Kuratowski Finite Sets in the UniMath Library

Luuk van de Laar[1]

Supervisors: Dr. Benedikt Ahrens[1]**, Kobe Wullaert**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Luuk van de Laar
Final project course: CSE3000 Research Project
Thesis committee: Dr. Benedikt Ahrens, Kobe Wullaert, Niel Yorke-Smith

Abstract

This paper focuses on implementing and verifying the proofs presented in "Finite Sets in Homotopy Type Theory" within the UniMath library. The UniMath library currently lacks support for higher inductive types, which are crucial for reasoning about finite sets in Homotopy Type Theory. This paper addresses that issue and introduces higher inductive types to UniMath. This is used to develop a computer-checked implementation of the proofs within "Finite Sets in Homotopy Type Theory." This implementation enables future research on finite sets in HoTT by providing accessible and reliable proofs.

This paper defines finite sets as Kuratowski-finite. This is in contrast with the most common notion of finiteness, e.g. Bishop-finite and enumerated types. I argue that Kuratowski-finiteness is the most general finite for which the usual operations of finite types and sub-objects can be operated upon.

1 Introduction

Homotopy type theory (HoTT) provides a unique perspective on mathematics, offering insights that are not easily attainable through set theory [1]. One way it achieves this is by employing the univalence axiom. This states that identity and equivalence are the same. This allows for reasoning about the equality of two different types [13].

The univalence axiom also allows to employ higher Inductive Types (HITs) to reason inductively over the types at an equivalence relation. The groundwork for defining sets in HoTT has already been laid [17]. However, this is independently written and is not integrated into the commonly used library, UniMath [18]. HITs are important for future research on HoTT, that is why the UniMath library should support these HITs. Access to these properties allows us to properly reason over finite sets. As these HITs allow us to reason about variables inside the sets. For example, proving how the union between two sets behaves if both sets are finite is important for future research in the area about sets and HoTT.

In this paper the background of HoTT will be discussed within detail, and we will answer the research question. *Can the proofs given in Finite Sets in Homotopy Type Theory be verified for correctness in the UniMath library [17]?* First, a bit of background on HoTT will be given, it will mostly focus on the material needed to understand HITs. After this short introduction to HoTT you can find other related works. These can be used to further develop an understanding of the topic. Then a short summary of the setup will be given. After which sets will be defined as a Kuratowski type [6]. The induction principle and induction property will be given and these some basic properties of a set will be proven. After which membership and subset of the Kuratowski finite set will be proven. The paper will end with discussing the results and conclude that the proofs discussed in this paper from Finite Sets in Homotopy Type Theory can indeed be implemented within the UniMath library.

It is important to note that this paper focuses on Kuratowski finite. And it does not focus on Bishop- or enumerated finite. The reason this decision has been made is because Kuratowski finite is "weaker" then both Bishop- and enumerated finite. This is because for Kuratowski finite you have to proof a surjection from a finite set of natural numbers up to an arbitrary n, while for Bishop finite you have to proof a bijection instead of a surjection. And for enumerated finite you have to proof that the set can be enumerated. Because Kuratowski finite is less strict it has been chosen to focus on Kuratowski finite in this paper.

In this paper the proofs given in Finite Sets in Homotopy Type Theory [17] will be independently computer checked and developed as if it is a package in the UniMath library. As the UniMath library does not include definitions for HITs, they will be introduced with this paper. Sections not covered by my implementation can be implemented later using my paper as its starting point.

All resources of this paper are formalised and can be found at https://github.com/blibliboe/Kuratowski_in_UniMath.

2 Homotopy Type Theory

In this chapter I will be giving a short breakdown of the Homotopy Type Theory used in this research. I will use the standard convention of notating HoTT [13], so readers familiar with HoTT can skip this part.

