type theory are combined to study new ways of writing proofs and discover new results because of its topological interpretations, see section 2.2. The definition of the univalence axiom is based on the concept of equivalences. It is proven in [VAC+13], section 4.5, that there are at least three different but equivalent definitions of equivalence. The definition of equivalences used in this section and in [PW14] corresponds with the definition `isContr` in [VAC+13]. In applications (see for examples section 3.5) it is often easier to use `biinv` which views equivalences as generalized homotopies.

Definition 1.3.1. *Let* $X, Y : \mathcal{U}$ *for some universe* $\mathcal{U}$ *be types and* $f : X \to Y$ *an ordinary function.*

Figure 1.6: Vladimir Voevodsky (1966-2017) was a Russian algebraist who found new ways to apply topology to algebraic geometry. When working on the simplicial model for a type theory that was inspired by algebraic topology, [Voe10] introduced the univalence axiom under the name of equivalence axiom. According to [Voe14], the term *univalence* comes from a Russian translation of [BV06] where the term "faithful functor" is translated as "univalent functor". He said about his terminology himself: "Indeed these foundations seem to be faithful to the way in which I think about mathematical objects in my head." Unfortunately, he suffered from hallucinations [Bae17] and died because of an aorta-aneurism [Reh17]. Picture taking in his office at Princeton and found on [Gra17].

- *For each $y : Y$, the* homotopy fiber *of f over y is defined as*
$$f^{-1}(y) \equiv \sum_{x:X} (f(x) =_Y y).$$
- *A type X is* contractible*, if there is an $y : X$, called the center of contraction, such that the type $x = y$ is inhabited, there is a term $contr_x$, for all $x : X$. In other words $isContr(X) \equiv \sum_{y:X} \prod_{x:X}(x = y)$.*
- *The function f is called an* equivalence *between X and Y if there exists a term of the type*
$$equiv(f) \equiv \prod_{y:Y} isContr(f^{-1}(y)).$$
The type of all equivalences between the types X and Y is given by
$$X \simeq Y \equiv \sum_{f:X \to Y} equiv(f).$$

Given two types $X, Y : \mathcal{U}$ for some universe $\mathcal{U}$ and a term $p : (X = Y)$ of the intensional identity type, it is quite straightforward to prove with the transport that there is a unique equivalence between the types $e : (X \simeq Y)$. This gives a function $\Phi_{X,Y}$ from the intensional identity type to the type of all equivalences.

Axiom 1.3.2 (Univalence axiom)**.** *Given types $X, Y : \mathcal{U}$ for some universe $\mathcal{U}$, the map $\Phi_{X,Y} : (X = Y) \to (X \simeq Y)$ is an equivalence of types.*

It can be proven that all equivalences as defined in def. 1.3.1 are propositions. The univalence axiom, once stated and accepted, has to be a proposition because the property of being an equivalence is a proposition. The univalence axiom gives ways to identify equivalent types. An algebraic topologist [Gra18b] said that it

> ... offers the hope that formalization and verification of today's mathematical knowledge can be achievable, relieving referees of articles of the tedious chore of checking the details of proofs for correctness, allowing them to focus on importance, originality, and clarity by exposition.

It was also mentioned in [Voe10] that the use of the univalence axiom in proof assistants such as Coq could be useful for the formalization of mathematics. This later resulted in an actual library [VAG+10] with code formalizing large portions of popular mathematics. The Univalent Foundations project was a joint work undertaken by mathematicians active in the fields of logic, computer science, algebra and topology around consequences of the univalence axiom that resulted in a textbook for univalent type theory, also called *homotopy type theory* [VAC+13]. The practical aspects of this project are discussed in [Bau13]. Meanwhile, there are more introductions available to univalent type theory such as the short introduction for mathematicians [Esc18], the longer introduction [Gra18b] and the implementation oriented course text [Esc19].

Definition 1.3.3 (Univalent type theory)**.** *A type theory (see def. 1.2.10) that includes the univalence axiom, defined in ax. 1.3.2, is called* univalent *or a* univalent type theory*. Because of its homotopic interpretation, see section 2.2, it is also called a* homotopy type theory*.*

Take two h-sets, or even two definitionally equal h-sets. In general, there are multiple bijections possible between these h-sets. Bijections between h-sets are the equivalences between h-sets. Simplifying the finer aspects of equivalences, the univalence axiom roughly gives a bijection between those equivalences and terms of the intensional identity type between h-sets. However, in case there are multiple bijections, multiple equalities can be possible and this unsatisfactory. An h-set should only be equal to itself in one way. Furthermore, the universe in which those h-sets are contained cannot be an h-set. Furthermore, the type of objects in a category is not an h-set, see [VAC+13], lemma 9.1.8. In general, there exist constructions of types that prove for any positive integer n that there is a nested tower of equalities which is non-trivial with depth n, see fig. 1.7.

This demonstrates that in univalent type theory certainly not all types are h-sets and there can be multiple proofs of identity. This implies that the *uniqueness of identity proofs principle* (UIP), stating that there is only one proof of identity, does not hold. So univalent type theory has some slightly counter-intuitive consequences for the classical mathematician that is used to working with (h-)sets and never really bothers too much with alternative interpretations of equality. The chapters on algebraic topology and category theory in [VAC+13] show what some other consequences are of accepting the univalence axiom for classical mathematical domains and rejecting the UIP principle.

$$\begin{tikzcd} a \arrow[r, bend left, equal, "p"] \arrow[r, bend right, equal, "q"'] \arrow[r, phantom, "\|"] & b \end{tikzcd}$$

Figure 1.7: The type $\mathtt{Id}_{\mathtt{Id}_X(a,b)}(p, q)$ can be interpreted as a “second layer” of intensional identity types.

Chapter 2

Interpretations and models

The presence of a non-trivial hierarchy of equalities in type theory was somewhat unexpected and gave rise to a new field in foundations of mathematics that also had many applications [VAC$^+$13]. Mathematicians who are working in foundations can try to find models of a theory they are working with. Models with category theory prove the relative consistency of the system in relation with classical foundational frameworks such as set theory, intuitionistic logic and optionally the axiom of choice. This was also the case in (univalent) type theory which was first modelled with groupoids in [HS98]. This section will explore some of the models that exist for univalent type theory.

2.1 Groupoid interpretation

Before the univalence axiom was discovered, one of the first models of type theory was done in the language of groupoids [HS98] and is called the *groupoid model*. A *groupoid* is a concept of category theory that is inspired by the behaviour of groups.

Definition 2.1.1. *A* groupoid *is a category with only invertable morphisms.*

Let X be any type. The model in groupoids of type theory worked by interpreting every term of $a, b : X$ as an object and the terms of the intensional identity type $\texttt{Id}_X(a, b)$ as invertable morphisms (see fig. 1.7 where the equalities are replaced by morphisms). The tower of equalities needed to be interpreted as well. This was done with a higher-dimensional groupoid, called an ∞-*groupoid* or ω-groupoid, see the thesis [Lum10] for an extensive overview of the model in groupoids. An ∞-groupoid contains morphisms between morphisms and so on. However, for the model to be a correct model, it also needs to satisfy all the properties that are satisfied by the tower of equalities in type theory. These properties of equalities are translated as *coherence laws* of the ∞-groupoid and stated for example that the composition of two morphisms between morphisms of the ∞-groupoid should be associative. But these coherence laws became very complex for higher levels of morphisms in the ∞-groupoid and there was not a clear simple way to state them generically.

The tower of equalities inspired [Voe16] to define a hierarchy of levels of types above h-sets. This hierarchy, that can be proven to be cumulative, is defined by induction on the integers, starting from -2.

Figure 2.1: In the context of topological spaces and algebraic topology, a path between two curves p, q starting and ending at the same base points a, b is called a *homotopy of paths*. In this picture there are homotopies s and r. Formally, a homotopy of paths is in general a continuous function $F : [0,1] \to X$ that satisfies $F(0,u) = a, F(1,u) = b, \forall u \in [0,1]$ and $F(v,1) = p(v), F(v,0) = q(v), \forall v \in [0,1]$. The study of homotopic paths leads to the study of fundamental or homotopy groups, see section 3.5.3.

Definition 2.1.2. *A type X has* h-level -2 *if it is contractible, see def. 1.3.1. The type X has h-level $n+1$ if for any term $a, b : X$, the intensional identity type $\mathtt{Id}_X(a,b)$ has h-level n.*

For example, types that are h-props are by definition on level -1 and h-sets are on level 0.

According to [Voe16] the h-level hierarchy can help to distinguish different levels of doing mathematics. On the bottom, there is element-level mathematics, the level corresponding to h-level -1. Working with h-level -1 types corresponds to working with elements of sets in classical mathematics. There is also set-level mathematics, which is about working with h-level 0 types, the types that correspond to h-sets or sets in classical mathematics. On top of all of this, there is higher-level mathematics, the mathematics that is about type-theoretical analogues to categories, ∞-groupoids or other higher-dimensional category theory.

2.2 Homotopy type theory

Besides the groupoid model from section 2.1, there are also other models. The homotopy interpretation of univalent type theory, was motivated by the complexity of the ∞-groupoid model and algebraic topology. In algebraic topology, the properties of topological, continuous spaces are studied with methods from group theory. In the homotopic interpretation of type theory, the type X is interpreted as a topological space and the intensional identity type $\mathtt{Id}_X(a,b)$ is interpreted as all the continuous paths between points a, b in X considered as topological space. Because paths are reversible, the symmetry of the intensional identity type is consistent and the other properties of equality can be verified to be consistent as well. Let $p, q : \mathtt{Id}_X(a,b)$ be two terms interpreted continuous paths (compare this with the construction in fig. 1.7). Terms $r, s : \mathtt{Id}_{\mathtt{Id}_X(a,b)}(p,q)$ are interpreted as two-dimensional paths, see fig. 2.1. This explains the naming *homotopy type theory* for univalent type theory, when interpreted using algebraic topology.

The definition of h-levels can be done in this interpretation as well. Here, a type is on h-level n if its analogue in the homotopy theoretic model, a topological space, has vanishing homotopy groups above order n. This topological model can be extended to all concepts in univalent type theory by giving interpretations to each concept in the language of homotopy theory, see table table 2.1. So the study of types in univalent type theory corresponds to the study of higher order homotopy groups.

Logic	Types	Homotopy
proposition	A	topological space
proof	$a : A$	point in space
$A \Rightarrow B$	$A \to B$	continuous functions
predicate	$B : A \to \mathcal{U}$, denoted $B(x)$	covering
conditional proof	$b(x) : B(x)$	section of a covering
$\bot, \top$	$0, 1$	$\emptyset, \star$
$A \vee B$	$A + B$	co-product
$A \wedge B$	$A \times B$	product space
$\exists_{x:A} B(x)$	$\sum_{(x:A)} B(x)$	total space of a covering
$\forall_{x:A} B(x)$	$\prod_{(x:A)} B(x)$	space of sections
equality	$\mathtt{Id}_A$	path space $A^{[0,1]}$

Table 2.1: The homotopic interpretation of type theory as written in the introduction of [VAC+13] gives a comparison of the concepts of univalent type theory together with their interpretations in the topological model and intuitionistic logic, see section 1.2.4.

2.3 Simplicial model

The topological spaces used in the homotopic interpretation of univalent type theory are actually not really topological spaces but simplicial sets, Kan complexes with fibrations. See [Str06] and [Voe09] for how universes are modeled in simplicial sets. The simplicial set model was completed in [KL12] and proved to be very useful for later models of univalent type theory such as the one in [CCHM16]. The model in simplicial sets does however have the same homotopic properties, the homotopy of spaces stays the same whether they are viewed as topological spaces or Kan complexes, so it is a simplification that can be made. This is also called the *homotopy hypothesis* [SCS+18]. Because the model in simplicial sets was proven to be consistent, it also proved that univalent theory as a mathematical theory was as consistent as Zermelo-Fränkel set theory with the axiom of choice. The usage of the axiom of choice made it non-constructive however, meaning that the translation of proofs in univalent type theory into the language of simplicial sets also became non-constructive. It was as well cumbersome to implement in computers, because simplicial sets and more specifically their building blocks, simplexes, do not have nice combinatorial properties. This lead to the development of cubical type theory, see chapter 3.

2.4 Categorical semantics

Both the groupoid and simplicial model turned out to have some problems. To properly deal with these problems, new constructive and easy to implement models have to be found. Models of type theory can be built with the framework of *categories with families* [Hof97], see fig. 2.2.

Categories with families can be seen as algebraic representations of type theory. The idea behind a category with families is to translate the types, terms and typing rules from type theory to a model in category theory.

Figure 2.2: Martin Hofmann (1965-2018) was a German computer scientist and the founder of the groupoid interpretation [HS98] and popularized the usage of categorical semantics in type theory [Hof97]. In January 2018, he died in a snow storm on the mountain Nikko Shirane [Rod18].

Definition 2.4.1. *A* category with families *model is a category and a functor denoted by* $(Ctx, \mathcal{F})$ *that satisfies props. 2.4.2 and 2.4.4 to 2.4.7.*

Property 2.4.2. *The* context category *Ctx is a category whose objects represent contexts of the type theory. There is a terminal object* $()$ *in Ctx, called the* empty context*. This is an object such that for every other object* $\Gamma \in Ctx$*, there is a unique morphism* $\Gamma \to ()$*.*

The objects of this category are denoted with capital Greek symbols Δ, Γ and are simply called contexts, even though they only represent or model the contexts from type theory. If the choice for Ctx is clear from the context,

$$\Gamma \vdash$$

denotes that Γ is a context in the context category Ctx. The morphisms of the category Ctx are called *substitutions* or *context morphisms*.

The first thing it is necessary is a way to capture the relation between terms and their types in set (and category) theory. This is done with a family of sets.

Definition 2.4.3. *The category of* families of sets ***Fam*** *has as objects pairs of the form* (A, B) *where* A *is a set and* $B = (B_a \mid a \in A)$ *is a* A*-indexed family of sets* $B_a, a \in A$*.*

A morphism between $(A, B) \to (A', B')$ is given by a pair (f, g) where $f : A \to A'$ is a function and g is an A-indexed family of functions such that $g_a : B_a \to B'_{fa}$.

Property 2.4.4. *Types and terms are modeled by a functor* $\mathcal{F} = (Ty, Tm)$ *that goes from the opposite category* Ctx^{op} *to the category of families* ***Fam****.*

This functor takes a context Γ, returns a set denoted by $\mathrm{Ty}(\Gamma)$, called the *types* in context Γ and a family of sets indexed by elements $A \in \mathrm{Ty}(\Gamma)$, called the terms $\mathrm{Tm}(\Gamma, A)$ of A. Because these category-theoretical types represent the real types, the dependency $A \in \mathrm{Ty}(\Gamma)$ is often denoted by the judgement

$$\Gamma \vdash A.$$

Similarly, the elements $a \in \mathrm{Tm}(\Gamma, A)$ are called *terms* of A within the context Γ, denoted by

$$\Gamma \vdash a : A.$$

Take a substitution $\sigma : \Delta \to \Gamma$, then the morphism $\mathscr{F}(\sigma)$ is a morphism that acts on $\mathrm{Ty}(\Gamma)$.

Property 2.4.5. *The action of a substitution on a type $A \in Ty(\Gamma)$ is written as $A\sigma$ and the action on terms a is written as $a\sigma$. Because $\mathscr{F}$ is contravariant, $A\sigma \in Ty(\Delta)$ and $a\sigma \in Tm(\Delta, A\sigma)$. Let Γ' be yet another context. The substitutions on terms should satisfy the following composition laws: $(A\sigma)\tau = A(\sigma\tau)$ and $(a\sigma)\tau = a(\sigma\tau)$ for $\tau : \Gamma' \to \Delta$.*

Because contexts in type theory interact with other judgements, the category-theoretical model should be able to do this as well. For example, if Γ is a context and A is a type in Γ, there is a context $\Gamma.A$ called the *context extension.* Intuitively, this context has all the terms and types that are already in Γ and the type A, including the term $\Gamma \vdash q : A$ defined in prop. 2.4.6.

Property 2.4.6. *Given a context Γ and type $\Gamma \vdash A$, there is an object representing the context extension $\Gamma.A$, a morphism $p : \Gamma.A \to \Gamma$, and a term $q \in Tm(\Gamma.A, Ap)$, denoted by $\Gamma.A \vdash q : Ap$.*

Here the term q stands for the variable in A that was added the latest to $\Gamma.A$

Property 2.4.7. *The objects p and q should satisfy the* universal extension property*: for any context Δ, Γ, substitutions σ, τ and term a there is a substitution $(\sigma, a) : \Delta \to \Gamma.A$ such that*

- *The substitution σ is a partial inverse of p: $p(\sigma, a) = \sigma$.*
- *The substitution (σ, a) of the term q returns a: $q(\sigma, a) = a$.*
- *The substitution is invariant under composition of σ with other substitutions τ: $(\sigma, a)\tau = (\sigma\tau, a\tau)$.*
- *The context morphism (p, q) is the identity on $\Gamma.A$.*

There is an alternative presentation of the category with families model in [Ort19], section 3.1.

Given a category with families, it is possible to formalize the informal description in def. 1.2.4 with a very short definition:

Definition 2.4.8. *A type B is a* dependent type *depending on another type A within context $\Gamma.A$ if $\Gamma.A \vdash B$.*

If B is a dependent type, substitution of terms in B is done by lifting the morphism or substitution from the universal property of extensions where $\sigma = 1$ is the identity substitution of contexts. This gives a morphism $(1, a) : \Gamma \to \Gamma.A$ that can be lifted with the Ty-functor to a morphism $[a] : \Gamma \to \Gamma.A$. The action of $[a]$ on B gives $B[a]$ which is by the contravariance of $\mathscr{F}$ a type in the context Γ, so $\Gamma \vdash B[a]$.

2.5 Presheaves

In practice, a category with families is built with more fundamental concepts. A base category is chosen that it possesses the properties which are required for the contexts in the particular type theory that is being modelled. With this base category, a category with families is built. For example, if the contexts of the type theory have to be indexable by dimension variables, a base category has to be chosen that has objects which represent dimensions.

Examples of presheaves

Definition 2.5.1. *Let $\mathcal{C}$ be any category. A* presheaf *on $\mathcal{C}$ is a contravariant functor from $\mathcal{C}$ into the category of sets, **Set**. This is a mapping of objects and morphisms of categories that reserves arrows.*

- *The category of all presheaves on base category $\mathcal{C}$ has as objects the presheaves and as morphisms the natural transformations between presheaves. It is denoted by $Psh(\mathcal{C})$.*
- *Let I, J be two base objects the base category and $\Gamma \in Psh(\mathcal{C})$ a presheaf. The mapping of a morphism $f \in Hom_{\mathcal{C}}(J, I)$ under Γ is formally denoted by $\Gamma(f)$ but often just f and written as a right action. It is a morphism $\Gamma(f) : \Gamma(I) \to \Gamma(J) : \rho \mapsto \rho f$ where it is said that ρf is the* restriction *of ρ by f.*

Example 2.5.2. *Take as base category $\mathcal{C}$ a category with only one object $\star$, then the presheaves in $Psh(\mathcal{C})$ are contravariant functors from $\star$ to **Set** mapping the object to any set.*

Example 2.5.3. *Let $\mathcal{C} = \{0, 1\}$ with two morphisms $f, g : 0 \to 1$.*

$$\begin{tikzcd} 0 \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] & 1 \end{tikzcd}$$

A presheaf Γ on $\mathcal{C}$ consists of two sets $\Gamma(0), \Gamma(1)$ together with two functions $\Gamma(f), \Gamma(g) : \Gamma(1) \to \Gamma(0)$.

$$\begin{tikzcd} \Gamma(1) \arrow[r, bend left, "\Gamma(f)"] \arrow[r, bend right, "\Gamma(g)"'] & \Gamma(0) \end{tikzcd}$$

If $\Gamma(0)$ is interpreted as the set of vertices and $\Gamma(1)$ as the set of edges, there is a directed graph. The map $\Gamma(f)$ gives the starting vertex of an edge, $\Gamma(g)$ gives the ending vertex.

