Finiteness in Cubical Type Theory

Donnacha Oisín Kidney[1] and Gregory Provan[2]

[1] University College Cork o.kidney@cs.ucc.ie
[2] University College Cork g.provan@cs.ucc.ie

Abstract. We study five different notions of finiteness in Cubical Type Theory and prove the relationship between them. In particular we show that any totally ordered Kuratowski finite type is manifestly Bishop finite.
We also prove closure properties for each finite type, and classify them topos-theoretically. This includes a proof that the category of decidable Kuratowski finite sets (also called the category of cardinal finite sets) form a Π-pretopos.
We then develop a parallel classification for the countably infinite types, as well as a proof of the countability of $A^\star$ for a countable type A.
We formalise our work in Cubical Agda, where we implement a library for proof search (including combinators for level-polymorphic fully generic currying). Through this library we demonstrate a number of uses for the computational content of the univalence axiom, including searching for and synthesising functions.

Keywords: Agda · Homotopy Type Theory · Cubical Type Theory · Dependent Types · Finiteness · Topos · Kuratowski finite

1 Introduction

1.1 Foreword

In constructive mathematics we are often preoccupied with *why* something is true. Take finiteness, for example. There are a handful of ways to demonstrate some type is finite: we could provide a surjection to it from another finite type; we could show that any collection of its elements larger than some bound contains duplicates; or we could show that any stream of its elements contain duplicates.

Classically, all of these proofs end up proving the same thing: that our type is finite. Constructively (in Martin-Löf Type Theory [16] at least), however, all three of the statements above construct a different version of finiteness. *How* we show that some type is finite has a significant impact on the type of finiteness we end up dealing with.

Homotopy Type Theory [19] adds another wrinkle to the story. Firstly, in HoTT we cannot assume that every type is a (homotopy) set: this means that the finiteness predicates above can be further split into versions which apply to sets only, and those that apply to all types. Secondly, HoTT gives us a principled and powerful way to construct quotients, allowing us to regain some of the flexibility

of classical definitions by "forgetting" the parts of a proof we would be forced to remember in MLTT.

Finally, for a computer scientist constructive mathematics has one invaluable feature missing from classical mathematics: computation. Cubical Type Theory [5], and its implementation in Cubical Agda [20], realise this property even in the presence of univalence, giving computational content to programs written in HoTT.

1.2 Contributions

In this work we will examine five notions of finiteness in Cubical Type Theory, the relationships between them, and their topos-theoretic characterisation. We also briefly examine a predicate for countable sets, comparing it to the finiteness predicates. Our work is formalised in Cubical Agda, where we also develop a library for proof search based on the finiteness predicates.

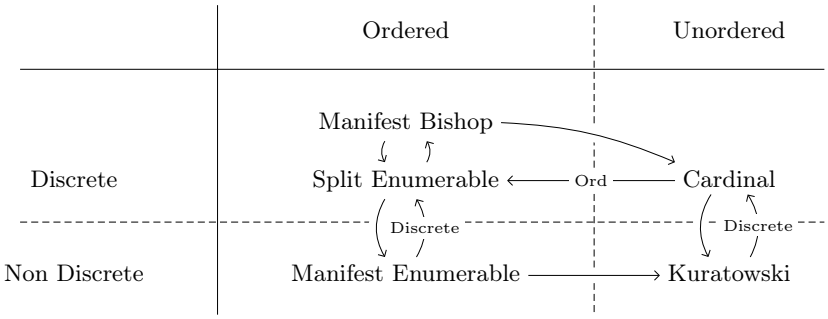

Fig. 1: Classification of finiteness predicates according to whether they are discrete (imply decidable equality) and whether they imply a total order.

The finiteness predicates we are interested in are organised in figure 1, and defined in section 2. We will explore two aspects of each predicate: its relation to the other predicates, and its topos-theoretic classification.