In HoTT there are a few terms that are useful to know when reading this paper. First of all, if you have a type A, you can say that a certain variable is of type A by saying, a is a witness of A. This is notated as follows: $a : A$. Secondly, you could have the following type: $a = b$. Each witness of this type would be called a path from a to b. Thirdly, you can have a type that says something for all $a : A$. This is called a pi type and is notated as follows: $\Pi(a : A), a = a$. This would create a type that says for each witness of A that witness has a path to itself. Lastly you also have the sigma type. The sigma type represents dependent pairs, where the type of the second component can depend on the value of the first component. The sigma type is not so relevant for this paper as it is barely used in finite sets, but it is important to keep in mind as it is one of the pillars of HoTT, namely the univalence axiom.

One of the most important types in HoTT is the identity type. This type is the inductive type that references itself, also known as refl of type: $\Pi(x : A), x = x$, with A indicating the type of x. To actually reason on this equality, any two elements x of the same propositional equality have to also be propositionally equal, a J eliminator is introduced. This rule says that given a type family $c : \Pi(x, y : A), x = y \to Type$ and a witness r of type $\Pi(x : A), c\ x\ x\ (refl\ x)$, we get the J eliminator.

$$J(c, r) : \Pi(x, y : A), \Pi(p : x = y), c\ x\ y\ p \tag{1}$$

In essence it comes down to if there exist a witness r (so $p : x = x$), then a proof that $q : x = y$ is sufficient to say that both p and q are in the same type family.

[1] From now on, I have utilized ChatGPT to verify my grammar and seek assistance in phrasing my sentences more elegantly [11]

This lets reasoning about symmetry:

$$symm_A : \Pi(x, y : A), x = y \to y = x \tag{2}$$

and about transitivity :

$$trans_A : \Pi(x, y, z : A), x = y \to y = z \to x = z \tag{3}$$

For paths $p : a = b$ and $q : b = c$ we notate it as $p^{-1} := symm_A\ a\ b\ p$ and $p\dot{q} := trans_A\ a\ b\ c\ p\ q$

It also allows reasoning about substituting paths along type families. For this a transport function is defined:

$$transportf : \Pi(x, y : A), x = y \to P\ x \to P\ y \tag{4}$$

With the type of P being $P : X \to Type$, a type family over A. The notation for transport is: $p_* := transportf\ x\ y\ p$.

Next to the identity type there also exist an equality called definitional equality. Definitional equality is a notion of equality that is established by the definition or construction of terms. Definitional equality is denoted by $\equiv$. Because this equality is established by definition it is implied that two equal types cannot be distinguished. Take for example $n : A$ and $B : Type$ such that $A \equiv B$, then $n : B$. For the transport function there exist also a definitional equality: $(relf\ t)_* s \equiv refl\ s$.

Another feature of HoTT is the univalence axiom, which I briefly touched upon when talking about sigma types and to employ HITs. Here it will be formulated a bit more precise. To do that I first need to define equivalence.

$$isEquiv(f) := \sum_{g:B\to A}(f \circ g = id_B) \times \sum_{h:B\to A}(h \circ f = id_A) \tag{5}$$

If we have an equivalence, $A \simeq B := \sum_{f:A\to B} isEquiv(f)$ has a witness. The axiom then says if there is an equivalence $A \simeq B$, then those types are equal and there exist a path $A = B$

HoTT also has a definition for a type of which all its witnesses are equal. This is called a mere proposition. This is formulated as follows: $isaprop(A) := \Pi(x, y : A), x = y$. It is customary to refer to a mere proposition as just"proposition". When referring to something as a proposition I mean that it is actually a mere proposition. If A would be a proposition you would write it as A : HPROP with HPROP $:= \Sigma(A : Type), isaprop(A)$. An example of a proposition is $A \simeq B$.

Next to propositions HoTT also has a notion of a set. We say that A : HSET if all paths in type A are all a proposition. Therefore, it is defined as HSET $:= \Sigma(A : Type), (isaset(A))$ and $isaset(A) := \Pi(x, y : A), (\Pi(p, q : x = y), p = q)$.