Example 2.5.4. *Take $\mathcal{C} = \{0, 1\}$, two morphisms $f, g : 0 \to 1$ as before and now a third morphism $h : 1 \to 0$ such that $h \circ f = h \circ g = id$. Then $\Gamma(0)$ is interpreted again as a set of vertices and $\Gamma(1)$ as a set of nodes.*

$$\begin{tikzcd} \Gamma(1) \arrow[r, bend left, "\Gamma(f)"] \arrow[r, bend right, "\Gamma(g)"'] & \Gamma(0) \arrow[l, "\Gamma(h)"'] \end{tikzcd}$$

In this interpretation, there is that $\Gamma(f) \circ \Gamma(h) = id_{\Gamma(0)}$ *and* $\Gamma(g) \circ \Gamma(h) = id_{\Gamma(0)}$*. This means that for each* $v \in \Gamma(0)$*,* $\Gamma(h)(v)$ *serves as a an edge with both starting and ending vertex* v*. This means that from every presheaf, it is possible to derive a unique* directed reflexive graph*.*

More examples of presheaves will arise later in the form of cubical sets (see def. 3.1.6) or can be found in [Hof14]. There does exist a concept of *sheaves*. Presheaves can be turned into a sheaf in a topological space by a sheaving functor. This implies presheaves are a generalization of sheaves contrary to the terminology. More specifically, sheaves are a *reflective subcategory* of presheaves, see [S+11]. Another related concept is the concept of a sieve, which arises in the context of cubical type theory [Coq15] and [CCHM16].

The presheaf model

But first, let's explain how it is possible to make a concrete category with families using presheaves. This explanation is based on [Hof97], section 4.1, page 45, [Hub16a], section 1.2 and [Ort19], section 3.2.

Definition 2.5.5. *Given a small category* $\mathcal{C}$ *(a category in which the morphisms form sets), called the* base category*, a* presheaf model *on* $\mathcal{C}$ *is defined as the category of families* $(Psh(\mathcal{C}), (Ty, Tm))$*, where the individual terms are defined in props. 2.5.6, 2.5.8 and 2.5.9 and have a specific mechanism for dealing with contexts defined in prop. 2.5.10.*

The first thing needed for the presheaf model to satisfy the definition of a category with families (see def. 2.4.1) is a category of contexts.

Property 2.5.6. *In a presheaf model over a base category* $\mathcal{C}$*, take the category of contexts to be the presheaf category* $Psh(\mathcal{C})$*. The empty context is the functor* 1 *mapping all the objects in the base category to the same set.*

The resulting category then satisfies prop. 2.4.2.

There also has to be a functor from the category of presheaves to families of sets that satisfies prop. 2.4.4 for types and terms in a category with families. To define this, another category has to be introduced.

Definition 2.5.7. *Given a functor* $A : \mathcal{C} \to \boldsymbol{Set}$*, the* category of elements *is the category denoted by* $\int_{\mathcal{C}} A$*:*

- *The objects are pairs* (c, a) *where* c *is an object of* $\mathcal{C}$ *and* $a \in A(c)$*.*
- *Take two objects* (c, a) *and* (c', a')*. Morphisms between these objects are denoted by* $(\phi \mid a)$ *for* $\phi : c \to c'$ *and* $a' = A(\phi)(a)$*.*

If the functor A is contravariant, define

$$\int_{\mathcal{C}} A = \left(\int_{\mathcal{C}^{\mathrm{op}}} A \right)^{\mathrm{op}}.$$

It is possible to now construct the functor $\mathcal{F}$ on the category of contexts from prop. 2.4.4. Let Γ be a context in the presheaf model, this means that Γ is a contravariant functor from the base category to the category of sets.

Property 2.5.8. *The interpretation of the types over a given context is denoted by* $Ty(\Gamma)$ *and defined as* $Psh\left(\int_{\mathcal{C}} \Gamma\right)$.

These presheaves form a set because the base category is small, so the morphisms form sets. This means that the presheaves can be used as the indexing set of a family of sets and the definition of Ty in prop. 2.5.8 makes sense.

When $A \in \mathrm{Ty}(\Gamma)$, A is called a type over context Γ or even use the type theoretical notation $\Gamma \vdash A$. But A is not a set of on its own or does not contain any terms. To work with types, it is necessary to look at the values of A (as a functor) which are written as $A\rho$ given $\rho \in \Gamma(I)$.

Property 2.5.9. *The interpretation of the terms over some context* Γ *and type* A *in* Γ *is denoted by* $Tm(\Gamma, A)$ *and defined as all the elements*

$$a \in \prod_{I \in \mathcal{C}, \rho \in \Gamma(I)} A(I, \rho)$$

such that $\forall f \in Hom_{\mathcal{C}}(J, I), (a\rho)f = a(\rho f)$. *In other words, to every base object* I, *set element* $\rho \in \Gamma(I)$ *a set element* $a\rho$ *is chosen in the set* $A(I, \rho)$ *defined by the given presheaf* A.

the set $\Gamma(I)$ *commutes with taking restrictions of terms* $a\rho$ *in the set* $A(I, \rho)$. *This means more precisely, that for all .*

The definitions in props. 2.5.8 and 2.5.9 make sure that $(\mathrm{Ty}, \mathrm{Tm})$ satisfies the requirements of prop. 2.4.4 and is an appropriate functor as required in def. 2.4.1.

However, to analyse the interpreted object deeper, it is necessary to observe which values $a\rho$, the term a, takes "above" an element $\rho \in \Gamma(I)$ according to prop. 2.5.9.

The definition of a category of families also requires to define extension of contexts for the presheaf model.

Property 2.5.10. *Given a context* Γ *and type* $\Gamma \vdash A$, *define the context extension in the presheaf model as a contra-variant functor. Given an object* $I \in \mathcal{C}$, *it takes values*

$$(\Gamma.A)(I) = \{(\rho, u) \mid \rho \in \Gamma(I), u \in A(I, \rho)\}.$$

This new functor is denoted by $\Gamma.A$. *Given* $\rho \in \Gamma(I), u \in A(I, \rho)$ *and a morphism* $f \in Hom_{\mathcal{C}}(J, I)$, $(\rho, u)f$ *is defined as* $(\rho f, uf)$.

The map p *in the presheaf model is defined by*

$$p : \Gamma.A \to \Gamma : (\rho, u) \mapsto \rho.$$

In other words, this is just the projection forgetting the second argument. Similarly, the term $q \in Tm(\Gamma.A, Ap)$ *is defined in the presheaf model as a mapping (polymorphic for an implicit object* I*)*

$$q : (I \in \mathcal{C}) \to I \times (\Gamma.A)(I) \to A(I, \rho) : (\rho, u) \mapsto u$$

that forgets the first argument.

The above construction gives an instance of the definition of a category with families (see def. 2.4.1): a presheaf model. This model is made using only the objects and morphisms in the base category and its presheaves. However, the exact representation depends on the choice of map for Ty and Tm. In very simple cases of base categories, the choice is limited. For example, the previously mentioned examples of graphs and sets (see ex. 2.5.4) become categories with families. This means that categories with families are things that really exist. When the base category is more complicated, the contexts and types behave very differently but there is a characterization of types in terms of context extensions given in lemma 2.5.12.

Definition 2.5.11. *An* equivalence of categories *between categories $\mathcal{C}$ and $\mathcal{D}$ is a pair of functors $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ and $\mathcal{G} : \mathcal{D} \to \mathcal{C}$ that satisfies the following properties:*

- *The composition of functors $(\mathcal{F} \circ \mathcal{G}) : \mathcal{D} \to \mathcal{D}$ is isomorphic through a natural transformation with the identity functor $I_{\mathcal{D}}$.*
- *There is a natural equivalence of functors $(\mathcal{G} \circ \mathcal{F}) \cong I_{\mathcal{C}}$.*

Lemma 2.5.12. *Take an arbitrary base category $\mathcal{C}$. If $\Gamma \in Psh(\mathcal{C})$, then there is an equivalence between the category of types $Ty(\Gamma)$ and the category*

$$U \equiv \{(\Delta, \sigma) \mid \Delta \in Psh(\mathcal{C}), \sigma \in Hom_{Ctx}(\Delta, \Gamma)\}$$

with as morphisms $Hom_U((\Delta_1, \sigma_1), (\Delta_2, \sigma_2))$ the context morphisms $\phi : \Delta_1 \to \Delta_2$ such that the following diagram commutes:

$$\begin{tikzcd} \Delta_1 \arrow[rr, "\phi"] \arrow[dr, "\sigma_1"'] & & \Delta_2 \arrow[dl, "\sigma_2"] \\ & \Gamma & \end{tikzcd}$$

Proof. 1. Let the first functor for the equivalence of categories be given by

$$\mathcal{F} : \mathrm{Ty}(\Gamma) \to U : A \mapsto (\Gamma.A, p).$$

The second functor, the candidate for an inverse of $\mathcal{G}$, is given by

$$\mathcal{G} : U \to \mathrm{Ty}(\Gamma) : (\Delta, \sigma) \mapsto ((I, \rho) \mapsto \{s \in \Delta(I) \mid \sigma(s) = \rho\}).$$

It will now be proven that the composite functor $\mathcal{F} \circ \mathcal{G}$ is isomorphic to the identity functor on U through a natural transformation. Take an object $(\Delta, \sigma) \in U$, it remains to prove that

$$(\mathcal{F} \circ \mathcal{G})(\Delta, \sigma) \cong (\Delta, \sigma).$$

But by writing out the definition of $\mathcal{G}$,

$$(\mathcal{F} \circ \mathcal{G})(\Delta, \sigma) = \mathcal{F}((I, \rho) \mapsto \{s \in \Delta(I) \mid \sigma(s) = \rho\})$$

which is in turn equal to

$$(\Gamma.((I, \rho) \mapsto \{s \in \Delta(I) \mid \sigma(s) = \rho\}), p).$$

This can be further simplified to

$$((I \mapsto \{(\rho, s) \mid \rho \in \Gamma(I), s \in \Delta(I), \sigma(s) = \rho\}), p).$$

In summary,

$$(\mathcal{F} \circ \mathcal{G})(\Delta, \sigma) \equiv (\Gamma.A, p)\,.$$

To prove $(\Gamma.A, p) \cong (\Delta, \sigma)$, it is necessary to prove that there is an isomorphism of contexts $\phi : \Delta \to \Gamma.A$ such that the following commutative diagram commutes:

$$\begin{tikzcd} \Delta \arrow[rr, "\phi"] \arrow[dr, "\sigma"'] & & \Gamma.A \arrow[dl, "p"] \\ & \Gamma & \end{tikzcd}$$

Because ϕ should be a morphism of contexts and morphisms of contexts are natural transformations of functors, ϕ has to be defined by a natural transformation. A natural transformation is given by morphisms $\alpha_I : \Delta(I) \to \Gamma.A(I)$ for all $I \in \mathcal{C}$ that satisfies commutativity properties. Set $\alpha_I : s \mapsto (\sigma(s), s)$ for every $I \in \mathcal{C}$. Each α_I is a bijection with inverse $\alpha_I^{-1} : (\rho, s) \mapsto s$. It only remains to prove the commutativity properties.

2. It still remains to prove the reverse direction. For a type $A \in \mathrm{Ty}\,(\Gamma)$, $(\mathcal{G} \circ \mathcal{F})(A) \cong A$ in the category of types, being presheaves on the category of elements. Indeed, take an element $(I, \rho) \in \int_{\mathcal{C}} \Gamma$. It remains to prove that

$$((\mathcal{G} \circ \mathcal{F})(A))(I, \rho) \cong A(I, \rho)$$

as sets. The left-hand side is by definition

$$\{s \in (\Gamma.A) \mid p(s) = \rho\} = \{(\rho', u) \mid \rho' \in \Gamma(I), u \in A(I, \rho), \rho' = \rho\}$$

and this is in bijection with $A(I, \rho)$.

The equivalence has been proven. □

Example 2.5.13. *If $\mathcal{C}$ is chosen to be the category with two objects and three morphisms as in ex. 2.5.4, the contexts in the presheaf model become directed reflexive graphs. Given a context Γ, a type $\Gamma \vdash A$ forms a dependent or nested direct reflexive graph.*

Proof. The lemma 2.5.12 tells that there is some context Δ and a morphism $\sigma : Delta \to \Gamma$. To verify that Δ is a dependent graph, it has to be proven that the morphism σ of presheaves corresponds to morphism of graphs. For this it is necessary, to show that the presheaf model of ex. 2.5.4 is equivalent as a category to the category of graphs. Because this has not been done yet, the claim will be verified directly instead.

A type A in the context Γ is according to prop. 2.5.8 a functor A from the category of elements to the category of sets:

- Every vertex in $\{(0, v) \mid v \in \Gamma(0)\}$ is mapped to a set $A(0, v)$. Every edge in $\{(1, e) \mid e \in \Gamma(1)\}$ is mapped to a set $A(1, e)$.

- The morphism $(f \mid e)$ in the category of elements is mapped to a set map $A(f) : A(1, e) \to A(0, \Gamma(f)(e))$. It is possible to do the same for the morphism g. Let $\Gamma(f)(e) = v$. f can be interpreted as sending an edge $e' \in A(1, e)$ in a new graph to an endpoint, a vertex v' which lies in a set $A(0, x)$ above the endpoint v of e in original graph Γ.

- The morphism $(h \mid v)$ is a mapped to a set map $A(h) : A(0, v) \to A(1, \Gamma(h)(v))$ which takes a vertex v' and maps it to an edge $A(h)(v') = e'$.

- The composition property $A(h \circ f) = A(f) \circ A(h)$ implies that

$$id = A\left(f \mid \Gamma(h)(v)\right) \circ A\left(h \mid v\right)$$

which is a map

$$A(0, v) \to A(0, \Gamma(f) \circ \Gamma(h)(v)) : v' \mapsto v'.$$

The same can be done with g. This can be interpreted as before that $A(h \mid v)(v')$ is an edge e' with identical start and end point v'.

This means that a type A is a refined reflexive graph of Γ, denoted by Δ with more vertices and edges, see the upper part of fig. 2.3.

A term is according to prop. 2.5.9 an element of $a\rho$ in $A(I, \rho)$ that depends on the choice of $\rho \in \Gamma(I)$ and satisfies an extra condition:

$$\forall J \in \mathcal{C}, f \in \mathrm{Hom}_{\mathcal{C}}\left(J, I\right), (a(I, \rho))\Gamma(f) = a(I, \rho\Gamma(f)).$$

- If $I = 0$, an element $\rho \in \Gamma(0)$ is an edge in Γ. It is possible to check how the maps $f, g : 0 \to 1$ act on terms. Assume for example that f (actually $\Gamma(f)$) is the map giving the source of edges. It is possible to do the same for g. The condition on terms tells that $(a\rho)f \equiv (a\rho)A(f \mid e)$ has to be equal to $a(\rho\Gamma(f))$. In other words, the source of $a\rho$ should be a vertex $a(\rho\Gamma(f))$ in the graph Δ of the vertex set that is above the source ρf (formally $\rho\Gamma(f)$) of ρ. See fig. 2.3.

- If $I = 1$, an element $\nu \in \Gamma(1)$ is a vertex in Δ and $a\nu$ is a vertex above ρ. Assume that $\rho f = \nu$. The only interesting map that acts on ν is the map h (or at least $\Gamma(h)$). νh gives the reflexive edge going from ν to ν. $a\nu$ also has a reflexive edge $(a\nu)h$. Now the condition says that the lift of the lower reflexive edge $a(\nu h)$ is the same one as the reflexive edge on the lifted vertex $(a\nu)h$.

The rules that are required by props. 2.4.4 and 2.5.9 tell that the lifting of nodes and edges is compatible. Taking σ as the graph map projecting lifed edges $a\rho \mapsto \rho$ and vertices $a\nu \mapsto \nu$, fig. 2.3 illustrates lemma 2.5.12 by showing that each type in the presheaf model as defined in def. 2.5.5 is a nested or dependent graph Δ that lifts the original graph Γ. $\square$

The presheaf model on a category with only two elements is in a sense the simplest example of a category with families there is. As soon more objects are added to the base category, the resulting category with families can be seen as a generalization of a reflexive directed graph.

A category with families supports a certain type if it is closed under the type rules of this type. It can be proven that every presheaf model supports all types from type

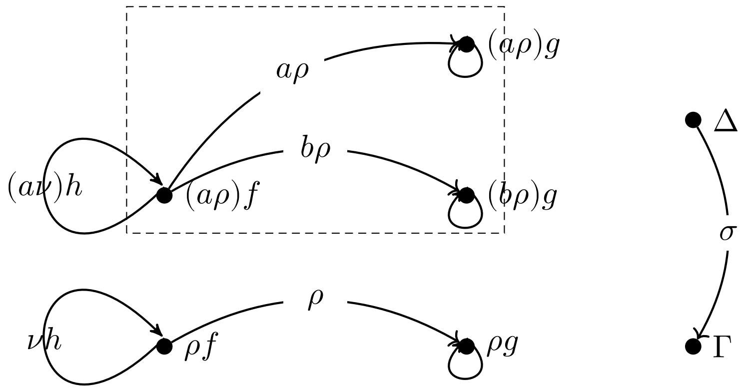

Figure 2.3: One edge ρ with endpoints ρf and ρg in Γ together with its lifts to Δ. Applying the terms a, b lifts the edge and its endpoits. The action of the morphisms f and g can be visualized as taking the endpoints of the edges $a\rho$ and $b\rho$ described by the terms a and b. The terms a and b also act on the node $\nu = \rho f$ and elevate it to higher nodes $a\nu$ and $b\nu$.

theory, see [Hof97], [Nuy18], page 9 or [Hub16a], section 1.2. However, the traditional interpretation of the intensional identity type in the presheaf model does not satisfy the typing rules of the intensional identity type in (univalent) type theory, see [Hub16a], section 1.2.3. The traditional intensional identity type is a constrained version that satisfied an extra axiom, called *axiom K* stating:

Axiom 2.5.14 (Axiom K)**.** *Let X be any type, if $\forall x : X$ and $p : (x = x)$, then $p = \texttt{refl}_x$.*

Because axiom K is related to the uniqueness of identity proofs principle (a relation that is covered in [VAC$^+$13], section 7.2), it is in contradiction with the univalence axiom, see ax. 1.3.2. A presheaf model does not necessary support the intensional identity type from type theory, an identity type that supports multiple terms. In univalent type theory the identity type must have multiple terms and not satisfy ax. 2.5.14. This means that not any presheaf model is sufficient to model univalent type theory. To find a presheaf model for univalent type theory in category theory, it is necessary need to look further into variations on the presheaf model explained above.