When we say "relation" we are referring to the arrows in figure 1. In section 3 we will provide a function which inhabits each arrow. Each unlabelled arrow is an unconditional implication: every manifest Bishop finite set is cardinal finite (lemma 17), for instance. The labelled arrows are strengthening proofs: every manifest enumerable set *with decidable equality* is split enumerable (lemma 16). Our most significant result here is the proof that a cardinal finite set with a decidable total order is manifestly Bishop finite.

We will then examine the closure properties of each predicate in section 4, culminating in a proof that decidable Kuratowski finite sets form a Π-pretopos (theorem 3). Our proofs follow the structure of [19, Chapters 9, 10] and [18].

After the finite predicates, we will briefly look at the infinite countable types, and classify them in a parallel way to the finite predicates (section 5).

All of our work is formalised in Cubical Agda [20]. We will make mention of the few occasions where the formalisation of some proof is of interest, but the main place where we will discuss the code is in section 6, where we implement a library for proof search, based on omniscience and exhaustibility. While proof search based on finiteness is not new, implementing it here does give us an opportunity to demonstrate some actual *computation* in a univalent setting. Many of the finiteness predicates are built using univalence, so for the proof search functionality to work at all we require computational content in the univalence axiom. Furthermore, the extensionality afforded to us by HoTT means that we can have Π types in the domain of the search, making our library more powerful than comparable libraries. We will demonstrate some interesting uses for this feature in particular.

1.3 Notation and Background

Notation We work in Cubical Type Theory [5]. For the various type formers we use the following notation:

Type We use **Type** to denote the universe of (small) types. "Type families" are functions into **Type**.

Universes To avoid Girard's paradox [12] we use explicit universe levels. A full explanation is beyond the scope of this work, but it means that there is a hierarchy of types starting with **Type**.

$$\begin{aligned} 2 &: \mathbb{N} \\ \mathbb{N} &: \mathbf{Type} \\ \mathbf{Type} &: \mathbf{Type}_1 \\ \mathbf{Type}_1 &: \mathbf{Type}_2 \end{aligned} \tag{1}$$

In our formalisation, every proof is done in the most universe polymorphic way possible: the proof that the cardinally finite sets form a Π-pretopos, for instance, is defined over any universe level.

0 , 1 , 2 We call the **0**, **1**, and **2** types $\bot$, $\top$, and **Bool** respectively. The single inhabitant of $\top$ is tt, and the two inhabitants of **Bool** are false and true. The "negation" of a type, written $\neg A$, means $A \to \bot$.

Dependent Sum and Product We use Σ and Π for the dependent sum and product, respectively. The two projections from Σ are called fst and snd. In the non-dependent case, Σ can be written as $\times$, and Π as $\to$.

Disjoint Union Disjoint union can be defined in terms of Σ:

$$A \uplus B \coloneqq \Sigma(x : \mathbf{Bool}), \text{if } x \text{ then } A \text{ else } B \tag{2}$$

However, we prefer to use it as an inductively defined type (though the two are equivalent).

$$\begin{aligned} A \uplus B \coloneqq \text{inl} &: A \to A \uplus B \\ \mid \text{inr} &: B \to A \uplus B \end{aligned} \tag{3}$$

Equalities, equivalences, and paths We use the symbol $\coloneqq$ for definitions. $\simeq$ will be used for equivalences, and $\equiv$ for equalities. Of course, we know that $(A \simeq B) \simeq (A \equiv B)$ by univalence, so the distinction isn't terribly important in usage: we will only use one or the other as a suggestion of how we constructed it or how it is to be used.

Cubical Type Theory Cubical Type Theory [5] is a constructive type theory with an implementation in Cubical Agda [20]. It allows us to do much of the same theory as in HoTT, but crucially the univalence "axiom" is a *theorem*, rather than an axiom. This allows us to actually compute with univalent proofs, a capability missing from HoTT.

Definition 1 (Path Types)**.** The equality type (which we denote with $\equiv$) in CuTT is the type of Paths[3]. The internal structure of paths is largely irrelevant to us here, as we will generally treat $\equiv$ as a black-box equivalence relation with substitution and congruence.