In HoTT you also have the concept of HITs, this allows for defining a type by giving inductive constructors and/or equations for that type. A useful example of such a type is the truncation type. Truncation is notated as $||A||$ of A. Whenever you use truncation on a type it creates a proposition. $||A||$ has one and only one element when A has at least one element. $||A||$ has zero elements if A has no elements.

$$\begin{aligned} &||A|| := \\ &| \ tr : A \to ||A|| \\ &| \ trc : \Pi(x, y : ||A||), x = y \end{aligned}$$

This is useful as when I will be discussing finite sets, we are interested if two elements of the sets have decidable equality. This means there is a witness of type $\Pi(x, y : A), x = y + \neg(x = y)$. In practice however it is often enough to see if the sets have decidable mere equality which is defined as $\Pi(x, y : A), ||x = y|| + ||\neg(x = y)||$.

There are also HITs which are not a set, the circle is an example of one of them.

$$\begin{aligned} &S^1 := \\ &| \ base : S^1 \\ &| \ loop : base = base \end{aligned}$$

For this HIT it has been shown that it is not a set [8].

3 Related works

Homotopy Type Theory which is the mathematical backbone of this paper, is explained fully in Homotopy Type Theory book [15]. This book goes a lot more in depth on every subject of HoTT which this paper did not touch. It does, however, not touch the non-trivial Higher Inductive Types or higher dimensional type theory.

There are several papers on the semantics of Higher Inductive Types [1, 9]. This can be related to programming languages formalized using Coq [16]. Here general rules for recursive and non-recursive HITs have been given.

There are a few examples of HITs applied to problems in computer science. A way to describe type theory in type theory [2], which is possible as type theory is a mathematical basis the same as set theory. Or a way to link cubical type theory to topos theory [12].

In mathematics finite sets have been studied and defined clearly [4]. In this paper you can find the definitions of finite sets based on relations towards other finite sets. Different definitions of finite sets exist, this includes Bishop-finite [10], Kuratowski finite sets [6] and enumerated sets. Different set have been formalised as HITs in HoTT [17]. Finite sets have also been implemented but those implementation did not use HITs [18].

It has been proven for Kuratowski finiteness that a topos of this space is a boolean if and only if every object in this set is decidable [5]. In HoTT it is proven that for Bishop finiteness, the universe of sets form a topos which is a boolean. [15, Theorem 10.1.15]. These proofs however did not consider Kuratowski finiteness.

Lastly there are other interfaces of finite sets that have already been developed. Most notably and relevant is the one by Lescuyer [7]. As this interface is also based on Coq. A note to this interface is that it does not use the HITs that are being used in this paper.

4 Formalization of proofs in UniMath: Finite Sets in Homotopy Type Theory

Finite Sets in Homotopy Type Theory [17] is a paper that defines finite sets using homotopy type theory. To do this it defines a Kuratowski finite set [6]. This set is defined with the univalence axiom in mind, using Coq to prove fact that the defined Kuratowski type is in fact finite and a set. As part of

the Univalent Mathematics group, the goal my paper tries to achieve is to verify and implement these finite sets and proofs of finiteness into the UniMath library. As the paper on finite sets contains proofs and types that are important for a better understanding of HITs. But these proofs and types have not yet been implemented within the framework of the UniMath library. [2]

This brings us to the research question, which is: *Can the proofs given in Finite Sets in Homotopy Type Theory be verified for correctness in the UniMath library [17]?*

4.1 Preparatory phase

During the first part of the research project I was still a novice in my understanding of HoTT. That is why I started with a preparatory phase to get a better understanding of HoTT. In this preparatory phase I have done research about the inner working of HoTT, the proof assistant Coq and the UniMath library. I have learned HoTT using the following resources: the HoTT book [15], online resource the HoTTest summer school [3] and question hours with my responsible professor and supervisor. I have learned about both the UniMath library and Coq at the same time, using the UniMath library, as the library is written in the programming language Coq. This learning has been done by studying the UniMath schools [14], which has 7 lectures on how to use UniMath and Coq.