Chapter 3

Cubical type theory

At this point a general framework for making models of type theory in category theory has been discussed, see section 2.4. However, remember from section 2.3 that the goal is to find a constructive model in category theory that allows to prove the univalence axiom by generating terms of instances of it and using these terms in further computations. The model in cubical sets [BCH14] (and the slightly different one in [CCHM16]) was proven to be a model of univalent type theory, so the univalence axiom holds in it and it possible to generate terms of it. Actually, there are even more constructive models of the univalence axiom but in this text the focus will be on [CCHM16] and its recent extensions in [CHM18] and [MV19], see section 3.6.1 for other options.

The model with cubical sets can help with computing with terms of the univalence axiom because it is easier to implement than other models and completely constructive. This means that all the evaluation the generated terms in this model does not get stuck and can always be simplified or computed into the primitives of the model. Proof of concept is the recent implementation of this model in the proof assistant Agda [MV19].

3.1 The face lattice

Cubical type theory is based on cubical sets which are built with cubes (coming from dimension variables). The vertices of the cubes will describe the steps in applying a chain of (univalent) equalities as defined in def. 1.2.6. Computing with equalities in the cubical set model is done by computing with cubes, so it is necessary to formulate precisely what a cube is. This will be done with an algebra of faces [Hub16a]. Here, faces are a generalization of the concept of vertices, edges and faces of 3-dimensional cubes to their higher-dimensional variants.

Based on [BD77]:

Definition 3.1.1. *A* De Morgan algebra *over a set L is an algebra with:*

- *two binary operations $\wedge$ and $\vee$ called* connections *that are distributive: $\forall x, y, z \in L, x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.*
- *a unary operator $\neg$ called the* involution *such that the following laws hold:*
  - *double negation: $\forall x \in L, \neg\neg x = x$,*
  - *De Morgan laws: $\forall x, y \in L, \neg(x \vee y) = \neg x \wedge \neg y, \neg(x \wedge y) = \neg x \vee \neg y$.*

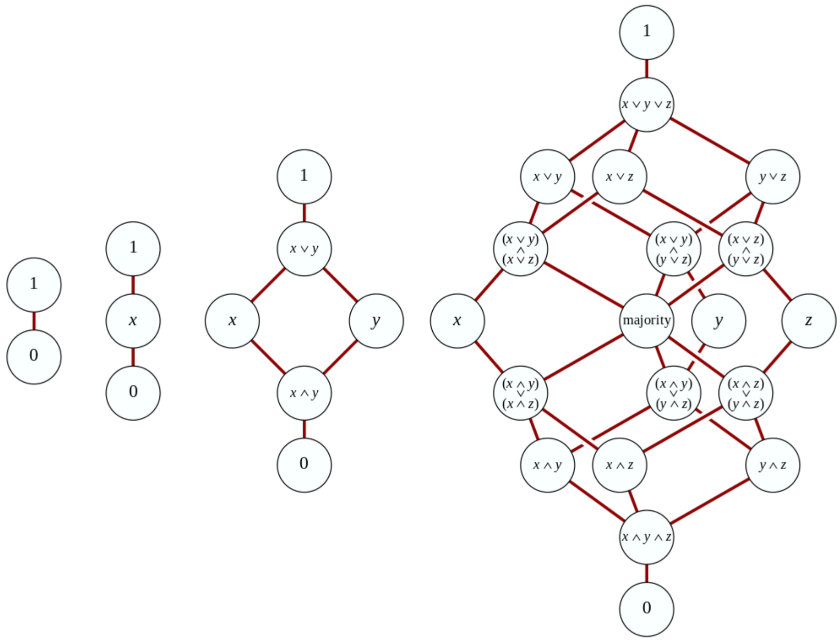

Figure 3.1: An illustration of the free bounded distributive lattice over generating sets of dimension variables $I = \emptyset$,$I = \{x\}$, $I = \{x, y\}$ and $I = \{x, y, z\}$ by writing edges $a \to b$ if and only if $a \leq b$. This lattice gives rise to a De Morgan algebra $dM(I)$ by defining $\wedge$ as the infininum in this lattice and $\vee$ as the supremum. Illustration taken from [Epp10].

- *bottom* 0 *and top element* 1 *satisfying* $\forall x \in L, x \wedge 0 = 0, x \vee 1 = 1$. *Also,* $\neg 0 = 1$ *and* $\neg 1 = 0$.

Definition 3.1.2. *Let* $\mathbb{A}$ *be a set containing at most countably many elements. This set is called the* set of names *because the elements* $x \in \mathbb{A}$ *serve as names for dimensions of the cubes.*

It is possible to consider a De Morgan algebra generated by dimension names. In this algebra, the individual variables describe the lines of a cube but the composite lattice elements that are also present in the algebra allow to define values on the interior of a cube.

Definition 3.1.3. *Let* $I \subset \mathbb{A}$ *be a finite subset. The* free De Morgan algebra *generated by* I *is the smallest De Morgan algebra that contains all expressions with elements of* I*, the operators* $\vee, \wedge, \neg$ *and the bounding elements* $0, 1$*. It can also be denoted by* $dM(I)$ *or* $\mathbb{I}$.

Every bounded distributive lattice (a partially ordered set with relations) gives rise to a free de Morgan algebra, see def. 3.1.1. This means that the connections can be interpreted as follows: $\wedge$ corresponds to taking the mininum and $\vee$ corresponds to taking the maximum.

The involution of an element x in a free De Morgan algebra is sometimes denoted by $1-x$ instead of $\neg x$. Viewing a De Morgan algebra as a lattice with extra operations (see def. 3.1.1), the involution can be seen as vertical mirroring within the lattice). When viewing the dimension variable x as a line in a cube, the involution of x represents the inverted line.

Definition 3.1.4 (Category of cubes)**.** *The category $\mathcal{C}$ contains as objects the finite subsets $I \subset \mathbb{A}$, called (dimension) names. Morphisms in this category are maps into De Morgan algebras. More precisely, given two objects $I, J \in \square$, $f \in Hom_{\square}(J, I)$ if and only if $f : I \to dM(J)$ is a set map.*

- *If $f \in Hom_{\square}(J, I), g \in Hom_{\square}(K, J)$, the composition $g \circ f$, written as fg, is defined as the composition $\hat{g} \circ f$, where $\hat{g}$ is the extension of g to the free De Morgan algebra $dM(J)$.*
- *The identity map in this category is just the identity.*

From now on, until the rest of this chapter, the symbol $\square$ will be used to denote the category defined in def. 3.1.4. $\square$ can be interpreted as an abstract higher-dimensional cube on which it is possible to apply operations described by a De Morgan algebra. As a base category it has much more objects and morphisms than the category that was used to construct directed and reflexive graphs, see ex. 2.5.4. As a consequence, types in a presheaf model over this category will be more complicated than in fig. 2.3.

Now a few examples will be given of morphisms that are present in this category and will be used in the rest of the text.

Example 3.1.5. *Let $I \subseteq J$. If $i \notin I$ then the morphism $s_i : I \cup \{i\} \to I$ is defined as the set map associated to the inclusion $I \subset dM(I \cup \{i\})$. Given $i \in I$ and $r \in dM(J)$, the morphism $(i/r) : I \to dM(J)$ maps the dimension name i on the element r of the free De Morgan algebra $dM(J)$ and leaves other names untouched. These maps are also called* substitutions *although they are not exactly the same as context substitutions.*

- *The* face maps *are compositions of substitutions (i/b) where $b \in \{0, 1\}$. These maps give the subfaces or edges of a higher-dimensional cube.*
- *The substitutions $(i/1 - i)$ can be interpreted as reflections along the i-axis or dimension of a cube. When intensional identity type will be modelled with higher-dimensional cubes, the terms will correspond with edges on a cube or paths and this substitution can be used to model the reverse path or equality.*
- *The substitutions $(i/i \wedge j)$ extend a line parametrized by i to a square depending on j as well.*

In presheaf models of type theory, the contexts, types and terms of the type theory are modelled with presheaves. In the cubical set model, the presheaves are taken over $\square$.

Definition 3.1.6. *A* cubical set Γ *is a presheaf* $\Gamma \in Psh(\mathcal{C})$.

Given two finite sets of dimension variables $I, J \in \square$ and a cubical set Γ, there is an element $\rho \in \Gamma(I)$. Using the assumption that Γ is a covariant functor, every substitution $f \in \mathrm{Hom}_{\square}(I, J)$ gives a set map $\Gamma(f) : \Gamma(I) \to \Gamma(J)$. The application of $\Gamma(f)$ to the object $u \in \Gamma(I)$ is written on the right and the context Γ is implicit: $\rho f \in \Gamma(J)$.

Example 3.1.7. *If $I = \{x, y\}$, it is possible to interpret $u \in \Gamma(I)$ as a square with for example the edge $u(x/0)$ and corner $u(x/0)(y/0)$. In general, $u \in \Gamma(J)$ behaves like a $|J|$-dimensional cube and the face maps give faces of this cube.*

Example 3.1.8. *Let R be a commutative ring with a unit element. Define for I a finite set of unknowns $\{x_1, ..., x_n\}$, the set $R[I]$ of all polynomials in these unknowns. Then $R[-]$ can be considered as a functor and even a cubical set. For example if $p(x, y, z) = 1 + x^2y + z \in R[x, y, z]$, it is possible to apply the face map $(y/0)$ to obtain $(1 + x^2y + z)(y/0) = 1 + z$.*

Definition 3.1.9. *The* face lattice $\mathbb{F}$ *is the distributive lattice generated by the face maps (which are the substitutions of the form $(i/b), i \in \mathbb{A}, b \in \{0, 1\}$). An element of the lattice $\varphi \in \mathbb{F}$ is called an* extent. *The top element in this lattice is denoted by $1_{\mathbb{F}}$ and the bottom element by $0_{\mathbb{F}}$ but the reference to $\mathbb{F}$ is often omitted in literature. The face lattice also satisfies $(i/0) \wedge (i/1) = 0_{\mathbb{F}}$.*

It is possible to define a cubical set using the face lattice. For a $I \in \square$, set $\mathbb{F}(I)$ to be the subset of extents in $\mathbb{F}$ that only contains dimension names $i \in I$. If $f \in \mathrm{Hom}_{\square}(J, I), \mathbb{F}(I) \to \mathbb{F}(J) : \varphi \mapsto f(\varphi)$ where f just substitutes the symbols in φ and applies the substitution rules on [Ort19], page 39. On the other hand, $\mathbb{F}$ can also be seen as a type in the presheaf model over $\square$.

Remember from def. 2.5.5 that in such a presheaf model, contexts are presheaves and types are presheaves over a category of elements. The contexts in the presheaf model over the cube category $\square$ are by def. 3.1.6 denoted by "cubical sets".

The interpretation of the empty context () in this presheaf model is by prop. 2.5.6 a presheaf denoted by 1 that maps all objects in the base category $\square$ to the same set $\{\star\}$.

To define $\mathbb{F}$ as a type in the empty context within the presheaf model, it is necessary to describe how $\mathbb{F}$ acts on the category of elements. By definition of the category of elements (see def. 2.5.7),

$$\int_{\square} 1 = \{(I, \rho) \mid I \in \square, \rho \in 1I\} \cong \{(I, \star) \mid I \in \square\} \cong \square$$

where the last isomorphism is just a bijection that works by sending $(I, \star)$ to the set $\mathbb{F}(I)$.

Definition 3.1.10. *The* interval object $\mathbb{I}$ *represents the unit interval. It is modelled as a context in the presheaf model. Such a model is a functor $\square \to$ **Set** $: I \mapsto dM(I)$. $(i = 0)$ acts as a term $\mathbb{I} \vdash (i = 0) : \mathbb{F}$ on elements $\rho \in dM(I)$. This map is defined as the map setting all the occurences of i in ρ to 0 and applying the rules of the distributive lattice.*

The interval object (or any other context) can also be seen as a type in the empty context $() \vdash \mathbb{I}$. In this interpretation it is a functor $\int_{\square} \Gamma \to \mathbf{Set} : (I, \rho) \mapsto dM(I)$.

3.2 Manipulating contexts

It would now be possible to describe how traditional types that are defined in (univalent type theory) can be interpreted in this particular presheaf model. However, the face and interval objects form a very important part of cubical type theory because of their interaction with other contexts. To understand this interaction, it is necessary how types interact with contexts in general.

The presheaf model over $\square$ also contains the context extensions from prop. 2.5.10: if $\Gamma \vdash A$ is a type, then the context extension is defined by

$$(\Gamma.A)(I) = \{(\rho, u) \mid \rho \in \Gamma(I), u \in A(I, \rho)\}.$$

Given $\Gamma \vdash A$ and an interpretation in the presheaf model, there is always a term q in

$$\Gamma, A \vdash q : A.$$

This is used in cubical type theory in combination with the interval object to parameterise terms and types by dimension variables. For example if Γ is any context, there is the type $\Gamma \vdash \mathbb{I}$ and an extension $\Gamma.\mathbb{I}$ but also a term $\Gamma.\mathbb{I} \vdash q : \mathbb{I}$. r $(\Gamma, q : \mathbb{I})$ where q is as before but depends on the context Γ and type $\mathbb{I}$. In this situation and in literature q is replaced by a letter i, j, k and written in the context extension part. More precisely one writes $\Gamma, (i : \mathbb{I})$ instead of $\Gamma.\mathbb{I}$. If

In the presheaf model on cubical sets, there is another way to form new contexts using extents. But to introduce this formally, it is necessary to pinpoint how extents are interpreted as terms.

Definition 3.2.1. *In general, the type $\Gamma \vdash \mathbb{F}$ is defined in the presheaf model by the contravariant functor*

$$\mathbb{F} : \{(I, \rho) \mid I \in \square, \rho \in \Gamma(I)\} \to \textbf{\textit{Set}} : (I, \rho) \mapsto \mathbb{F}(I)$$

where ρ is just ignored. Terms $\Gamma \vdash \varphi : \mathbb{F}$ are defined by a dependent function

$$\varphi : (I : \square) \to (\rho : \Gamma(I)) \to \mathbb{F}(I, \rho) : I \mapsto \rho \mapsto \varphi\rho$$

where I is implicit.

To calculate with φ, it is necessary a precise interpretation in the presheaf and categeries with families model. Define $0_{\mathbb{F}}, 1_{\mathbb{F}} \in \mathbb{F}$ to be terms $0, 1 : \mathbb{F}$ in the model by setting $0\rho = 0, 1\rho = 1$ for all $\rho \in \Gamma(I)$, $\Gamma \in Psh(\square)$ and $I \in \square$.

For other extents, the definition or interpretation can depend on the context. When φ is an extent containing only variables $i, j, k \in \mathbb{A}$, there is a term $(i : \mathbb{I}), (j : \mathbb{I}), (k : \mathbb{I}) \vdash \varphi : \mathbb{F}$ also denoted $(i, j, k : \mathbb{I}) \vdash \varphi : \mathbb{F}$. It's interpretation in the category with families is as follows. Take an $I \in \square$ and $\rho \in (i, j, k : \mathbb{I})$. Then $\rho = ((\rho_i, \rho_j), \rho_k)$, where $\rho_i \in dM(I)$, $\rho_j \in \mathbb{F}(I, \rho_i)$ which means that ρ_j is an extent that only uses the variables in I. Further on, $\rho_k \in \mathbb{F}(I, (\rho_i, \rho_j))$. Now it is possible to define $\varphi(I, \rho) = \varphi(I, (\rho_i, \rho_j, \rho_k))$ as the extent where every variable i occuring in φ is replaced by the corresponding ρ_i and apply the rules on [Ort19], page 39.

Definition 3.2.2. *Given a context Γ and term $\Gamma \vdash \varphi : \mathbb{F}$, it is possible to define a new context (Γ, φ), called the* context restriction by extent *φ. It is defined as*

$$(\Gamma, \varphi)(I) = \{\rho \in \Gamma(I) \mid \varphi\rho = 1_{\mathbb{F}} \in \mathbb{F}(I)\}.$$

Example 3.2.3. *Let $\Delta \equiv (\mathbb{I}, (i = 0))$. Then it is possible to prove that $\Delta \cong ()$ as contexts.*

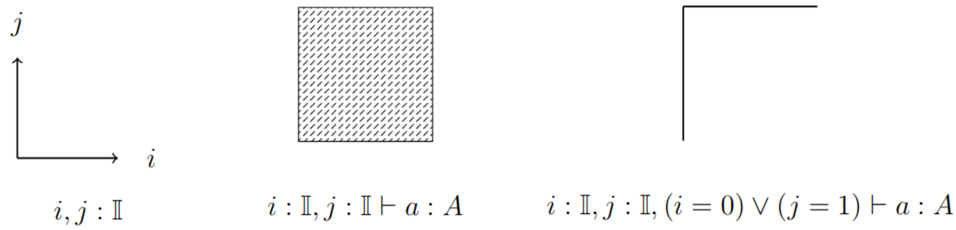

Figure 3.2: An illustration of the context restriction by $\varphi = (i = 0) \vee (j = 1)$ coming from [Ort19].

Proof. It rmeains to prove that there is a natural transformation between the two functors. First it has to be proven that for all $I \in \square$, the sets $\Delta(I)$ and $()(I) = \{\star\}$ are isomorphic. In other words, it is necessary prove that the set $\Delta(I)$ exactly contains one element. But by definition of the interval object and the context restriction $\Delta(I) = \{\rho \in dM(I) \mid (i = 0)\rho = 1\}$. But the only way that this could lead to 1 is when $\rho = 1$. So $\Delta(I)$ contains one element: $\rho = i \vee 1$. It is still necessary to verify the commutativity properties of the natural transformation. $\square$

Similarly, it can be proven that the context $(i : \mathbb{I}, i = 0, i = 0)$ is isomorphic to the context $(i : \mathbb{I}, i = 0)$.

There is an inclusion of the restricted context in the original context using a context morphism $\iota_\varphi : (\Gamma, \varphi) \to \Gamma$. Intuitively, this restricted context can be used to define types and terms that are only defined the faces of the higher-dimensional cube Γ specified by the extent φ.