4.2 Research phase

During the research phase of the project some of the proofs in Finite Sets of Homotopy Type Theory were be implemented, as well as some of the HITs defined in the paper. Due to time constraints not all proofs and types were formalised. The goal of the project is to verify the Kuratowski finite set and verify membership and subsets of the Kuratowski finite set [17, Section 2].

These proofs and types will then be verified using Coq as a proof assistant. The proof assistant will only verify a proof is it can be proven with its internal resources. The reason Coq is used is because the UniMath library is build with the proof assistant Coq. Every proof formalised within my paper will also have a similar proof implemented within the UniMath library. This will be done step by step, first proving the statement on paper, then proving the statement within UniMath using Coq.

The proofs and types were written in windows 11, using visual studio code as the IDE. For visual studio code the extension, VsCoq, was installed so that I could interface Coq within visual studio code. A binary file of UniMath was installed as well. An important note is that the binary file used is from an older version of the UniMath library. This decision was made due to the fact that the UniMath library can only be compiled on Linux, which was not available to me. Consequently, a binary file of the UniMath library code, specifically UniMath-20220204, was employed for proof development. As a result, my research is somewhat lagging behind the latest advancements in the library.

[2] I have summarized and condensed this part of my paper using ChatGPT

A second disadvantage of the binary file is that I was unable to establish my own file structure while working on the implementation of finite sets. This made it so that the source code of this paper is all developed in the same file.

5 Defining finite sets

The goal of this paper is to computer type check proofs within the UniMath library. It is necessary to define finite sets to be able to do this. In the UniMath framework one can easily define a data type using inductive types. This however will create serious hurdles when trying to define a subset of the type. That is why finite sets will be defined using Higher Inductive Types.

This is done because HITs allow for both point and path constructors in the type definition, of which only the first is allowed by inductive types. In this section the representation of the Kuratowski type will be given. The types are built step by step starting with the empty set and a set with only one element. It will then define a union and define path constructors. These path constructors describe the behaviour of the Kuratowski type when reasoning over the union.

5.1 Kuratowski finite set

We start by defining the type $\mathcal{K}(A)$ as a Kuratowski type, of which its witnesses are a finite set.

Definition 5.1: Given a type A, we define type $\mathcal{K}(A)$ as follows:

$\mathcal{K}(A) :=$
$| \ \emptyset : \mathcal{K}(A)$
$| \ \{\cdot\} : A \to \mathcal{K}(A)$
$| \ \cup : \mathcal{K}(A) \to \mathcal{K}(A) \to \mathcal{K}(A)$
$| \ idem : \Pi(x : \mathcal{K}(A)), \{x\} \cup \{x\} = x$
$| \ nl : \Pi(x : \mathcal{K}(A)), \emptyset \cup x = x$
$| \ nr : \Pi(x : \mathcal{K}(A)), x \cup \emptyset = x$
$| \ assoc : \Pi(x, y, z : \mathcal{K}(A)), x \cup (y \cup z) = (x \cup y) \cup z)$
$| \ comm : \Pi(x, y : \mathcal{K}(A)), x \cup y = y \cup x$
$| \ trunc : isaset(\mathcal{K}(A))$

Each of the lines is a different constructor for the finite set. The first three lines are the point constructors of the HIT. The empty set is a witness of the Kuratowski type, the function making a set with a single element is a witness of the Kuratowski type. Lastly, a function that says that the union of two Kuratowski types is in itself a Kuratowski type.

The other six contribute to the path constructors. Of these six, all of them, expect trunc, describe relations of the Kuratowski set towards itself. Specifically that the Kuratowski type is idempotent, associative and commutative. It also defines two constructor which say that the union between a Kuratowski type and the empty set is the original Kuratowski type. The last path constructor, trunc, forces the $\mathcal{K}(A)$ to be an HSET. This is the same as in the original paper was described [17, Definition 2.1].

Now that the definition of the finite set is given, we are ready to define the induction principle and induction property. This is different from the original paper as they only

define the induction property. You also have the recursion principle, this will not be discussed within this paper as you can always derive it from a valid induction principle.

Definition 5.2 Given a type family $P : \mathcal{K}(A) \to Type$ the induction principle suggests that an eliminator of type $\Pi(x : \mathcal{K}(A)), P\ x$ exists. The different constructors from the induction principle you can be found below:

$ind :=$
$|A : Type$
$|P : \mathcal{K}(A) \to Type$
$|H : \Pi(X : \mathcal{K}(A)), isaset(PX)$
$|eP : P\ \emptyset$
$|lP : \Pi(a : A), P\ \{a\}$
$|uP : \Pi(x, y : \mathcal{K}(A)), P\ x \to P\ y \to P\ (x \cup y)$
$|assocP : \Pi(x, y, z : \mathcal{K}(A))\ (px : P\ x)\ (py : P\ y)\ (pz : P\ z),$
$\quad transportf\ P(assoc\ x\ y\ z)(uP\ x\ (y \cup z)px(uP\ y\ x\ py\ pz)$
$\quad = (uP\ (x \cup y)\ z\ (uP\ x\ y\ px\ py)\ pz)$
$|commP : \Pi(x, y : \mathcal{K}(A))\ (px : P\ x)\ (py : P\ y),$
$\quad transportf\ P(comm\ x\ y)(uP\ x\ y\ px\ py) = (uP\ y\ x\ py\ px)$
$|nlP : \Pi(x : \mathcal{K}(A))\ (px : P\ x),$
$\quad transportf\ P\ (nl\ x)\ (uP\ \emptyset\ x\ eP\ px) = px$
$|nrP : \Pi(x : \mathcal{K}(A))\ (px : P\ x),$
$\quad transportf\ P\ (nr\ x)\ (uP\ x\ \emptyset\ eP\ px) = px$
$|idemP : \Pi(a : A),$
$\quad transportf\ P\ (idem\ a)\ (uP\ \{a\}\{a\}\ (lP\ x)(lP\ x)) = lP\ x$

---

there exists	**ind** $: \Pi(x : \mathcal{K}(A)), P\ x$
such that	**ind** $\emptyset = eP$
	ind $\{a\} = lP$
	ind $(x \cup y) = uP$ (**ind** x) (**ind** y)

The first three constructors define what we described above. The next three constructors eP, lP and uP describe how the function P interacts with the point constructors of $\mathcal{K}(A)$. The last five constructors describe the way the function P interacts with all the constructors of $\mathcal{K}(A)$ that are defined in HITs.

It is important to note that the transportf function has to be used to define this induction principle, otherwise the function P will not be lifted into the respective higher function constructors.

To verify the induction principle we can show how the eliminator acts on each of the constructors. We will only need the computation rules for the points, that is why the computation rules for the paths are not given.

Now that we know how induction is defined, it is also useful to know the rules of the type when $P\ x$ is a HPROP instead of a HSET. For this I have defined the induction property.

Definition 5.3 The induction property.

$ind\_prop :=$
$|A\_prop : Type$
$|P\_prop : \mathcal{K}(A) \to Type$
$|H\_prop : \Pi(X : \mathcal{K}(A)), isaprop(PX)$
$|eP\_prop : P\ \emptyset$
$|lP\_prop : \Pi(a : A), P\ \{a\}$
$|uP\_prop : \Pi(x, y : \mathcal{K}(A)), P\ x \to P\ y \to P\ (x \cup y)$

Lemma 5.4 Given $x : \mathcal{K}(A)$ and the induction property we can show the induction principle.