Example 3.2.4. *A term $i : \mathbb{I}, j : \mathbb{I} \vdash a : A$ can be thought of as a square that is indexed by dimension variables i and j. The top and left sides of the square are described by the lattice formula $\varphi = (i = 0) \vee (j = 1)$. So if the attention has to be restricted to these sides, it is possible to look at a in the restriction of the original context: $i, j : \mathbb{I}, \varphi \vdash a : A$. This is illustrated in fig. 3.2.*

The operation of extending (actually restricting) a context by an extent, can be viewed as a typing rule:

$$\frac{\Gamma \vdash \varphi : \mathbb{F}}{\Gamma, \varphi \vdash}$$

Definition 3.2.5. *Let $\Gamma \vdash A$ and $\Gamma \vdash \varphi : \mathbb{F}$. A term u is called a* partial element of extent φ *if it is a term $(\Gamma, \varphi) \vdash u : A\iota_\varphi$.*

There is also another definition of a partial element [Ort19]. This alternative definition let's the choice of u depend on $\rho \in \Gamma(I)$.

one should I use

Definition 3.2.6. *Let Γ be context and $I \in \square, i \notin I, \rho \in \Gamma(I \cup \{i\}), \varphi \in \mathbb{F}(I)$. A partial element u of $A(\rho)$ is a family $u_f \in A(\rho f)$ for every $f : J \to I \cup \{i\}$ such that $\mathbb{F}(s_i \circ f)(\varphi) = 1_\mathbb{F}$, with the property that for any $g : K \to J$, $u_{f \circ g} = A(g)(u_f)$. Call φ the* extent *of the partial element u.*

For each $\Gamma \vdash u : A$ and extent φ, there is partial element of extent φ, or term $(\Gamma, \varphi) \vdash u\iota_\varphi : A\iota_\varphi$. When a term $(\Gamma, \varphi) \vdash v : A\iota_\varphi$ is induced by a restriction by an extent φ, or in other words $v = u\iota_\varphi$, v is called *extensible*. This can be intuitively seen as v being extendible to u by dropping the restriction by the extent φ. In type notation this relation between u and v is written as $\Gamma \vdash u : A[\varphi \mapsto v]$ expressing that φ extends the term v through φ. This can clarify the use of the word extent.

This specific notation for extents does not really state the presence of new types, contexts or terms in the model but can also be seen as a typing rule and written with natural deduction:

$$\frac{\Gamma \vdash v : A \qquad \Gamma, \varphi \vdash v = u : A}{\Gamma \vdash u : A\ [\varphi \mapsto v]}$$

Definition 3.2.7 (Generalized extent)**.** *A generalization of extents, in this text called* generalized extent*, is given in [Hub16a], page 95 and can be summerized as follows:*

$$\frac{\Gamma \vdash a : A \qquad \Gamma, \varphi_i \vdash a = u_i : A, \quad i = 1, ..., k}{\Gamma \vdash a : A\ [\varphi_1 \mapsto u_1, ..., \varphi_k \mapsto u_k]}$$

Definition 3.2.8 (System introduction rule)**.** *Assume that the following terms are given:*

- *for each $1 \leq i \leq n$, there is a term $\Gamma, \varphi_i \vdash t_i : A$ that is only defined on the extent φ_i,*
- *the extents cover the whole hypercube: $\Gamma \vdash \varphi_1 \vee ... \vee \varphi_n = 1_\mathbb{F} : \mathbb{F}$*
- *the definitions are compatible $\Gamma, \varphi_i \vdash t_i = t_j : A, \forall 1 \leq i, j \leq n$,*

Then there exists a term of type A, called a system*, which is denoted by $[\varphi_1\ t_1, ..., \varphi_n\ t_n] : A$. This term also has to satisfy*

This term also has to satisfy the elimination rules mentioned in [Hub16a], figure 6.4. A system allows to treat compatible terms, terms that agree at all points of the hypercube, as one term. The following lemma proves that compositions can be treated as generalized extents.

Lemma 3.2.9. *Let $\Gamma \vdash \varphi : \mathbb{F}$ and $\Gamma \vdash A$. If $\varphi = \bigvee_i \varphi_i$ and $\Gamma \vdash \varphi_1 \vee ... \vee \varphi_n = 1_\mathbb{F} : \mathbb{F}$, then the following types are logically equivalent:*

- $A\ [\varphi \mapsto [\varphi_1\ t_1, ..., \varphi_n\ t_n]$
- $A\ [\varphi_1 \mapsto t_1, ..., \varphi_n \mapsto t_n]$

Proof. It is believed but has not yet been completely verified that this follows from the definition of generalized extents and the elimination rule in [Hub16a], p. 95, figure 6.4 which states for any valid expression J:

$$\frac{\Gamma, \varphi_i \vdash J, 1 \leq i \leq n \qquad \Gamma \vdash \varphi_1 \vee ... \vee \varphi_n = 1_\mathbb{F} : \mathbb{F}}{\Gamma \vdash J}$$

□

Example 3.2.10. *The notation $[(i = 0) \vee (i = 1) \mapsto [(i = 0)\ u, (i = 1)\ v]$ can be replaced by $[i = 0 \mapsto u, i = 1 \mapsto v]$. This is done very often in [Hub16a] and implementations [MV19].*

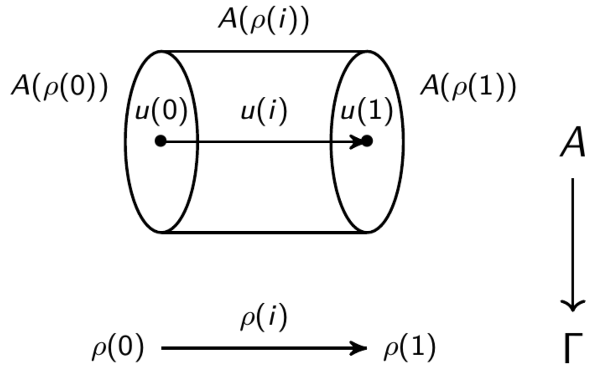

Figure 3.3: Picture taken from [Hub16b].

3.3 Adding operations

It is possible to now study the category theoretical interpretations of other terms and types in the presheaf model of cubical sets. In the presheaf model over the cube category, types have an intuitive description as cylinders. When $\Gamma \vdash A$ is a type in the presheaf model over cubical sets, it is described by a functor from the category of elements

$$\mathrm{Ty} : \int_{\square} \Gamma \to \mathbf{Set}.$$

This is noted as $A(\rho) \in \mathbf{Set}$ for some $|I|$-dimensional hypercube $\rho \in \Gamma(I)$. At the same time the substitutions that apply to objects in $\square$ and substitute dimension variables can be lifed by Ty to the level of types.

Example 3.3.1. *Take a type A in some cubical context Γ. If $I = \{x\}$ is one dimensional and $\rho \in \Gamma(I)$, subsitutions (x/r) that map the dimension variable x on 0 or 1 can be visualized as mapping ρ on the endpoints of an edge. The lift $A(\rho)$ is mapped by (x/r) on the faces of the cylinder similarly to the action of the morphisms f and g in fig. 2.3. Another way to interpret the type A is using lemma 2.5.12 by viewing the type A as a new context Δ. This is visualized in fig. 3.3.*

The interesting properties of univalent type theory and the consequences of the univalence axiom all follow from the intensional identity type and its typing rules. The intensional identity type of univalent type theory can be modelled in cubical sets but it is special in the following sense. Given a type A and terms $a, b, c : A$, the interpretation of the intensional identity type $a =_A b$ and $b =_A c$ in cubical sets is only transitive if A satisfies a *Kan extension property* (see fig. 3.4). This property of types A tells how the terms of A can be extended from a restricted context cube to a full cube. It is made precise with a composition and filling operation on the types and terms in the presheaf model with cubical sets.

Adding *operations* to the the model is done by adding axioms stating the existence of certain structures, sets, functors or elements, satisfying properties that are related to the properties of the type system that is modelled. Operations will be denoted with another font type. To arrive at a complete model of univalent type theory, including the univalence axiom, also a glue operation has to be added (see section 3.3.4).

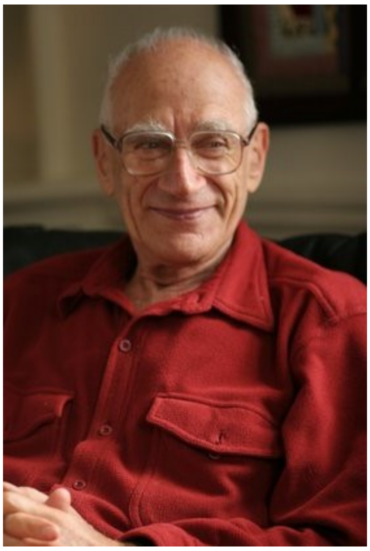

Figure 3.4: Daniel Kan (1927-2013) was a Dutch category and homotopy theorist. He defined his Kan extension property in the context of algebraic topology for cubical sets, see his book [Kan55] saying how boxes can be filled. Although [CHS19] mentions a predecessor to this property in [Eil39]. This Kan extension property later involved into different flavors such as a version for simplicial sets stating that every "horn" can be filled and the composition operation, see def. 3.3.2. Picture taken in his home in Massachusetts by [Kan05].

3.3.1 The composition operation

The most important operation in cubical type theory is the composition operation [Ort19]. The semantics of the composition operation are as follows:

Definition 3.3.2. *A* composition structure *for* $\Gamma \vdash A$ *is an operation,* $\texttt{comp}$, *that depends on the following elements:*

- $I \in \square, i \notin I, \rho \in \Gamma(I \cup \{i\}), \varphi \in \mathbb{F}(I)$, *where* ρ *can be seen as an* $(I+1)$*-dimensional hypercube.*
- u *a partial element of* $A(\rho)$ *of extent* φ *and* $a_0 \in A(\rho(i/0))$ *such that for all* $f : J \to I$, $A(f)(a_0) = u_{(i/0)\circ f}$ *and* $\mathbb{F}(f)(\varphi) = 1_{\mathbb{F}}$. *Here* u *can be viewed as a set of values for* A *on sides* φ *of* ρ *that takes values* a_0 *on the bottom of* ρ *described by* $i = 0$. *The value* a_0 *has to agree with the values* u *on other sides.*

Then there is a set element $\texttt{comp}(I, i, \rho, \varphi, u, a_0) \in A(\rho(i/1))$. *This element is uniform in the sense that for any* $f : J \to I$ *and* $j \notin J$:

- $\texttt{comp}\,(I, i, \rho, \varphi, u, a_0)\, f = \texttt{comp}(J, j, \rho(f \cup (i/j)), \varphi f, u(f \cup (i/j)), a_0 f)$ *where* $f \cup (i/j) : J \cup \{j\} \to I \cup \{i\}$ *is just the extension of* f *with the substitution* (i/j) *and* $u(f \cup (i/j))$ *is the partial element defined for* $g : K \to J$ *as* $u_{(f \circ g \cup (i/j))}$.
- $\texttt{comp}(I, i, \rho, 1_{\mathbb{F}}, u, a_0) = u_{(i/1)}$. *In other words, the values of* A *on top of the hypercube* ρ *can be extended to to values on top of the faces of the hypercube* ρ *for which* $i = 1$. *In other words, it is possible to close the "lid" of the box* ρ *along the side* $i = 1$.

Again, it is possible to write this operation with natural deduction as a introduction rule telling the existence of the category theoretical interpretation in the model of a specific term with a specific type. The notation $\mathtt{comp}(I, i, \rho, \varphi, u, a_0)$ is written with a shorter syntax and with application to ρ implicit as $\mathtt{comp}^i\ A\ [\varphi \mapsto u]\ a_0$.

$$\frac{\Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \qquad \Gamma \vdash a_0 : A(i/0)[\varphi \mapsto u(i/0)]}{\Gamma \vdash \mathtt{comp}^i A[\varphi \mapsto u]a_0 : A(i/1)[\varphi \mapsto u(i/1)]}$$

If $\sigma : \Delta \to \Gamma$ is a context substitution, then this morphism acts on terms as follows:

$$\frac{(\mathtt{comp}^i A[\varphi \mapsto u]a_0)\,\sigma}{\mathtt{comp}^j A(\sigma, i/j)[\varphi\sigma \mapsto u(\sigma, i/j)]a_0\sigma}$$

where j does not appear as a dimension variable in a syntactical representation of Δ yet. For types the morphism acts analogously.

The existence of a composition structure cannot be taken for granted. Its existence depends on the choice of context and type.

Definition 3.3.3. *A* Kan type *or* fibrant type *A is a type for which there exists a composition structure* $\mathtt{comp}_A$.

3.3.2 The path type

The use of the identity type in cubical type theory is replaced by the use of a path type. There is also a naming problem because cubical type theory is modelled with presheaf models and the standard model of the identity type is defined as a functor $\mathtt{Id}$: if $\Gamma \vdash A$, $\Gamma \vdash a : A$ and $\Gamma \vdash b : A$, then

$$(Id_A(a, b))\,\rho = \{\star \mid a\rho = b\rho\}$$

However, the type modelled by the functor $\mathtt{Id}$ does not satisfy the properties of the intensional identity type in univalent type theory. It does for example only admit at most one term and satisfies the uniqueness of identity proofs principle (and also ax. 2.5.14). This means it is incompatible with the univalence axiom and insufficient to model univalent type theory.

To solve these problems in cubical type theory and other presheaf models, another interpretation is given to the identity type. This interpretation is done with a functor that models the intensional identity type between terms $a, b : A$ correctly (up to the computatoin rule prop. 1.2.8) is called $\mathtt{Path}_A\,(a, b)$.

Definition 3.3.4. *Given terms* $\Gamma \vdash a, b : A$, *the* path type $\mathtt{Path}_A(a, b)$ *is defined as follows:*

- *Given* $\rho \in \Gamma(I)$, $\mathtt{Path}_A(a, b)(\rho)$ *is the set of equivalence classes generated by pairs* (i, w) *with* $i \notin I$ *and* $w \in A(\rho s_i)$ *such that* $w(i/0) = a(\rho)$ *and* $w(i/1) = b(\rho)$, *where* (i, w) *is identified with* (i', w') *if and only if* $w' = w(i/j)$.
- *The action of* $\mathtt{Path}_A(a, b)$ *on a morphism* $f : (J, \rho, f) \to (I, \rho)$ *is given by* $(i, w)f \equiv (j, w(f \cap (i/j))$ *for* $j \notin J$.

The equivalence class of (i, w) *is also denoted by* $\langle i \rangle\, w$ *where* w *implicitly depends on a choice of* ρ*. Formally, using typing rules and natural deduction:*

$$\frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t : A \qquad \Gamma, i : \mathbb{I} \vdash t(i/0) = a : A \qquad \Gamma, i : \mathbb{I} \vdash t(i/1) = b : A}{\Gamma \vdash \langle i \rangle\, t : \mathtt{Path}_A\,(a, b)}$$

$$\frac{\Gamma \vdash t : \mathtt{Path}_A\,(u_0, u_1)}{\Gamma \vdash t0 = u_0 : A}$$

$$\frac{\Gamma \vdash t : \mathtt{Path}_A\,(u_0, u_1)}{\Gamma \vdash t1 = u_1 : A}$$

To obtain a model for intensional identity type that also satisfies the computation rule of the intensional identity type (see prop. 1.2.8), the identity type of Swan [Swa14] has to be used instead of the path type in def. 3.3.4. The identity type of Swan does exactly have the elimination rule of prop. 1.2.8. However, the difference between the path type in def. 3.3.4 and the identity type of Swan is only minor (up to paths) and will not be covered thoroughly in this text. Both Swan's identity type and the path identity type in def. 3.3.4 are logically equivalent, meaning that they can be substituted by each other. See [Hub16a], p. 114 for a comparison of both types and the more advanced text [Swa18] about a general framework for choosing identity types. In this text, the notation $a =_A b$ or simply $a = b$ will be used for both Swan's identity type (constructed by Swan) and the path identity type between terms $a, b : A$ defined in this section.

The standard term of the path type $\mathtt{Path}_A(a, a)$ between a term $a : A$ and itself is the reflexive term $\mathtt{refl}_A(a)(I, \rho) \equiv (i, a(\rho s_i))$ for some $i \notin I$. It is in [Hub16a], page 90 also denoted by 1_a.

This definition applies to any type, but to have that the path type is Kan type and satisfies all the properties of the intensional identity type in univalent type theory, it is necessary that the original type to which the path is applied has a composition structure or is Kan. If the original type was not Kan, the path would for example not necessarily be transitive.

Example 3.3.5. *The* transitivity *property of equality, in this case the transitivity of the path type, states that if* $\Gamma \vdash a, b, c : A$*,* $p : \mathtt{Path}_A(a, b)$ *and* $q : \mathtt{Path}_A(b, c)$*, there is a term* $r : \mathtt{Path}_A(a, c)$*. If* A *is Kan, then* $\mathtt{Path}_A$ *is transitive.*

Proof. The proof is illustrated in fig. 3.5.

If A is a Kan type, by definition, it has a composition operation. The syntactical object $u = [i = 0 \mapsto a, i = 1 \mapsto qj]$ can be viewed as a term that is only defined on three sides:

- The extent $\varphi = (i = 0) \vee (i = 1)$ defines two sides of the hypercube. On these sides the values of u vary along a second dimension j. This is written as $\Gamma, (i = 0) \vee (i = 1), j : \mathbb{I} \vdash u$.

- On the side $j = 0$ the values of u coincide with the values of the path p. This means that for fixed i, in type notation $\Gamma \vdash (p\ i) : A(j/0)[\varphi \mapsto u(j/0)]$

$$\begin{tikzcd} a \arrow[r, dashed, "(p \cdot q)\ i"] & c \\ a \arrow[u, "\texttt{refl}"] \arrow[r, "p\ i"] & b \arrow[u, "q\ j"'] \end{tikzcd}$$

Figure 3.5: Transitivity can be modelled in cubical type theory by a composition of paths along the edges of a square. Assume that i is the horizontal axis and j the vertical axis, then the composition operation comp gives the closing lid: $p \cdot q$, proving transitivity. Diagram inspired by [CCHM16], section 4.3.

For fixed i, take $a_0 = p\ i$. The composition operation now gives a term that extends u along the remaining side $j = 1$:

$$\texttt{comp}^j\ A\ [(i/0 \mapsto a, (i/1) \mapsto q\ j](p\ i)$$

which can be rewritten using lemma 3.2.9 as

$$\texttt{comp}^j A[(i/0) \mapsto a, (i/1) \mapsto q\ j](p\ i) \equiv z.$$

The term z is of type $A(j/1)[i = 0 \mapsto a, i = 1 \mapsto c]$.

If this term depends on i, the introduction rule of a path can be applied to return a term denoted by $\langle i \rangle\, z$ that has type $\texttt{Path}_A\,(a, c)$. To verify this it is necessary to check that the endpoints of this path $\langle i \rangle\, z$ are indeed a and c.

The endpoints are obtained by applying the context morphisms

$$(i/0), (i/1) : (\Gamma.(j : \mathbb{I})) \to (\Gamma.(j : \mathbb{I}).(i : \mathbb{I}))$$

to the generalized extent judgement

$$(\Gamma.(j : \mathbb{I}).(i : \mathbb{I})) \vdash z : A(j/1)[i = 0 \mapsto a, i = 1 \mapsto c].$$

The application of a context morphism σ is just the application of σ to each judgment contained in the generalized extent, which are by def. 3.2.7 (omitting $(\Gamma.(j : \mathbb{I}).(i : \mathbb{I}))$):

- $() \vdash z : A(j/1)$
- $(i = 0) \vdash z = a : A(j/1)$
- $(i = 1) \vdash z = c : A(j/1)$

These judgments and the introduction rule of the path type in def. 3.3.4 prove that z is the right path, making the transitivity property hold for Path.

□

If A is Kan, it can be proven that the path type $\texttt{Path}_A(a, b)$ not only satisfies transitivity but also all the typing rules of the intensional identity type (as mentioned in def. 1.2.6 or listed on page 35 in [Ort19]) except the computation rule. However, it is not known if the Kan property and the composition structure are necessary for this property of the identity type to hold, and whether the composition operation can be weakened.

In [CM19], the composition operation is weakened to a *weak composition operation*. This allows to develop a type theory in which the mathematical properties of the different cubical type theories can be studied simultaneously, see for example the alternative theory described in section 3.6.2.

$$\begin{array}{ccc} p(0) & \xrightarrow{p} & p(1) \\ {\scriptstyle p(0)}\uparrow & & {\scriptstyle p}\uparrow \\ p(0) & \xrightarrow{p(0)} & p(0) \end{array}$$

Figure 3.6: Take two terms $a, b : A$ and a path $p : \mathtt{Path}_A\,(a, b)$, then the term $p(i/i \wedge j)$ is a two-dimensional path that takes two arguments i, j and returns values based on the interpretation of $i \wedge j$ as $\min(i, j)$. The two-dimensional path $p(i/i \wedge j)$ can be illustrated as a square taking i as a horizontal axis and j as a vertical axis.

3.3.3 The filling operation

The composition operation of a Kan type extends the definition of a partial term from one that defined only on one endpoint of a dimensional variable to one that is also defined on the other endpoint of the dimension variable. It however only gives the lid of the box as described in ex. 3.3.5 which is actually the example on [CHM18], page 10.

Complementary to the composition operation, there is a filling operation that not only "fills" the endpoints but also covers the interior of partial terms that are defined on hypercubes by introducing a new variable dependency in the context. This operation is used in [CHM18] for the definition of higher inductive types. The filling operation makes use of the lattice structure that is defined in def. 3.1.1.

For def. 3.3.6, the lattice structure defined in def. 3.1.1 has to be used. The use of the $\wedge$ lattice combinator is illustrated in fig. 3.6.

Definition 3.3.6. *Let A be a Kan type. The Kan* filling *operation on A is an operation that returns a term. More specifically, set $\mathtt{fill}^i\ A\ [\varphi \mapsto a_0]\ a_0$ to be the term*

$$\mathtt{comp}^j\ A(i/i \wedge j)\ \ [\varphi \mapsto u(i/i \wedge j), (i = 0) \mapsto a_0]\ \ a_0 : A$$

in context $\Gamma.(i, j : \mathbb{I})$.

Using the definition of extents, it is possible to decompose the filling term v into judgments (omitting $\Gamma, i, j : \mathbb{I}$):

- The judgements $() \vdash v : A(i/i \wedge j)$ and $\varphi \vdash v = u(i/i \wedge j) : A(i/i \wedge j)$. This means that the term v is defined on the interior $A(i/i \wedge j)$. Applying the substitution (j/i) gives the usual composition term.
- There is a judgment $(i = 0) \vdash v = a_0 : A(i/i \wedge j)$ that is equivalent with

  $$() \vdash v(i/0) = a_0 : A(i/0).$$

  Now $a_0\rho$ is on the left side of the hypercube $\rho \in \Gamma\,(\{i, j\})$.

The filling is the composition plus something else, also called the interior. But the interior in this context is not the interior in a topological sense because there is no concept of continuity or topology present. In the setting of cubical sets it is possible to only compute with a finite amount of (subsets of) corners of standardized hypercubes.

The existence of Kan types, types with a composition operation is not so straightforward. For each type that is used in a type system and has a valid interpretation in the category with families model, the composition operation has to be defined. Examples of such composition structures for natural numbers and path types can be found in [Hub16a], section 6.4.5, p. 97-98. Once these composition structures are given for all basic types in intensional type theory, including paths, the category with families model is strong enough to model univalent type theory. However, the univalence axiom is an axiom about universes of intensional type theory and universes are also types. So it is necessary a composition structure for universes. This composition structure is defined in [Hub16a] with the help of a `Glue` type which will be introduced in section 3.3.4.

3.3.4 The `Glue` type

Now that the basic types for cubical type theory (omitting the definitions of dependent product and sum types that are given for general presheaf models in [Hub16a], section 1.2) have been modeled by their semantics in cubical sets, the concepts from def. 1.3.1 can be translated to their analogue in cubical type theory. The definition of a term of contractibility in intensional type theory is

$$\mathtt{isContr}\ A = (x : A) \times ((y : A) \to \mathtt{Path}_A\,(x, y))$$

and is a valid expression in cubical type theory because sum types and path types have a valid interpretation in cubical sets.

Similarly, a function $f : T \to A$ is an *equivalence* if and only if the following type is inhabited

$$\mathtt{isEquiv}\ T\ A\ f = (y : A) \to \mathtt{isContr}((x : T) \times \mathtt{Path}_A\,(y, f\ x))$$

and an equivalence in general is a term of the type

$$\mathtt{Equiv}\ T\ A = (f : T \to A) \times \mathtt{isEquiv}\ T\ A\ f.$$

There is a semantic interpretation of the `Glue` type in terms of cubical sets given in [Hub16a] but this will be skipped because it is quite complicated and does not add much intuition for the value of this type in the context of the proof of univalence. The text [Ort19], chapter 5-6, tries to simplify the semantics of the `Glue` type using topoi. The introduction of the `Glue` type is given in [Ort19], page 30.

Definition 3.3.7 (`Glue` introduction rule)**.** *Assume that a (total, not necessarily partial) type $\Gamma \vdash B$ is given together with another type that is partial $\Gamma, \varphi \vdash A$ for some extent φ. If the partial type is equivalent to the total type $\Gamma, \varphi \vdash f : \mathtt{Equiv}\ A\ B$ as far as it is defined, there is a type $\Gamma \vdash \mathtt{Glue}\ [\varphi \mapsto (A, f)]\ B$ that satisfies certain elimination rules.*

This can also be expressed with natural deduction as:

$$\frac{\Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma \vdash B \qquad \Gamma, \varphi \vdash A \qquad \Gamma, \varphi \vdash f : \mathtt{Equiv}\ A\ B}{\Gamma \vdash \mathtt{Glue}\ [\varphi \mapsto (A, f)]\ B}$$

Another way to phrase this is as follows: the formation rule of the `Glue` type tells that given an element of the face lattice $\mathbb{F}$ expressing the sides of a square on which f is defined as an equivalence $A \to B$, there is a bigger type that “glues the sides together”. It is a bigger type because by one of the eliminating typing rules as mentioned on [Hub16a], figure 6.5, the `Glue` type extends the partial type A. More precisely, it says that for $\varphi = 1_{\mathbb{F}}$, there is $\Gamma \vdash [1_{\mathbb{F}} \mapsto (T, f)]\ B = A$. Terms of the `Glue` type come into two forms:

- If it is known whether B has a particular partial type, there is a typing rule giving a `glue` term. More precisely, if t is a term of A, f is the equivalence and $\Gamma \vdash b : B\ [\varphi \mapsto f\ a]$, then there is a term denoted by `glue` $[\varphi \mapsto a]\ b$ of the glue type.

- The `unglue` term functions as an inverse of this term, a computation typing rule. For example, given a `Glue` term c : `Glue` $[\varphi \mapsto (A, f)]\ B$, the term `unglue` c is of type $B\ [\varphi \mapsto f\ c]$.

Example 3.3.8. *Take $\varphi = (i = 0) \vee (i = 1)$. Then the type A is only defined for $i = 0$ or $i = 1$, so there are two sides of the hypercube on top of which the type A and the partial equivalence f are defined. The values of A on top of these sides are called A_0 and A_1. The values of B are computed with substitutions. The partial equivalence f also has values f_0 and f_1. Let* `G` = `Glue` $[i = 0 \mapsto (B(i/0), f_0), i = 1 \mapsto (B(i/1), f_1)]$, *then this can be illustrated:*

$$\begin{tikzcd} A_0 \arrow[r, dashed, "\mathtt{G}"] \arrow[d, "f_0"] & A_1 \arrow[d, "f_1"] \\ B(i/0) \arrow[r] & B(i/1) \end{tikzcd}$$

As it goes for all the types that are introduced in cubical type theory, it is also necessary to introduce a composition operation for the `Glue` type. This is done on [Hub16a], section 6.6.2 and makes the `Glue` type into a Kan type.

The only remaining step to do is to prove that the universe type is also Kan, which is done in [Hub16a], section 6.7.2. Once the composition and glueing operations are defined for the universe types, they are defined for all types. As explained in ex. 3.3.5, this implies that the path type over universes also satisfies transitivity. This means the model described in this section supports all types from intensional type theory, the `Glue`-type and a composition structure:

Definition 3.3.9 ([Hub16a]). *A* cubical type theory *is an extension of intensional type theory (see def. 1.2.10) with the following properties:*

- *All types from intensional type theory, including natural numbers, dependent sum and product types, are modeled with a presheaf model over the cube category def. 3.1.4 by defining them as functors (see [Hub16a], p. 28-30.)*

- *The intensional identity type is modeled by modifying the path type of which the semantics and typing rules are given in section 3.3.2.*

- *There are the following additional objects that are not in intensional type theory or homotopy type theory:*

  - *The objects $\mathbb{F}$ and $\mathbb{I}$ introduced in defs. 3.1.9 and 3.1.10 and the context restriction def. 3.2.2.*

  - *The* `Glue` *type from section 3.3.4 allows the construction of a proof of univalence, see section 3.4.*

- *Every type (excluding $\mathbb{F}, \mathbb{I}$ has a composition, satisfying the typing rules (and semantics) in def. 3.3.2, allowing the intensional identity type to be transitive and cubical type theory be a proper extension of intensional type theory.*

The theory in def. 3.3.9 (in which the univalence axiom is provable) satisfies *canonicity of numerals* (see theorem 3.6.2).

3.4 Proof of univalence

Cubical type theory was invented to give a "computational interpretation of the univalence axiom" which would allow to compute with expressions in univalent type theory. In this section, an overview will be given of historical attempts to this proof and one of the most recent versions of the proof.

3.4.1 Statement of theorem

Remember from ax. 1.3.2 that the *univalence* theorem in cubical type theory (which is an axiom in univalent type theory) states:

Theorem 3.4.1. *Given two types $A, B : \mathcal{U}$ of the same universe, there is an equivalence between the types $A = B$ and $A \simeq B$.*

Because it can be proven in certain models such as [KL12] and cubical type theory as in def. 3.3.9, the axiom can also be called a theorem within a particular model. In this section, the proof of the theorem in [MV19] will be discussed.

3.4.2 History of the proof

In this section the development of the most recent revision of the proof in [MV19], `Cubical.Foundations.Univalence` as of writing this text will be discussed. For a history of the development of the semantics, see section 3.6.1.

The first model in which this theorem became provable was simplicial sets [KL12]. The proof in simplicial sets used the ideas such as the Glue type that were reintroduced in later proofs in cubical sets, see [Hub16a], p. 103-104 and p. 122-123 or [BCH18] for an approach without lattice structure (mentioned in section 3.6.1).

The proof in [Hub16a] was the first one that was formally verified in a type theory, more specifically cubical type theory. The first part of this proof has since been rewritten in [Wei16]. The formalization has been continuously worked on since [CCHM15] and [CCHM16] in [MV19]. Recently, this library code was surveyed in [Mö18a] but recent restructuring and revisions of the files in this library can have made this explanation less up-to-date. In the following sections the different parts of the implementation the proof will be discussed.

3.4.3 Contractibility of equivalence singletons

Theorem 3.4.2 (`unglueIsEquiv`). *Let* $\Gamma, \varphi \vdash f : T \to A$ *be a partial equivalence with extent* φ*. The function*

$$\texttt{unglue} : \texttt{Glue}\ [\varphi \mapsto (T, f)]\ A \to A$$

is an equivalence.

Another proof is [Hub16a], p. 103, Theorem 6.7.2 and uses two extra lemmas to construct to proof the equivalence.

Proof. As mentioned in def. 1.3.1 and ax. 1.3.2, an equivalence is a function such that its fibers (pre-images) are contractible. So, to prove that `unglue` is an equivalence amount to proving that for any b in the co-domain A, the fiber `fiber unglue` b is contractible. In other words, the following proofs have to be given:

- A point x : `fiber unglue` b.
- A proof that given y : `fiber unglue` b, $x = y$.

The point x is constructed by constructing the `glue` term on top of `hcomp` $u\ b : A$ where `u` is a partial eterm. It has to be proven that this term is indeed in the fiber. For the second part, an arbitrary element of the fiber y is taken. Now a path is constructed that is point-wise defined with a partial term `u'` and `glue` terms. The end points of this path are indeed x and y because `u`$'$ extends `u`. There also have to be proofs that the terms on the path are in the fiber (or pre-image of b). These proofs are built with an application of `hfill`, a formalization of the filling operation described in section 3.3.3. □

Now `unglueEquiv` expresses that any partial family of equivalences can be extended to a total one.

The following theorem is an intermediate statement of the univalence axiom that is more directly provable than the traditional statement. It will function as a lemma for proving the more traditional version of the univalence axiom.

Theorem 3.4.3 (`EquivContr`).

$$\forall A : \mathcal{U}, \texttt{isContr}\ \left(\sum\ (T : \mathcal{U})\ \ (T \simeq A)\right).$$

The proof in [Hub16a], p. 104, Corollary 6.7.3 uses an extra lemma.

Proof. It is necessary to give a term $x : \sum\ T : \mathcal{U}\ \ (T \simeq A)$ and a proof that there is an equality between x and any other term y of the same type. Take the trivial equivalence $(A, \texttt{idEquiv})$ for x. It remains to prove that every other term is equal to this one. So take another term

$$\texttt{y} : \Sigma(\texttt{Set}\ell)(\lambda \texttt{T} \to \texttt{T} \simeq \texttt{A}))$$

then it can be shown with `unglueIsEquiv` that er is a path. □

According to [Hub16a], [VAC+13], Theorem 5.8.4, it should follow that the equivalence type forms an identity system. The statement of the univalence theorem is however not immediate.

The path type in cubical type theory can be slightly modified (see section 3.3.2) such that it satisfies the J-eliminator, see prop. 1.2.8. Equivalences also satisfy an eliminator, called *equivalence induction.*

Theorem 3.4.4 (`EquivJ`)**.** *Given a property* $P : (A\ B : \mathcal{U}) \to (e : B \simeq A) \to \mathcal{U}$*, a proof of the base step* $r : (A : \mathcal{U}) \to P\ A\ A\ (\texttt{idEquiv}\ A)$*, then it is possible to produce a proof of the property for any* $A\ B : \mathcal{U}$*:*

$$P\ A\ B\ e.$$

Proof. For the proof, some lemmas are needed:

- The lemma $\texttt{idIsEquiv} : \forall A \in \mathcal{U}, \texttt{isEquiv}\ (\lambda\ x \mapsto x)$ proves that the identity map is is an equivalence. This also an immediate proof of the reflexitivity of equivalences which is formalized in $\texttt{idEquiv} : \forall A : \mathcal{U}, A \simeq A$.

- Applying theorem 3.4.3 to some type B gives a proof of contractibility of a sum type depending on B. More precisely, there is a $(C, C \simeq B)$ with $C : \mathcal{U}$ such that every other C' gives an equality $(C, C \simeq B) = (C', C' \simeq)$. Contractibility is very similar to being an h-prop. The theorem $\texttt{isContr} \to \texttt{isProp}$ which is stated in `Cubical.Foundations.HLevels` converts proofs of contractibility into proofs of being an h-prop using the composition operation. It shows that any terms of this particular sum type over B are connected by a path or equality.

- Assuming $e : A \simeq B$, the above theorem is applied to connect the tuples $(B, \texttt{idEquiv}\ B)$ and (A, e) by an equality in the proof of theorem `contrSinglEquiv`.

- Transport is used in the form of

$$\texttt{subst} : (B : A \to \mathcal{U})\,(p : x = y) \to B\ x \to B\ y.$$

  This theorem is part of a few theorems that are all formalizations of univalent type theoretical transport in defined with the composition operation in `Cubical.Core.Prelude`.

Eventually the J-combinator for equivalences is defined by transporting the base case r over the equality returned by `contrSingleEquiv e)`.

□

This construction of the map `ua` is based on the one defined in [Hub16a], Example 6.6.2, p. 101.

Definition 3.4.5. $\texttt{ua} : \forall\ A\ B : \mathcal{U}, A \simeq B \to A = B$ *is defined with the* `Glue` *type. The definition can be illustrated with a diagram:*

$$\begin{tikzcd} A \arrow[r, dashed, "\texttt{ua}"] \arrow[d, "e"] & B \arrow[d, "\texttt{idEquiv}\ B"] \\ B \arrow[r] & B \end{tikzcd}$$

The implementation of the definition takes a partial equivalence e *and type* A*, a dimension variable or parameter* i *and maps it on a term* `ua e` *of type* `Glue` $[(i = 0) \mapsto (A, e), (i = 1) \mapsto (B,$ `idEquiv` $B)]$ B*. By the elimination rules of the* `Glue` *type given on [Hub16a], fig. 6.5,* `ua e` *is a path with endpoints* A *and* B*.*

`uaIdEquiv` : $\forall A,$ `ua` (`idEquiv` A) = `refl`, where `refl` is the reflexive path or equality. It tells that the reflexive equivalence is mapped onto the reflexive identity by `ua`.

Definition 3.4.6. *An* isomorphism *(or* homotopy equivalence, quasi-inverse *[VAC+13]) is defined in* `Cubical.Foundations.Isomorphism` *as a function* $f : B \to A$ *with a pseudo-inverse* $g : B \to A$ *such that* $f \circ g$ *and* $g \circ f$ *are both homotopic to the identity map.*

The theorem `isoToIsEquiv` is a proof that any isomorphism $f : A \to B$ is an equivalence.

Example 3.4.7. *Let* $A =$ `Bool` *and* $B =$ `Fin` 2*. It is possible to prove that* $A = B$*. This is done using any bijection between* A *and* B *which is also an isomorphism. Such an isomorphism is also an equivalence and applying the map* `ua` *returns path of type* $A = B$*.*

3.4.4 Conclusion of the proof

Using the previous lemmas and results, the statement and proof of the univalence theorem is now much easier. `Cubical.Foundations.Univalence`.

Theorem 3.4.8 (`Univalence.thm`)**.** *Given a function*

$$\texttt{au} : A = B \to A \simeq B$$

and a proof that `au` *maps the* `refl` : $A = A$ *identity type constructor onto the constant equivalence* $A \to A$*, the theorem* `Univalence.thm` *states that* `au` *is an equivalence.*

Proof. First it is shown that the given map `au` has to be an isomorphism with as inverse the previously constructed map `ua`. An isomorphism is the homotopy type theoretic variant of a homotopy equivalence or more precisely, a bi-invertable map as in chapter 4 of [VAC+13]. To prove that `au` is a isomorphism, it is necessary to prove the left-inverse and right-inverse properties (up to a path). Both proofs use that `compPath` is just the transitivity property of "=" and congruence.