Proof. First, we need to prove H with H_prop. $P\ x$ is a HPROP given by H_prop, it is also a HSET [15, Lemma 3.3.4]. This proves H. Next, using the fact that H_prop is a HPROP, it follows that $assocP$, $commP$, nlP, nrP, and $idemP$ are also correct. Lastly, we need to prove eP, lP and uP, they simply follow from eP_prop, lP_prop and uP_prop.
QED.

The reason to prove that we can get the induction principle from the induction property is very useful when trying to proof an attribute of Kuratowski type which is a HPROP.

Lemma 5.5 For any $x : \mathcal{K}(A), x \cup x = x$.

Proof. By induction property, first I need to prove that $x \cup x = x$ is a HPROP. As both $x \cup x$ and x are a witness of $\mathcal{K}(A)$, this holds by the definition of trunc. Next, we need to prove it for $x = \emptyset$. This can be done by either **nl or nr**. Next, a proof for $x = \{a\}$ is needed. This also follows directly from **idem**. Lastly, given $x = x \cup x$ and $y = y \cup y$, we need to prove that $(x \cup y)\ \cup\ (x \cup y) = x \cup y$. With the **assoc** and **comm** rule we can rewrite it to: $(x \cup x)\ \cup\ (y \cup y) = x \cup y$. And with the given hypotheses we can show $x \cup y = x \cup y$.
QED.

The way that it is proven in my paper is different than in the proof given in Finite Sets in Homotopy Type Theory [17, Lemma 2.3]. As I have defined the induction property, and proved lemma 5.5 with the induction property. While the original paper proved it with the induction principle.

5.2 Extensionality

In set theory one of the most important axioms is extensionality. This axioms state that two sets are the same if and only if they have the same elements. Because the witnesses of the Kuratowski type represent a finite set, a start has been made to proof this for the Kuratowski type. This shows that I can define members of the Kuratowski set as well as subsets of the Kuratowski set.

Definition 5.6 Assuming univalence, given a type A, a membership function $\in$ has been defined. $\in$: $A \to \mathcal{K}(A) \to$ HPROP. For $a : A$ we define the membership for the point

constructors as follows:

$$a \in \emptyset \equiv hfalse$$
$$a \in \{b\} \equiv ||a = b||$$
$$a \in (x \cup y) \equiv a \in x \vee a \in y$$

Where hfalse means there is no HPROP that exist that satisfies the condition. After having defined this for the point constructors it has to be proven for the path constructors.

Lemma 5.7 Given a membership function $\in$: $A \to \mathcal{K}(A) \to$ HPROP, membership of a Kuratowski finite set can be determined.

Proof. The point constructors follow from definition 5.6. Trunc follows from the fact that $isaset$ is a HPROP [18]. The other path constructors follow from the fact that $\vee$ is idempotent, associative, commutative and $x \vee false = x$. QED.

Definition 5.8 Assuming univalence, given a type A, a subset function $\subseteq$ has been defined $\subseteq$: $\mathcal{K}(A) \to \mathcal{K}(A) \to$ HPROP. For $a : \mathcal{K}(A)$ we define a subset function as follows:

$$\emptyset \subseteq a \equiv htrue$$
$$\{b\} \subseteq a \equiv b \in a$$
$$\{x, y\} \subseteq (a \cup a) \equiv x \subseteq a \wedge y \subset a$$

Where htrue means that every HPROP that exist satisfies the condition. The same way as for membership after defining the point constructors it has to be proven for the path constructors.

Lemma 5.9 Given a subset function $\subseteq$: $\mathcal{K}(A) \to \mathcal{K}(A) \to$ HPROP, subset of a Kuratowski finite set can be determined.

Proof. The point constructors follow from definition 5.8. Trunc follows from the fact that $isaset$ is a HPROP [18]. The other path constructors follow from the fact that $\wedge$ is idempotent, associative, commutative and $x \wedge true = x$. QED.