- The left-inverse is proven with equivalence induction `EquivJ` on the property that `au` is a left-inverse for equivalences `e: au (ua e) ≡ e)`.
- The right-inverse is proven with equality induction `J` on the property that `ua` is a right-inverse for paths `p: ua (au p) ≡ p`.

Although an isomorphism is not well-behaved (it is for example not a proposition), it contains more information than equivalence and from an isomorphism, an equivalences can be extracted. Applying `isoToIsEquiv` gives that `ua` : $A = B \to A \simeq B$ is an equivalence or `isEquiv ua`. □

The theorem of univalence is then a simple consequence of an application of `Univalence.thm`

Theorem 3.4.9 (`univalence`). *Given any $A, B : \mathcal{U}$, there is an equivalence $(A \equiv B) \simeq (A \simeq B)$.*

Proof. The function `lineToEquiv` : $A = B \to A \equiv B$ that maps paths directly on an equivalence serves as a candidate for the inverse `au` in `Univalence.thm`. The necessary condition that is still needed to apply `Univalence.thm` is the fact that `lineToEquiv` maps the reflexive equality onto the trivial equivalence. This follows by proving that the underlying maps of the image under `lineToEquiv`, `pathToEquiv refl` and `idEquiv A` are equal with transport and using the fact that the property of being an equivalence is a proposition, which means there can only be one such proof, such that the the equivalences also have to be equal and `pathToEquiv refl ≡ idEquiv A`. The proof is formalized in `pathToEquivRefl`. Applying `Univalence.thm` to the map and this fact, gives a proof that `pathToEquiv` is an equivalence. Assembling this into an equivalence gives a term of the equivalence type `(A ≡ B) ≃ (A ≃ B)`, which is a proof of univalence. □

3.4.5 Univalence with topoi

The *language of toposes* can be used to discover more fundamental models of cubical type theory and univalence [Ort19]. A topos is a category that behaves like the category of sets. Examples of topoi are the category of sets and presheaves.

First, the cubical type theory as presented in [Hub16a] and this text is translated in the language of toposes [Ort19], section 5.3. Some definitions become much simpler such as the De Morgan algebra used for the base category and the face lattice that were presented in def. 3.1.9 become [Ort19], figure 5.4. This results in an alternative proof of univalence that can be more insightful [Ort19], section 5.5. An overview of related foundational work can be found in [Pit18] and [LOPS18].

3.5 Applications of the proof of univalence

In section 3.4, it was shown that the univalence axiom holds in cubical type theory. The theorem of univalence allows to use all the concepts from univalent type theory, also called homotopy type theory, in cubical type theory. Cubical type theory was implemented in `Cubical.Core.HoTT-UF` of [MV19].

In combination with the standard library of Agda [DDG+19], it is possible to produce concrete examples in univalent type theory of reasoning with univalence.

3.5.1 Isomorphism invariant algebra

In the theories of groups and other algebraic structures, proofs are preferably done for only one structure in an isomorphism class. The proofs for other structures in the same class are not considered valuable because they do not add information. With the univalence axiom, it is possible to treat these different proofs as equal.

This can be shown in the context of monoids (sets with a binary operation and neutral element) for the following two concrete examples:

$$s_1 \equiv (\mathbb{N}, (m, n) \mapsto m + n, 0)$$

```
op₂ : Op₂ ℕ₀
op₂ (x , p)  (y , q) =
    (predℕ (x + y) , (predℕ (predℕ (x + y)) , sumLem x y p q) )

s₂ : myMagma _ _
s₂ = record {
  Carrier = ℕ₀ ;
  _◇_ = op₂ ;
  s = transport (λ i → isSet (fEq i)) isSetℕ
  }
```

Figure 3.7: The implementation of the definition of s_2 in agda works by giving a term for the record type magmas. Such a term, called a record, takes all the components of a magma such as the carrier set, operator and a proof that the carrier set is an h-set (see def. 1.2.11) and groups them in one object.

and

$$s_2 \equiv (\mathbb{N}_0, (m, n) \mapsto m + n - 1, 1)$$

from [CD13] and [Dan12].

The goal of this example is to create a term of the path type between s_1 and s_2 in `Mon` and transport proven properties of s_1 from the the type of properties on s_1 to the one on s_2 without repeating the proof for s_2. For example, if there is a proof of the fact that s_1 is commutative, there is also a proof of the fact that s_2 is commutative. This shows one application for the proof of univalence: the possibility of doing proofs up to isomorphism.

The file `Algebra.Structures` in the standard library [DDG+19] contains definitions for monoids and other algebraic structures as record types. Because of the number of laws that need to be supplied to define a monoid, the example was reduced to a custom definition of magmas together with a proof that the carrier type is an h-set (defined in def. 1.2.11), see for an example fig. 3.7.

A path between s_1 and s_2 coincides at time i with a "point-wise defined" algebraic structure, a magma. So defining such path is equivalent with defining a magma at every time i that coincides with the given s_1 and s_2 on its endpoints. To understand how this works, look at the definition of magmas as record types. Record types are nested dependent sums and a path between two terms of a dependent sum is composed of paths between the components of the dependent sum. So for each component of the record type there should be path, including the carrier sets and operators on both structures.

It can be proven using the map $f : n \mapsto n+1$ that there is an equivalence between $\mathbb{N}$ and $\mathbb{N}_0$ which gives by the univalence axiom (proven to hold in def. 3.3.9 by section 3.4) a path `ℕ ≡ ℕ₀`. This gives a path between one component in the term of the record types of magmas.

Another component is the operator. In this case, the operator is actually a path between the two given operators. So it is necessary a point-wise defined operator that coincides with the respective operators on its ending points. The construction is done by transporting the arguments of the operator from intermediate carrier set `fEq i` to the carrier set `fEq i0` which is equal to $\mathbb{N}$ and applying `op₁` there, see fig. 3.8.

```
zeroPath : (i : I) → (fEq i) ≡ (fEq i0)
zeroPath i = λ j → fEq (i ∧ (~ j))
...
opᵢ' : PathP (λ i → Op₂ (fEq i)) op₁ op₂'
opᵢ' i x y = transport (sym (zeroPath i))
    (op₁ (transport (zeroPath i) x) (transport (zeroPath i) y))
```

Figure 3.8: This definition of the intermediate operator is done by transporting the arguments along `zeroPath` back to `fEq i0`, applying `op₁`. The result is then translated back forward to `fEq i`.

The operator `opᵢ'` from fig. 3.8 does not have the right operators at its ending points however and it is necessary to define another operator, see fig. 3.9.

The rest of the definition of the intermediate point-wise defined algebraic structure is straightforward. Once a term `algPath` of the equality `s₁ ≡ s₂` (see fig. 3.10) between both magmas is obtained by assembling all point-wise constructs in a record type, algebraic properties of magmas can be transported.

For example, such a transport of properties could consist of mapping a term of the property that `s₁` is commutative to a term of the property that `s₂` is commutative, see fig. 3.11.

At this point it can be concluded that if there is a homotopy type equivalences (which is simply a bijection for discrete set-like types or h-sets, see def. 1.2.11) between the carrier types of two terms of an algebraic type that have "compatible operations", there is an equality between the terms. In this case, the equality is a term `algPath` of type `s₁ ≡ s₂`. The compatibility of the operations means that the algebraic operations can be connected by a path but actually corresponds to the homotopy type equivalence being a morphism in category theory.

An application of the path `s₁ ≡ s₂` between the two magmas, is that the proof of commutativity of the first magma `s₁` can be transported to a proof of the commutativity of the second magma `s₂`, defined as `com₂ = transport (λ i → isCommutative (algPath i)) com₁`. Although the property of being commutative is very simple, it can be generalized to more complicated properties and algebraic structures. This shows that it can be possible to do "isomorphism-invariant" algebra in practice. The complete code of this implementation can be found on [Van19] and verified with the installation instructions.

However, due to the numerous trivial lemmas that have to be implemented and the unusual homotopy theory inspired approach, it is not the easiest approach for proving algebraic properties. Definitional identities that in traditional mathematics simply hold by simple computation do not only hold up to some path in univalent type theory and cubical type theory. This means that simple facts about natural numbers are not simple anymore. However, not all structures in mathematics are as well-known as natural numbers and the structural and overly rigorous approach that such type theories require can be more beneficial in a general setting At least in this case of simple traditional algebra, it seems easier to use specialized software such as [HBJ+18] to carry out algebraic computations and proofs.

```
startLemma : op₁ ≡ op₁'
startLemma i = λ x y → x + (transportRefl y) i

endLemma : op₂' ≡ op₂
endLemma i x y =
  (op₂' x y
      ≡⟨ refl ⟩
   transport fEq (op₁ (transport (sym fEq) x)
                      (transport (sym fEq) y))
      ≡⟨ transpR
           (op₁ (transport (sym fEq) x)
                (transport (sym fEq) y)) ⟩
   f (op₁ (transport (sym fEq) x)
          (transport (sym fEq) y))
      ≡⟨ doubleCong (λ x y → f (op₁ x y))
           (transpL x) (transpL y) ⟩
   f (op₁ (g x) (g  y))
      ≡⟨ cong f (l₂' x y) ⟩
   f (predℕ ( predℕ ((fst x) + (fst y))))
      ≡⟨ refl ⟩
   (suc (predℕ ( predℕ ((fst x) + (fst y)))) ,
      (predℕ ( predℕ ((fst x) + (fst y)))) , refl )
      ≡⟨ lₑ x y ⟩
   (predℕ (fst x + fst y) ,
     (predℕ (predℕ (fst x + fst y)) ,
      sumLem (fst x) (fst y) (snd x) (snd y)))
      ≡⟨ refl ⟩
   op₂ x y ∎) i

pathLemma : (PathP (λ i → Op₂ (fEq i)) op₁' op₂')  ≡
             PathP ((λ i → Op₂ (fEq i))) op₁ op₂
pathLemma = cong₂ (PathP (λ i → Op₂ (fEq i))) startLemma endLemma

opᵢ : PathP (λ i → Op₂ (fEq i)) op₁ op₂
opᵢ = transport pathLemma opᵢ'
```

Figure 3.9: This code shows a lemma that proves that the endpoints of the original intermediate operator `opᵢ'` are (path) equal to `op₁` and `op₂`. Using these proofs and transport, a new intermediate operator `opᵢ` is defined that does satisfy the requirements for the intermediate operator of the intermediate magma.

```
isSetᵢ : (i : I) → isSet (fEq i)
isSetᵢ i = {!transport  (λ j → isSet (fEq (i ∧ j))) isSetℕ!}

algPath : s₁ ≡ s₂
algPath = λ i → record {
   Carrier = (fEq i) ;
   _◇_ = opᵢ i ;
   s = isSetᵢ i
   }
```

Figure 3.10: A term of the intensional identity type between the two algebraic structures is a structure depending on a parameter `i` that has components depending on the parameter `i`. Each component was point-wise defined before and can be assembled into a point-wise structure.

```
isCommutative : myMagma → Set _
isCommutative m =
    (x y : (Carrier m)) → (x m._◇_ y) ≡ y m._◇_ x

com₁ : isCommutative s₁
com₁ = +-comm

com₂ : isCommutative s₂
com₂ = transport (λ i → isCommutative (algPath i)) com₁
```

Figure 3.11: Here a magma is defined to be commutative if it's operation is commutative on the carrier type. Commutativity for natural numbers has already been proven in the standard library. This proof can be transported along the term of equality $s_1 \equiv s_2$ to give a proof of commutativity of s_2.

N.	Name	Day	Month	Year
4	John	30	5	1956
...	...	...	...	...
42	Sun	20	3	1980

N.	Name	Month	Day	Year
4	John	5	30	1956
...	...	...	...	...
42	Sun	3	20	1980

Table 3.1: The European and American databases on [Lic13b], p. 47.

3.5.2 Generic datatypes

Apart from mathematical applications, there are also more practical applications. Datatypes in mainstream programming languages such as Java can be parametrized by parameters. For example, a list can be parametrized by the type of it's contents. The resulting datatype is then called generic, the approach is called *generic programming*. There might be multiple ways of implementing a datatype and several implementations might be equivalent. This is were the univalence axiom comes into play.

Assume two date representations are given: "(month, day)" and "(day, month)". These representations are equivalent by the swapping map. Univalence gives a path between the equivalent representations. Assume databases are generically defined for a certain date representation. For example, the type of databases is defined as `List` $(\mathbb{N} \times$ `String` $\times (X \times \mathbb{N}))$ where $X = \mathbb{N} \times \mathbb{N}$ is any type of date representation. The goal is to prove that the databases given in table 3.1 are equal.

The path between representations is lifted to a path on the level of the type of databases. The lifted path becomes a path between two different databases using two different but equivalent representations. Because paths behave like equality, the two databases are equal. The example has been implemented in the file `Cubical.Experiments.Generic` of [MV19].

This simple example shows that the implementation of univalence in cubical type theory allows a programmer to treat generic datatypes as the same datatype up to equivalence of the parameter type.

3.5.3 Formalizing algebraic topology

Higher inductive types

Higher inductive types are a concept coming from [VAC$^+$13]. A *higher inductive type* is a type that has formation and construction rules as other types but these rules are accompanied by additional typing rules stating how to produce terms of the path types between its terms. In other words, the type is generated by constructors and equalities. Higher inductive types are useful in synthetic homotopy type theory.

The first attempts at using higher inductive types in the cubical presheaf model date from [LB15], [Hub16a] and [CCHM16] but it was only until recently that the semantics were worked out in [LS17] and [CHM18].

add information of implementations of higher inductive types

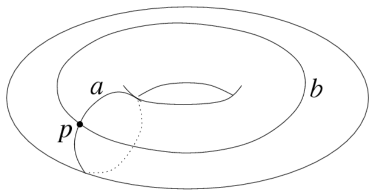

Figure 3.12: A topological representation of the torus [Din05].

Example 3.5.1. *The* circle S^1 *as a topological object is defined as line of which the endpoints are identified. In the language of higher inductive types this becomes two constructors:* `base` : S^1, `loop` : `base` = `base`.

Example 3.5.2. *The* torus *is defined similarly but now there is an extra loop which is introduced with an extra constructor. In addition, both loops intersect, meaning that doing one loop is topologically equivalent to taking the other loop. This is written as a path that starts and ends at the second loop:* `square` : `PathP` $(\lambda\ i \to$ `a` $i =$ `a` $i)$ b b. *See fig. 3.12. Another way to construct the torus is given in [VAC+13], section 6.6.*

Algebraic topology

Based *loop spaces* in algebraic topology contain the loops in a toplogical space that start and end at a particular point. Because there can be many loops, the spaces become very big. It is possible to add an operation to a loop space that composes two loops, called concatenation. This type of space can be generalized to higher-dimensional based loop spaces by iterating the loop space construction. These higher-dimensional loop spaces can be interpreted as spaces that contain loops between loops and also have a concatenation operation. Loop spaces with such an operation in a topological space X are denoted by $\Omega_i(X)$ (omitting the base point) but the operation does not have to be well-behaved. This is were homotopy comes into play. Loops can be considered up-to path-homotopy. This means in pratice that an equivalence relation is defined on the loop space, identifiying the loops that can be deformed without tearing into eachother. The resulting space with this equivalence relation based on continuous deformations retains the concatenation operation but is more well-behaved. The concatenation operation is invertable and also commutative in higher-dimensional loop spaces with such an equivalence relation, a result that is called the Eckmann-Hilton theorem (see [VAC+13], 2.1.6. Because the concatenation is invertable, the loop spaces form groups for this operation, called *homotopy groups*. Given a topological space X, this is denoted by $\pi_i(X)$ where the base point is often omitted for path-connected topological spaces. Homotopy groups are the same for homotopy equivalent spaces, they are homotopy invariants. The first homotopy group is called the fundamental group. For more information on general algebraic topology, see the text [Hat01].

As mentioned in section 2.2, types in univalent type theory behave like topological spaces. Equalities in univalent type theory correspond to paths and this intepretation can be extend to more concepts of algebraic topology. Loop spaces can be defined and studied in univalent type theory in the same way they are defined in algebraic topology.

Example 3.5.3. *The first loop space of* S^1 *is formalized in* `Cubical.HITs.S1.Base` *as* $\Omega(S^1) \equiv$ (`base` = `base`) *for a fixed* `base`. *So this type consists of all the identifications* *of the base point with itself through a term of the path type. In the context of the univalence axiom, this path type can have multiple terms that represent paths. So the terms correspond to closed paths which are loops in S^1 as a topological space. The loops are generated by iterating the reflexive* `loop` *from the definition of S^1 as a higher inductive type. Because there are $\mathbb{Z}$ ways of iterating this loop, it can be shown that $\Omega_1(S^1) \simeq \mathbb{Z}$.*

General *loop spaces* of types are defined by iterating the Ω construction. In formal homotopy type theory, general loop spaces are denoted by $\Omega_i(X)$. However, these type-theoretic loop spaces can be turned into homotopy groups $\pi_i(X)$ using a truncation operation. The resulting homotopy spaces are quite complicated to compute for high values of i. The proofs use the Hopf fibration and long exact sequences. Many cases were already proven in homotopy type theory in [VAC+13], Chapter 8. See also [Lic13a] for a survey of most results that have been proven in homotopy type theory and the methods used. One important result that was proven is:

Theorem 3.5.4 ([VAC+13], th. 8.10.1)**.** *There exists a k such that for all $n \geq 3$, $\pi_{n+1}(S^n) = \mathbb{Z}_k$.*

A special case is the following theorem from homotopy type theory:

Theorem 3.5.5 ([Bru16])**.** $\Omega_4(S^3) \simeq \mathbb{Z}/\mathbb{Z}_2$

In the presence of the univalence axiom, the equivalence is also an equality. The proof of this theorem uses advanced concepts such as the Blakers-Massey theorem, the James construction and Gysin exact sequences. See [Bru16] for the complete proof.

A summary of the concepts used in this constructive proof compared to the proof in classical algebraic topology is given on [Bru16], page 123. The author mentions in the conclusion that cubical concepts are not always perfectly suited for this proof.

Nevertheless, there have been attempts at implementing the proof in cubical type theory based on the sketch in [Bru16], Appendix B. The first implementation was in the module `Examples.Brunerie` of the library [CCHM15] but is very long. The latest version is implemented in Agda and can be found in the file `Cubical.Experiments.Brunerie` of [MV19].

In these implementations, the definition of n in the expression $\Omega_4(S^3) \simeq \mathbb{Z}/n\mathbb{Z}$ is formalized as an Agda expression called `brunerie` or the "Brunerie number" that makes use of the univalence axiom. The expression is the application of a composition of several complicated functions to a representation of S^3 as a higher inductive type. Some of these functions use the concepts necessary for the proof of univalence such as gluing but do not reference the theorem of univalence explicitly. There are signs that the proof of canonicity in cubical type theory (see theorem 3.6.2) can help in computing or reducing `brunerie`. Unfortunately the expression `brunerie` does not reduce to the number 2 after hours of computation time. The process of reducing was sped up but the length of the shortest reduced form of `brunerie` was only reduced from a length in characters of order 10^6 to 10^5 when rewriting the original formalized proof [CCHM15] in [MV19], according to [Bru18].

Recently, there has been a lot of ongoing activity in the development of the Agda Cubical library and bufixes can have eliminated some reasons for the lack of normalization of `brunerie` but it still does not normalize. Although this example still has problems, it shows how univalence could be used to formalize proofs of algebraic topology and verify their correctness, at least for relatively low order homotopy groups.

3.5.4 Future applications

Until now, most applications of cubical type theory are related to implementing simple inductive types and higher inductive types. Most mathematical concepts from univalent type theory [VAC+13], part 2 are built with higher inductive types. So it should be possible to formalize more advanced concepts in cubical type theory. Concepts from univalent type theory that can be useful and have not been implemented with the help of cubical type theory and [MV19] include:

- The *structure identity principle* defined in [VAC+13], theorem 9.8.2 and [Acz12] states that isomorphic structures are equal. As a consequence proofs can be transferred or recycled between isomorphic structures. This principle could also be seen as a generalization of the simple example in section 3.5.1 to any category. The principle has not been formalized in all generality yet in cubical type theory but if this was done, it would be much easier for a practicing mathematician to identify isomorphic structures, one of the "main benefits" of homotopy type theory. Right now, software such as [HBJ+18] has to be used to reason about isomorphic algebraic structures.

- The application in section 3.5.3 needs the use of the univalence axiom. The theory of algebraic topology has some other theorems that can be formulated in homotopy type theory and have proofs using univalence such as Freudenthal, Blakers-Massey and Seifert-Van Kampen. These theorems can be used to compute homotopy groups in algebraic topology by putting information about smaller parts of a space together. The proofs have been formulated in the programming language Agda as a library [HBC+18] based on the theorems that were more informally presented in [VAC+13], chapter 8. For now, section 3.5.3 shows that computations do not succeed for the fourth fundamental group of S^3, but it can be possible that computations of other homotopy groups of topological spaces are more efficient when [HBC+18] is combined with an interface to cubical type theory, for example through [MV19].

- Next to homotopy groups, there are also homology groups of topological spaces. For a fixed topological space, these do not necessarily have to be the same as the homotopy groups. Recently, synthetic homology groups have been studied in homotopy theory with filling operations of cubical type theory in [Gra18a], based on the initial approach to synthetic homotopy theory in [LB15]. The results in [Gra18a] have not been completely formalized in cubical type theory because the reasoning with higher-dimensional paths in cubical type theory posed some challenges. There exist computational systems for computing homology (and homotopy) groups that do not make use of type theory such as [GSS+99] and it might be interesting to check how [Gra18a] improves on this. Type theory can be better suited at solving complex new problems in computational topology than just trying to do this with the algorithms in [GSS+99].

A list of open problems for univalent type theory is on [SRvD+19] and contains subjects such as semantics, some of which have been solved with cubical type theory. There are also other open problems that remain relevant for future improvements on cubical type theory because cubical type theory is an extension of univalent type theory.

3.6 Alternative cubical models of univalence

The proof of univalence described in section 3.4 was not established in one paper but it took a while until the right syntax and semantics was found to be able to speak of cubical type theory as in def. 3.3.9.

3.6.1 Historical development of cubical set models

The first time dimension variables or cubes were used in type theory was in [BM12]. The theory of *nominal sets* which is introduced in [Pit13] introduced a way of working with dimension variables as objects in a category. In [BCH14], this idea was extended to a categorical model of type theory in cubical sets together with Kan extension properties. The way nominal sets were used in [BCH14] was discussed in [Pit14]. Cubical sets were however already discussed in [GM03]. The model was extended in [Hub15] to include more detailed proofs and semantics of the universe. In [Coq13a] it was suggested how this model could be allow a proof of the univalence axiom. By that time there was already an implementation of [Hub15] available in [CCHM15]. The proof of univalence with this model was given in [BCH18] but slightly more complicated than in later models and higher inductive tpes could not yet be modelled.

This earlier model was however valuable because it proved the consistency of cubical type theory relative to the framework of category theory in cubical sets. It also did not use choice principles which made it the first constructive model of the univalence axiom and univalent type theory. Models of type theory in type theory that do not use any extra structure on the base category are called *monoidal* or sub-structural because they can be represented using only products of an interval object.

The *lattice structure* in def. 3.1.1 was the first step in allowing higher inductive types and simplifying operations. It was was historically later introduced by [CCHM16] and slightly corrected in [Hub16a] for simplifying composition and using higher inductive types. There is still research going on on whether the lattice structure is necessary in the base category and other models such as [Alt17] do not make used of it. There is an unpublished comparative study in [Awo16] on choices of base categories that are related to the base categories used in [BCH14] and [Hub16a]. Each choice of a variation of the base category can lead to more desirable properties of the type theory being modelled but no better alternative has been found yet (a choice that results in a more efficient cubical type theory). For example, [PK19] uses twisted cubes to model directed (cubical) type theory.

The latest version of cubical type theory that is implemented in proof assistents has slightly different operation thant described in [Hub16a]. To allow working with higher inductive types, [Hub17b] suggested to decompose the composition operation in a homogeneous composition and a generalized transport. This idea was worked out in [CHM18]. Since then, no big changes were made to the semantics but there is still research on models that can unify the semantics of cubical type theory and variations such as two-level (cubical) type theory in [Uem19] and [CM19].

3.6.2 Cartesian computational type theory

There a completely different approach taken to cubical type theory and proving the univalence axiom in Pittsburgh. This version does have a different syntax and does not (directly) make use of categorical semantics but also admits a proof of univalence. The Pittsburgh approach is called "cartesian cubical computational type theory" and discussed in [ABC^{+}17], [AHH18]. This version of cubical type theory is based on an extensional variant of intuitionistic type theory and has an identity type, denoted by "$\doteq$" that is more like "equality by definition". The identity "$\doteq$" is called the extensional identity type and satisfies the uniqueness of identity proof principle opposed to the intensional identity type def. 1.2.6 and the path type of cubical type theory. Another path identity type is introduced on top of the theory that is similar to the path type in prop. 1.2.8.

Computational type theory

This is also the approach taken by NuPRL and computational type theory on which cartesian cubical computational type theory is based. Using the extensional identity type, these theories extend type theory with quotients of types, set comprehension types, partial recursive function types, intersection types and other special types [Con11], p. 6. Computational type theory has been implemented in the proof assistant NuPRL and the underlying formal theory was described using partial equivalence relations (PERs) in [All87] according to [AHH18] which was later generalized to a set-theoretic semantical model in [Har91]. A historical overview of the relation between computational type theory and def. 1.2.10 is given in [Con03] and recently [Con15]. A computational type theory is usually built as follows:

- First, there is a definition of a set of "closed terms" which are like the terms of the intensional type theories but they are untyped, in the sense that the introduction of types comes later. If M and N are closed terms, then new closed terms are formed by constructors such as

$$\lambda x.M, M(N), \mathtt{fst}(M), \mathtt{snd}(M), 0, \mathtt{M}, ...$$

- There are operational semantics defined on the closed terms to specify how they compute. Operational semantics describe how evaluation of terms influences evaluation of other terms, evaluation of the term M to M_0 is denoted by $M \mapsto M_0$. The expression $M \Downarrow M'$ denotes that both terms M and M' reduce to the same normal form. Examples of operational semantics are given on [Har91], p. 75.

- A type A is introduced as a partial equivalence (transitive and symmetric), called a *PER* relation denoted by $[\![A]\!]$ over the terms. Then the term a is of type A if and only if $[\![A]\!](a, a)$, also denoted $a \in A$. In that case a is called a *canonical value* of type A. So types are defined by all the closed terms that are reflexive according to a particular PER.

- Every type A has an identity for terms a, b defined by setting $a \doteq b$ if and only if $[\![A]\!](a, b)$. In this case, a, b are said to have *equal canonical values.*

Cubical extensions

Cartesian cubical computational type theory is based on results in the technical report [AHH17] and [AHH18]. The more informal text [Ben18] gives an introduction in the style of [VAC+13]. The base category of cartesian cubical computational type theory not explicitly mentioned because dimension variables are not considered as independent objects. This implies the theory is not proven to be consistent in a presheaf model which was done for the cubical type theory introduced in earlier chapters of this text in [Hub16a], part 2. Instead, dimension names such as x, y, z are treated as variables that can be substituted by *dimension terms* r, where r is either an endpoint of the unit interval (0 or 1) or a dimension name. This makes substitutions possible of the form (x/y) that can be considered as diagonals in a square. This can also be seen as adding terms of the form $\varphi \equiv (x = y)$ to the face type $\mathbb{F}$ of cubical type theory [Hub16a]. This is where the name "cartesian" comes from. Dimension names are grouped in *dimension contexts* Ψ and types are PERs that are indexed by dimension contexts. Opposed to the theory in previous chapters (based on [Hub16a]), there is no lattice structure on the dimension variables occuring in a dimension context Ψ, which means that substitutions $A(i/i \wedge j)$ do not make sense anymore.

Operations from cubical type theory that are crucial for applications use the lattice structure such as the filling operation given in def. 3.3.6. According to [Mö18c], the solution is to give a different composition operation, a stronger one. This operation is defined in [AHH18], section 4 or [Mö18c], p.18. This new composition operation generalizes the filling and composition operation. It is however split up into two operations: homogeneous composition `hcomp` and coercion `coe`. According to [Mö18c], this splitting, makes certain implementations shorter.

Univalence

There is also an intensional path equality type defined in [AHH18] as is done in [Hub16a]. To turn equivalences between types of some universe into paths between those types and prove the univalence axiom, there are two possibilities in cartesian cubical computational type theory:

- Introduce `Glue` types as in section 3.4, after [Hub16a]. This was also taken in a previous version of cartesian cubical computational type theory [ABC+17].

- In [AHH18], the `Glue` type is weakened to a type called `V`-type with simple typing rules. On top of the `V`-type, instances of the `hcomp` and `coe` composition operations can be defined.

Together with the extensional equality that comes with all computational type theories and the proof of univalence, this turns cartesian cubical computational type theory into the first *two-level* homotopy type theory. Other such two-level type theories did not, but this theory satisfies *canonicity of booleans* according to [AHH18]. This canonicity is however equivalent to canonicity of numerals (see theorem 3.6.2). The theory was also extended to include definitions for higher inductive types in [CH18] and [CH19]. The biggest advantage of this theory seems according to [Mö18c] to be that the proof of univalence with `V`-types can be more efficient and allow an easier computation of the Brunerie number that was introduced in section 3.5.3.

Implementations

It was implemented in Haskell [MA18] and in the language RedPRL [SHA+18] with a proof of univalence that is slightly longer than the one in [MV19]. Development was continued in RedTT [SHC+18] which has many similarities but also extra features such as extension types according to [Mö18a] but stopped by the end of 2018. The library code in [SHC+18] also contains an alternative proof of univalence. The syntax is different from [MV19] but the structure of the libraries is very similar. The composition and composition operations used in [SHC+18] have also been implemented in [MV19] in the file `Foundations.CartesianKanOps`.

3.6.3 Open challenges

Strongly normalizing

Type theories are very expressive formal theories. It is important to know whether a type theory exhibits nice behaviors when implemented. The type theory in def. 1.2.10, which was described in [Ml75] is a "nice" theory because it is *strongly normalizing*: there is no infinite sequence of reductions. This implies *decidability of type checking*, the procedure of checking whether a term does inhabit a certain type. The property of being strongly normalizing has not been proven at all for cubical type theory. It is noted in [CCHM16] that this could be done with resizing rules and in [Hub16a] it is conjectured that the proof of canonicity could be somehow adapted, see theorem 3.6.2. The method described of computation and evaluation in [Abe13] can be useful.

The *confluency* of terms means that two sequences of reductions starting from the same term can be made to converge again. It is mentioned in the context of homotopy type theory in [HC11]. Confluency always holds in a strongly normalizing type theory but has also not been proven yet.

Canonicity

A weaker property of type theory than strongly normalizing is the property of *canonicity*. It says that every term can be written (uniquely) as a reduced form, containing only a finite amount of applications of the constructors of it's type. Canonicity does not imply strongly normalizing and is usually stated for the types of natural numbers or booleans. Canonicity for general types is never proven because it does not even hold for function types in most type theories.

Canonicity for homotopy theory was conjectured by Voevodsky according to [Bru18]:

Conjecture 3.6.1 (Homotopy canonicity). *Given a term $t : \mathbb{N}$ constructed using the univalence axiom, it is possible to construct two terms $u : \mathbb{N}$ and $p : t =_{\mathbb{N}} u$ (that can involve univalence) such that u does not involve the univalence axiom.*

This conjecture can fail to hold when axioms such as function extensionality or univalence are added the theory and as a consequence, it was not solved for homotopy type theory but was solved for its extension, cubical type theory, in [Hub16a] and published in [Hub17a]. The proof was improved by removing a necessary condition for path liftings in [CHS19].

This makes cubical type theory into an example of a theory in which both canonicity and univalence holds. The statement of canonicity for cubical type theory is slightly different from the original conjecture (see [Hub16a], p. 125):

Theorem 3.6.2 (Cubical canonicity)**.** *given a context I of the form $i_1, ..., i_k : \mathbb{I}$ and a derivation of $I \vdash u : \mathbb{N}$ (where $\mathbb{N}$ is an inductive type), there is a unique $n \in \mathbb{N}$ (in a metatheory) with $I \vdash u = \mathtt{suc}^n\ \mathtt{zero} : \mathbb{N}$ for some $n \in \mathbb{N}$. This n can more-over be effectively calculated (in a metatheory).*

There is also another version of canonicity in literature: "canonicity of booleans" (see [AHH18] or section 3.6.2). But because numeral values give rise to boolean expressions and the other way around, canonicity of booleans is equivalent with canonicity of numerals. When cubical canonicity holds, at least every basic type (the numerals and similar types) has a standard form that can be computed but in practice computing this standard form does not always work.

For example, in section 3.5.3, the proof of canonicity tells that there is a term

$$\mathtt{Id}_{\mathbb{N}}(\mathtt{brunerie}, \mathtt{suc}^n\mathtt{zero})$$

and that n can be computed to be 2. However, the computation of the expression `brunerie` does not result (quickly enough) in a numeral that uses only a finite number of applications of `suc`. This can be because the process of reducing a term of the $\mathbb{N}$ type is not the most effective way of computing the number n of the standard reduced form and the most effective implementation for the effective procedure for computing the number 2 in the metatheory (or internally) is not given explicitly in [CHS19]. It could be a future challenge to implement this procedure.

There seem to be problems with the current implementations of `Glue` types and there is ongoing work on trying to replace them with more efficient alternatives such as `V`-types (see section 3.6.2) but cubes may not be a perfect solution after all according to [Mö18b].

Conclusion

This text gave an overview of the motivations and concepts behind the cubical set model of an extension univalent type theory in section 2.4 and the type theory called cubical type theory that is modelled by with cubical sets in chapter 3. Using the properties of the model, more precisely the `Glue` construction, it is possible to encode a proof of the univalence axiom in cubical type theory, see section 3.4. The application of the proof of univalence in cubical type theory to isomorphic algebraic structures in section 3.5.1 was not immediate but allowed to study transport of properties of structures. In standard univalent type theory it would not be possible to compute the transports of such properties but with the help of cubical type theory this became possible. Studying these formalized proofs in cubical type theory can give a (better) topological intuition about using the structure identity principle (see [Acz12]) in computer assisted proofs. Having this intuition becomes very useful with the implementation of cubical type theory because it allows to improve and understand the theorems from homotopy type theory using the primitives of the cubical set model which are cubes together with a composition operation. However, there remain big challenges to solve such as proving normalizability and using canonicity for in formalizing fundamental theorems in homotopy theory, see section 3.6.3. There is ongoing research whether the primitives of cubical type theory can be replaced by more general or efficient alternatives.

Which parts of [Shi18]
interesting? This rema
also made by other lea
of homotopy type theo
as [Boo18]

Index

Bibliography

[ABC$^+$17] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-bang Hou, Robert Harper, and Daniel R. Licata. Cartesian cubical type theory. 2017.

[Abe13] Andreas Abel. Normalization by evaluation: Dependent types and impredicativity. March 2013. Available at `http://www2.tcs.ifi.lmu.de/~abel/habil.pdf`.

[ACD$^+$19] Andreas Abel, Jesper Cockx, Nils Anders Danielsson, Ulf Norell, Andrés Sicard-Ramírez, et al. Agda 2.5.4.2: dependently typed programming language / interactive theorem prover, April 2019. Available at `https://github.com/agda/agda`.

[Acz12] Peter Aczel. Homotopy type theory and the structure identity principle. `https://www.newton.ac.uk/files/seminar/20120207160016301-153011.pdf`, February 2012.

[ADH$^+$19] Emilio Jesús Gallego Arias, Maxime Dénès, Hugo Herbelin, Pierre-Marie Pédrot, Matthieu Sozeau, Enrico Tassi, et al. Coq 8.9.0: a formal proof management system, January 2019. Available at `https://github.com/coq/coq`.

[AHH17] Carlo Angiuli, Kuen-Bang Hou, and Robert Harper. Computational higher type theory iii: Univalent universes and exact equality. December 2017.

[AHH18] Carlo Angiuli, Robert Harper, and Kuen-Bang Hou. Cartesian cubical computational type theory: Constructive reasoning with paths and equalities. In *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018. Available at `https://www.cs.cmu.edu/~rwh/papers/cartesian/paper.pdf`.

[All87] Stuart Allen. A non-type-theoretic definition of martin-löf's types. April 1987.

[Alt17] Thorsten Altenkirch. *Introduction to Homotopy Type theory, lecture notes.* 2017.

[Awo16] Steve Awodey. Notes on cubical models of type theory. June 2016. Available at `https://github.com/awodey/math/blob/master/Cubical/cubical.pdf`.

[Bae17] John Carlos Baez. Vladimir voevodsky 1966-2017. Blog post on `https://johncarlosbaez.wordpress.com/2017/10/06/vladimir-voevodsky-1966-2017/`, October 2017.

[Bau09] Andrej Bauer. Constructive gem: irrational to the power of irrational that is rational, December 2009.

[Bau13] Andrej Bauer. Socio-technological aspects of making the hott book, 2013. Available at `https://www.ias.edu/ideas/2013/bauer-hott-book`.

[BCdMH18] Edwin Brady, David Thrane Christiansen, and Jan de Muijnck-Hughes. Idris 1.3.1: Code generation. Available at `http://docs.idris-lang.org/en/latest/tutorial/miscellany.html#c-target`, April 2018.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, page 107–128, 2014.

[BCH18] Marc Bezem, Thierry Coquand, and Simon Huber. The univalence axiom in cubical sets. *Journal of Automated Reasoning*, June 2018.

[BD77] Raymond Balbes and Philip Dwinger. Distributive lattices. 1977.

[Ben18] Bruno Bentzen. Cubical informal type theory: the higher groupoid structure. *arXiv*, Jun 2018.