6 Results

With the given proofs, it results that implementing the proofs within Finite Sets in Homotopy Type Theory, is possible within the UniMath library. I have implemented the proofs, lemmas and definitions, as described in this paper, within the UniMath library. From the research that I have done it can be concluded that the proofs, lemmas and definitions up to membership of Finite Sets in Homotopy Type Theory are valid and can be implemented within the UniMath library.

7 Responsible Research

My research barely touched on ethical questions. The research is highly theoretical and uses a new way of mathematics which has not yet seen the light of day outside of research. This means that reasoning about the ethical implications of my research is nigh impossible.

Next my research is also highly reproducible. I have written everything I did in the research paper, including the sources I used to learn about UniMath, Coq and HoTT. Implementing the proofs in HoTT should be reproducible as this is highly mathematical and the proofs do not change when using the same library. I also made my code publicly available at https://github.com/blibliboe/Kuratowski_in_UniMath so you can check the proofs for yourself.

8 Challenges

While doing my research there were some challenges that had to be overcome in order to formalize the results. In this chapter the challenges that I had while working on the problem will be discussed as well as the impact it has on the final results.

The main challenge I faced when formalizing the results was applying the right lemmas. These lemmas already existed within the UniMath library, but finding them was a real challenge. This could lead to the proofs that have been implemented not being sufficient for all contexts they could be used in. However, with the proofs within Finite Sets in Homotopy Type Theory in mind, this is rather unlikely.

Another challenge I faced were the nuances of certain proofs. These nuances changed between my paper and Finite Sets in Homotopy Type Theory [17]. This should not lead to any discrepancies between the two papers as the conclusions of both papers stayed the same for each individual proof.

There is also a discrepancy between the discussed proofs in this paper and the proofs in my code. Some of the proofs in my code are admitted which means that these proofs are not yet finalized, and therefor left out of the paper. These proofs are probably able to be proven, or in some cases can be proven if I would now the right tactic. Further research needs to be done to formalize those proofs.

Lastly, a more personal challenge I faced was the difficulty of actually proving the statements. This was always more difficult than originally anticipated. It mostly came down to my inexperience using UniMath and Coq. Making a lot of relatively "easy" proofs, such as the recursion principle, a principle you can always deduce from the induction principle, a rather daunting task that took me way longer than I had hoped. Mostly through my inexperience with specifically the $transportf$ function.

Another reason most proofs took longer than anticipated, was due to the fact that proving something for finite sets would mean you have to proof nine different sub-proofs for a single proof, four in the case I could use the induction property. As every proof would hinge on each constructor of the Kuratowski type. This made proving lemmas a lot more difficult as making a wrong assumption in one of the point constructors could lead to me being unable to proof certain path constructors.

9 Conclusions and Future Work

In this paper I have shown that certain proofs and types of Finite Sets in Homotopy Type Theory can be implemented in the UniMath library. Most important of these proofs and

types is the Kuratowski type. The Kuratowski type can easily be implemented as a Higher Inductive Type $\mathcal{K}(A)$ for which reasoning about its properties is sufficient in UniMath. In addition to the Kuratowski type, membership and subset have of the Kuratowski type have been defined. Allowing reasoning on witnesses of the Kuratowski type.

There is still a lot of work that could be done working on finite sets. It might be useful to finish the proofs on extensionality based on membership and subset of the Kuratowski type. It might also be useful to create a listed set of which the size is known and proof an equality between the listed set and the Kuratowski type. This is useful as you get a way to reason about the exact size of the Kuratowski finite set. In a similar fashion one could proof extensionality with booleans instead of using HPROP, allowing for a different reasoning of the Kuratowski finite set.

In mathematics there are multiple ways to describe the finiteness of an object. These different ways, Kuratowski-, Bishop- and enumerated finite could all be a reasonable approach when implementing finite sets in UniMath. In future research one could define finite sets in UniMath using either Bishop or enumerated finiteness.

Lastly, the work that I have delivered should be kept up to date with the UniMath library. This should be done so that the Kuratowski type which I have defined in my paper can still be used within the UniMath framework. To continuing developing the research, UniMath should look at a way to start supporting Higher Inductive Types as they are the pillar of the Kuratowski type.

A Use of AI in this paper

Usage of AI was allowed by writing this paper as long as it was clear to the reader what kind of AI was used. In this chapter I will write down a small report of things that I have used AI for while working on the research project.

First of all the AI was used to improve my writing skills. This was done by asking ChatGPT [11] a very simple prompt. Can you spot grammatical or spelling mistakes inside the following piece of text? Can you also list them by cross referencing the original text? Followed by the text that I wanted to get checked. ChatGPT then gave a list of what it thought would be improvements to the readability of the paper. To incorporate these suggestions, I manually reviewed each one and made changes to my paper accordingly, either by adopting the full suggestion or by incorporating a modified version. In essence I used ChatGPT as a proofreader of my paper pointing out mistakes and sections which could use a bit of an improvement.

Another place I have used AI is while trying to summarize some parts of the paper, I have used this only once. This worked almost the same as asking for suggestions. I asked ChatGPT the following: Can you summarize the following text in X words or less? Followed by the text. I then used this summary as a basis to write my own paper in a more concise matter.

I also used ChatGPT to seek assistance when encountering challenges while working on a code segment. I would provide ChatGPT with the code in question and request help on achieving the desired outcome. However, this approach did not yield significant results, as ChatGPT is not well-versed in the UniMath library. Nevertheless, this experience prompted me to explore tactics employed in Coq, which proved to be helpful in my search for solutions.

References

[1] Kaposi A.& Kovács A. A Syntax for Higher Inductive-Inductive Types. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:18, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[2] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. *ACM SIGPLAN Notices*, 51(1):18–29, 2016.

[3] Western Univeristy Canada. Hottest summer school, 07 2022.

[4] Agata Darmochwał. Finite sets. *Formalized Mathematics*, 1(1):165–167, 1990.

[5] PT Johnstone and FEJ Linton. Finiteness and decidability: Ii. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 84, pages 207–218. Cambridge University Press, 1978.

[6] Casimir Kuratowski. Sur l'opération a de l'analysis situs. *Fundamenta Mathematicae*, 3(1):182–199, 1922.

[7] Stephane Lescuyer. *Formalizing and implementing a reflexive tactic for automated deduction in Coq*. PhD thesis, Paris 11, 2011.

[8] Daniel R Licata and Michael Shulman. Calculating the fundamental group of the circle in homotopy type theory. In *2013 28th annual acm/ieee symposium on logic in computer science*, pages 223–232. IEEE, 2013.

[9] Lumsdaine P. F. & Shulman M. Semantics of higher inductive types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 169(1):159 – 208, 7 2020.

[10] Ray Mines. Constructive analysis. by errett bishop and douglas bridges. *The American Mathematical Monthly*, 95(2):159–163, 1988.

[11] OpenAI. Chatgpt (may 24 version). https://openai.com/chat, 2023.

[12] Ian Orton and Andrew M Pitts. Axioms for modelling cubical type theory in a topos. *Logical Methods in Computer Science*, 14, 2018.

[13] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. *arXiv preprint arXiv:1308.0729*, 2013.

[14] Paige Randall North, Benedikt Ahrens, Niels van der Weide, et al. Unimath school — lectures about a computer-checked library of univalent mathematics. available at http://unimath.org.

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

[16] Basold H. Geuvers H. & Weide N.M. van der. Higher inductive types in programming. *Journal of Universal Computer Science*, 23:63–88, 7 2017.

[17] Frumin Geuvers Gondelman van der Weide. Finite sets in homotopy type theory. *CPP 2018: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 201–214, 1 2018.

[18] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath — a computer-checked library of univalent mathematics. available at http://unimath.org.