[BG70] De Bruijn and Nicolaas Govert. The mathematical language automath, its usage, and some of its extensions. In *Symposium on automatic demonstration*, page 29–61. Springer, 1970.

[Bib19] Curry–Howard correspondence - Wikipedia, April 2019. [Online; accessed 12. Apr. 2019].

[BL95] Daniel Bienstock and Michael A. Langston. Chapter 8 algorithmic implications of the graph minor theorem. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, page 481–502. Elsevier, 1995.

[BM12] Jean-Philippe Bernardy and Guilhem Moulin. A computational interpretation of parametricity. In *2012 27th Annual IEEE Symposium on Logic in Computer Science*, page 135–144. IEEE, 2012. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.908.7845&rep=rep1&type=pdf`.

[Boo18] Auke Booij. What i wish i knew when learning hott, 2018. Available at `https://abooij.github.io/wiwikwlhott/index.html`.

[Bro05] Luitzen Egbertus Jan Brouwer. Life, art and mysticism, 1905. Available at `https://projecteuclid.org/download/pdf_1/euclid.ndjfl/1039886518`.

[Bru16] Guillaume Brunerie. *On the homotopy groups of spheres in homotopy type theory*. PhD thesis, 2016. Available at `https://arxiv.org/pdf/1606.05916.pdf`.

[Bru18] Guillaume Brunerie. Experiments in cubical type theory. Slides from a talk given at UCSanDiego available at https://guillaumebrunerie.github.io/pdf/cubicalexperiments.pdf, January 2018.

[BV06] John Michael Boardman and Rainer M. Vogt. *Homotopy invariant algebraic structures on topological spaces*, volume 347. Springer, 2006.

[CCHM15] Thierry Coquand, Cohen Cyril, Simon Huber, and Anders Mörtberg. Cubicaltt: Experimental implementation of cubical type theory. GitHub repository available at https://github.com/mortberg/cubicaltt, February 2015.

[CCHM16] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *arXiv preprint arXiv:1611.02108*, 2016.

[CD13] Thierry Coquand and Nils Anders Danielsson. Isomorphism is equality. *Indagationes Mathematicae*, 24(4):1105–1120, 2013.

[CFL05] Luíz Cruz-Filipe and Pierre Letouzey. A large-scale experiment in executing extracted programs. 2005.

[CH18] Evan Cavallo and Robert Harper. Computational higher type theory iv: Inductive types. January 2018.

[CH19] Evan Callo and Robert Harper. Higher inductive types in cubical computational type theory. https://www.cs.cmu.edu/~rwh/papers/higher/paper.pdf, January 2019. (Accessed on 05/12/2019).

[CHM18] Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. *arXiv preprint arXiv:1802.01170*, 2018.

[CHS19] Thierry Coquand, Simon Huber, and Christian Sattler. Homotopy canonicity for cubical type theory. February 2019. Available at https://arxiv.org/abs/1902.06572.

[CM19] Evan Cavallo and Anders Mörtberg. A unifying cartesian cubical type theory. 2019. Available at http://www.cs.cmu.edu/~ecavallo/works/unifying-cartesian.pdf.

[Con03] Robert L. Constable. Naïve computational type theory. http://nuprl.org/documents/Constable/naive.pdf, Feb 2003. (Accessed on 04/28/2019).

[Con11] Robert L. Constable. The triumph of types: Creating a logic of computational reality. 2011. Available at https://ncatlab.org/nlab/files/ConstableTriumphOfTypes.pdf.

[Con15] Robert L. Constable. Two lectures on constructive type theory. https://www.cs.uoregon.edu/research/summerschool/summer15/notes/OPLSS-Short-2015-2.pdf, jul 2015. (Accessed on 04/28/2019).

[Coq13a] Thierry Coquand. A property of contractible types. December 2013.

[Coq13b] Thierry Coquand. Type theory and univalent foundation, October 2013. Available at http://www.cse.chalmers.se/~coquand/russell.pdf.

[Coq15] Thierry Coquand. A category of cubical sets. September 2015. Available at http://www.cse.chalmers.se/~coquand/vv.pdf.

[Cur34] Haskell Brooks Curry. Functionality in Combinatory Logic. *PNAS*, 20(11):584, November 1934.

[Dan12] Nils Anders Danielsson. Isomorphism implies equality, September 2012. Available at https://homotopytypetheory.org/2012/09/23/isomorphism-implies-equality/.

[DDG+19] Nils Anders Danielsson, Matthew Daggitt, Guillaume Gallais, Ulf Norell, Nicolas Pouillard, et al. The agda standard library. Available at https://github.com/agda/agda-stdlib, 2019.

[Din05] Jonathan Dinkelbach. Fundamental group of torus, February 2005. Available at https://commons.wikimedia.org/wiki/File:Fundamental_group_torus2.png.

[Eil39] Samuel Eilenberg. On the relation between the fundamental group on a space and the higher homotopy groups. *Fundamenta Mathematicae*, 32(1):167–175, 1939.

[Epp10] David Eppstein. The free distributive lattices of monotonic boolean functions on up to three variables. March 2010. Available at https://commons.wikimedia.org/wiki/File:Monotone_Boolean_functions.svg.

[Esc18] Martín Hötzel Escardó. A self-contained, brief and complete formulation of voevodsky's univalence axiom. October 2018. Available at https://arxiv.org/pdf/1803.02294.pdf.

[Esc19] Martín Hötzel Escardó. Introduction to univalent foundations of mathematics with agda. https://www.cs.bham.ac.uk/~mhe/HoTT-UF-in-Agda-Lecture-Notes/index.html, May 2019.

[Eur13] Academia Europeia. Member page: Per martin-löf. Available at http://www.ae-info.org/ae/User/Martin-L%C3%B6f_Per, 2013.

[FT63] Walter Feit and John Thompson. Chapter i, from solvability of groups of odd order, pacific j. math, vol. 13, no. 3 (1963. *Pacific journal of mathematics*, 13(3):775–787, 1963.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, and Et Al. A machine-checked proof of the odd order theorem, 2013.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, 39(1):176–210, December 1935.

[GM03] Marco Grandis and Luca Mauri. Cubical sets and their site. *Theory and Applications of Categories*, 2003. Available at http://www.tac.mta.ca/tac/volumes/11/8/11-08.pdf.

[Gra17] Daniel R. Grayson. Vladimir voevodsky, 4 june 1966 – 30 september 2017. Memorial webpage at http://www.math.ias.edu/Voevodsky/, 2017.

[Gra18a] Robert Graham. Synthetic homology in homotopy type theory. https://arxiv.org/pdf/1706.01540.pdf#page=20&zoom=100,0,701, December 2018. (Accessed on 05/14/2019).

[Gra18b] Daniel R. Grayson. An introduction to univalent foundations for mathematicians. *Bulletin of the American Mathematical Society*, 55(4):427–450, March 2018.

[GSS+99] Julio Rubio Garcia, Francis Sergeraert, Yvon Siret, et al. Kenzo: a symbolic software for effective homology computation. April 1999.

[GT90] Jean-Yves Girard and Paul Taylor. Proofs and types. 1990. Available at http://www.paultaylor.eu/stable/prot.pdf.

[Har91] Robert Harper. Constructing type systems over an operational semantics. https://core.ac.uk/download/pdf/82210261.pdf, November 1991. (Accessed on 05/14/2019).

[Hat01] Allan Hatcher. *Algebraic Topology*. Cambridge University Press, 2001. Available at http://www.math.cornell.edu/{~{}}hatcher/AT/ATpage.html.

[HBC+18] Kuen-Bang Hou, Guillaume Brunerie, Evan Cavallo, Andrej Bauer, Guillaume Brunerie, et al. Homotopy type theory in agda: Development of homotopy type theory in agda. October 2018.

[HBJ+18] Max Horn, Reimer Behrends, Christopher Jefferson, Markus Pfeiffer, Alexander Konovalov, Alexander Hulpke, Steve Linton, et al. Gap - groups, algorithms, programming - a system for computational discrete algebra. Downloads available at https://www.gap-system.org/, November 2018.

[HC11] Simon Huber and Thierry Coquand. Towards a computational justification of the axiom of univalence. September 2011. http://www.cse.chalmers.se/~simonhu/slides/types11.pdf.

[Hey30] Arend Heyting. Die formalen regeln der intuitionistischen logik, 1930.

[Hof97] Martin Hofmann. Syntax and semantics of dependent types. In *Extensional Constructs in Intensional Type Theory*, page 13–54. Springer, 1997.

[Hof14] Pieter Hofstra. Presheaves of sets. Online notes available at http://mysite.science.uottawa.ca/phofstra/MAT5147/presheaves.pdf, 2014.

[HS98] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. *Twenty-five years of constructive type theory (Venice, 1995)*, 36:83–111, 1998.

[Hub15] Simon Huber. *A model of type theory in cubical sets*. PhD thesis, Chalmers University of Technology, 2015. Available at http://www.cse.chalmers.se/~simonhu/misc/lic.pdf.

[Hub16a] Simon Huber. *Cubical Interpretations of Type Theory*. PhD thesis, University of Gothenburg, Gothenburg, Sweden, November 2016. Available at http://www.cse.chalmers.se/~simonhu/misc/thesis.pdf.

[Hub16b] Simon Huber. Cubical interpretations of type theory: Phd defense. November 2016. Available at http://www.cse.chalmers.se/~simonhu/slides/defense.pdf.

[Hub17a] Simon Huber. Canonicity for cubical type theory. *Journal of Automated Reasoning*, page 1–38, 2017. Available at https://arxiv.org/pdf/1607.04156.pdf.

[Hub17b] Simon Huber. A cubical type theory for higher inductive types. August 2017. Available at http://www.cse.chalmers.se/~simonhu/misc/hcomp.pdf.

[Jac17] Konrad Jacobs. Nicolaas govert de bruijn in oberwolfach. Available at https://opc.mfo.de/detail?photo_id=541, 2017.

[Kan55] Daniel M. Kan. Abstract homotopy. *Proceedings of the National Academy of Sciences*, 41(12):1092–1096, 1955.

[Kan05] Jonah Kan. A picture of daniel kan at his home in waban, massachusetts., October 2005. Available at https://commons.wikimedia.org/wiki/File:Daniel_Kan.JPG.

[KL12] Chris Kapulkin and Peter Lefanu Lumsdaine. The simplicial model of univalent foundations (after voevodsky). *arXiv preprint arXiv:1211.2851*, 2012.

[LB15] Daniel R. Licata and Guillaume Brunerie. A cubical approach to synthetic homotopy theory. In *Proceedings of the 2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 92–103. IEEE Computer Society, 2015. Available at https://guillaumebrunerie.github.io/pdf/lb15cubicalsynth.pdf.

[Lic13a] Daniel R. Licata. Homotopy theory in type theory: Progress report. May 2013. Available at https://homotopytypetheory.org/2013/05/20/homotopy-theory-in-type-theory-progress-report/.

[Lic13b] Daniel R. Licata. Programming and proving in homotopy type theory, August 2013. Available at http://dlicata.web.wesleyan.edu/pubs/l13jobtalk/l13jobtalk.pdf.

[LOPS18] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal universes in models of homotopy type theory. *arXiv preprint arXiv:1801.07664*, 2018.

[LS17] Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types. https://arxiv.org/abs/1705.07088, may 2017. (Accessed on 05/18/2019).

[Lum10] Peter Lefanu Lumsdaine. *Higher categories from type theories*. PhD thesis, PhD thesis, Carnegie Mellon University, 2010.

[MA18] Anders Mörtberg and Carlo Angiuli. Yet another cartesian cubical type theory. https://github.com/mortberg/yacctt/, June 2018.

[Ml75] Per Martin-löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. Shepherdson, editors, *Logic colloquium '73: proceedings of the Logic Colloquium, Bristols*, volume 80 of *Studies in Logic and the Foundations of Mathematic*, page 73–118. Elsevier, Amsterdam, 1975.

[MV19] Anders Mörtberg and Andrea Vezzosi. Cubical: An experimental library for cubical agda. Github repository, May 2019. Available at https://github.com/agda/cubical.

[Mö18a] Anders Mörtberg. Cubical agda. Blog post available at https://homotopytypetheory.org/2018/12/06/cubical-agda/, December 2018.

[Mö18b] Anders Mörtberg. Cubical variations: Hits, $\pi_4(s^3)$ and yacctt. http://www.cs.cmu.edu/~amoertbe/slides/Mortberg-MURI18.pdf, March 2018. (Accessed on 05/18/2019).

[Mö18c] Anders Mörtberg. Yet another cartesian cubical type theory. http://www.cs.cmu.edu/~amoertbe/slides/MortbergBonn.pdf, June 2018.

[Nuy18] Andreas Nuyts. Presheaf models of relational modalities in dependent type theory. May 2018.

[Ort19] Richard Ian Orton. *Cubical Models of Homotopy Type Theory-An Internal Approach*. PhD thesis, University of Cambridge, 2019. Text available at https://www.repository.cam.ac.uk/handle/1810/289441 and code on https://doi.org/10.17863/CAM.35681.

[Pal14] Erik Palmgren. Lecture notes on type theory, January 2014. Available at http://staff.math.su.se/palmgren/lecturenotesTT.pdf.

[PCZF+18] Clément Pit-Claudel, Théo Zimmermann, Jim Fehrle, Maxime Dénès, et al. Coq 8.9.0: Reference manual, December 2018. Available at https://coq.inria.fr/refman/addendum/extraction.html#a-detailed-example-euclidean-division.

[Pea79] Giuseppe Peano. Arithmetices principia, nova methodo exposita. *English translation in 1899*, page 83–97, 1879.

[Pit13] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.

[Pit14] Andrew M. Pitts. Nominal presentation of cubical sets models of type theory. 2014. Available at http://drops.dagstuhl.de/opus/volltexte/2015/5498/pdf/12.pdf.

[Pit18] Andrew Pitts. Axiomatizing cubical sets models of univalent foundations, September 2018. Available at https://hott-uf.github.io/2018/slides/PittsHoTTUF2018.pdf.

[PK19] Gun Pinyo and Nicolai Kraus. From cubes to twisted cubes via morphisms in type theory. May 2019. Available at https://arxiv.org/pdf/1902.10820.pdf.

[PW14] Álvaro Pelayo and Michael Warren. Homotopy type theory and voevodsky's univalent foundations. *Bulletin of the American Mathematical Society*, 51(4):597–648, 2014.

[Reh17] Julie Rehmeyer. Vladimir voevodsky, revolutionary mathematician, dies at 51. September 2017. Available on https://www.nytimes.com/2017/10/06/obituaries/vladimir-voevodsky-revolutionary-mathematician-dies-at-51.html.

[Rod18] Sigrid Roden. Prof. martin hofmann, phd. Message on the website of Ludwig-Maximilians-Universität München https://www.tcs.ifi.lmu.de/people/martin-hofmann, 2018.

[Ros84] Murray Rosenblatt. Erret bishop: Reflections on him and his research. 1984.

[Rus03] Bertrand Russel. *The principles of mathematics*. Cambridge University Press, 1903. Available at http://fair-use.org/bertrand-russell/the-principles-of-mathematics/.

[S$^+$11] Urs Schreiber et al. Sheafification. January 2011. Available at https://ncatlab.org/nlab/show/sheafification.

[SCS$^+$18] Urs Schreiber, David Corfield, Mike Schulman, David Roberts, Praphulla Koushik, and Others. Homotopy hypothesis. https://ncatlab.org/nlab/show/homotopy+hypothesis, 2018.

[SHA$^+$18] Jonathan Sterling, Kuen-bang Hou, Carlo Angiuli, Evan Cavallo, James Wilcox, et al. The people's refinement logic. https://github.com/RedPRL/sml-redprl, 2018.

[SHC$^+$18] Jonathan Sterling, Kuen-bang Hou, Evan Cavallo, Carlo Angiuli, et al. Redtt: a core language for cartesian cubical type theory with extension types, November 2018. File available at https://bit.ly/2FxsTyl.

[Shi18] Andy Shiue. Cubical type theory for dummies, April 2018. Available at https://gist.github.com/AndyShiue/cfc8c75f8b8655ca7ef2ffeb8cfb1faf/.

[Shu18] Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science*, 28(6):856–941, 2018.

[SRvD+19] Bas Spitters, David Roberts, Floris van Doorn, Mike Shulman, Steve Awodey, Ulrik Buchholtz, Urs Schreiber, et al. Open problems in homotopy type theory. `https://ncatlab.org/homotopytypetheory/show/open+problems`, May 2019.

[Str06] Thomas Streicher. Identity types vs. weak ω-groupoids. Talk given at the Work- shop "Identity Types – Topological and Categorical Structure" (Uppsala, November 13-14, 2006), November 2006.

[Swa14] Andrew Swan. An algebraic weak factorisation system on 01-substitution sets: A constructive proof. *arXiv preprint arXiv:1409.1829*, 2014.

[Swa18] Andrew Swan. Identity types in algebraic model structures and cubical sets. `https://arxiv.org/abs/1808.00915`, August 2018. (Accessed on 05/12/2019).

[Tar51] Alfred Tarski. A decision method for elementary algebra and geometry. 1951. Available on `https://www.rand.org/content/dam/rand/pubs/reports/2008/R109.pdf`.

[Uem19] Taichi Uemura. A general framework for the semantics of type theory. April 2019. Available at `https://arxiv.org/pdf/1904.04097.pdf`.

[VAC+13] Vladimir Voevodsky, Steve Awodey, Thierry Coquand, et al. Homotopy type theory: Univalent foundations of mathematics. *Institute for Advanced Study (Princeton)*, 2013.

[VAG+10] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath-a library of formalized mathematics. Github repository available at `https://github.com/UniMath/UniMath`, 2010.

[Van19] Willem Vanhulle. Transport of magmas. May 2019. GitHub repository on `https://github.com/wvhulle/transport-magmas`.

[Voe09] Vladimir Voevodsky. Notes on type systems. Unpublished note available at `https://github.com/vladimirias/old_notes_on_type_systems/raw/master/old_notes_on_type%20systems.pdf`, September 2009.

[Voe10] Vladimir Voevodsky. The equivalence axiom and univalent models of type theory. Notes from a talk at Carnegie Mellon University available at `http://www.math.ias.edu/vladimir/files/CMU_talk.pdf`, 2010.

[Voe14] Vladimir Voevodsky. Univalent foundations—new type-theoretic foundations of mathematics. Slides from a talk at IHP, Paris, on April 22, 2014, available at `https://math.ias.edu.vladimir/files/2014_04_22_slides.pdf`, 2014.

[Voe16] Vladimir Voevodsky. Unimath - a library of mathematics formalized in the univalent style. Talk at ICMS, Berlin available at https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/2016_07_14_Berlin_ICMS_short.pdf, July 2016.

[Wei16] Jonathan Weinberger. The cubical model of type theory. Master's thesis, November 2016. Available at https://jonathanweinberger.files.wordpress.com/2016/10/msc-weinberger.pdf.

DEPARTMENT OF MATHEMATICS
Celestijnenlaan 200B
3001 LEUVEN, BELGIË
tel. + 32 16 32 70 15
https://wis.kuleuven.be/english