Definition 2 (Homotopy Levels)**.** Types in HoTT and CuTT are not necessarily sets, as they are in MLTT. This means that equalities are not necessarily unique. In fact, we have an entire hierarchy of homotopy levels, starting with contractions, of which sets are level 2.

$$\begin{aligned}
\mathrm{isContr}(A) &\coloneqq \Sigma(x : A), \Pi(y : A), (x \equiv y) && (4)\\
\mathrm{isProp}(A) &\coloneqq \Pi(x, y : A), (x \equiv y) && (5)\\
\mathrm{isSet}(A) &\coloneqq \Pi(x, y : A), \mathrm{isProp}(x \equiv y) && (6)\\
\mathrm{isGroupoid}(A) &\coloneqq \Pi(x, y : A), \mathrm{isSet}(x \equiv y) && (7)
\end{aligned}$$

We can define the above types inductively like so:

$$\begin{aligned}
\mathrm{isOfHLevel}(0, A) &\coloneqq \mathrm{isContr}(A) && (8)\\
\mathrm{isOfHLevel}(n + 1, A) &\coloneqq \Pi(x, y : A), \mathrm{isOfHLevel}(n, x \equiv y) && (9)
\end{aligned}$$

This type is defined similarly in [19, definition 7.1.1], although there the numbering starts at -2.

Definition 3 (Fibres)**.** A fibre [19, definition 4.2.4] is defined over some function $f : A \to B$.

$$\mathrm{fib}_f(y) = \Sigma(x : A), (f(x) \equiv y) \tag{10}$$

Definition 4 (Equivalences)**.** We will take contractible maps [19, definition 4.4.1] as our "default" definition of equivalences.

$$\begin{aligned}
\mathrm{isEquiv}(f) &\coloneqq \Pi(y : B), \mathrm{isContr}(\mathrm{fib}_f(y)) && (11)\\
A \simeq B &\coloneqq \Sigma(f : A \to B), \mathrm{isEquiv}(f) && (12)
\end{aligned}$$

[3] Actually, CuTT does have an identity type with similar semantics to the identity type in MLTT. We do not use this type anywhere in our work, however, so we will not consider it here.

Lemma 1. Univalence

$$(A \simeq B) \simeq (A \equiv B) \tag{13}$$

This is the so-called univalence "axiom", which is a theorem in CuTT.

Definition 5 (Decision)**.**

$$\mathbf{Dec}(A) \coloneqq A \uplus \neg A \tag{14}$$

Definition 6 (Discrete)**.** A discrete type is one with decidable equality.

$$\text{Discrete}(A) \coloneqq \Pi(x, y : A), \mathbf{Dec}(x \equiv y) \tag{15}$$

By Hedberg's theorem [13] any discrete type is a set.

Definition 7 (Higher Inductive Types)**.** Normal inductive types have *point* constructors: constructors which construct values of the type. Higher Inductive Types (HITs) also have *path* constructors: ways to construct paths in the type.

Definition 8 (Propositional Truncation)**.** The type $\|A\|$ on some type A is a propositionally truncated proof of A [19, 3.7]. In other words, it is a proof that some A exists, but it does not tell you *which* A.

It is defined as a Higher Inductive Type:

$$\begin{aligned} \|A\| \coloneqq |\cdot| \quad & : A \to \|A\|; \\ | \text{ squash} \quad & : \Pi(x, y : \|A\|), x \equiv y; \end{aligned} \tag{16}$$

We will use three eliminators from $\|A\|$ in this paper.

1. For any function $A \to B$, where $\text{isProp}(B)$, we have a function $\|A\| \to B$.
2. A special case of 1 implies that $\|\cdot\|$ forms a monad: this means that we get the usual Monadic operators on $\|\cdot\|$ (bind, pure, fmap, etc.).
3. We can eliminate from $\|A\|$ with a function $f : A \to B$ iff f "doesn't care" about the choice of A ($\Pi(x, y : A), f(x) \equiv f(y)$). Formally speaking, f needs to be "coherently constant" [15], and B needs to be an n-type for some finite n.

2 Finiteness Predicates

In this section, we will define and briefly describe each of the five predicates in Figure 1. The reason we explore predicates other than our focus (cardinal finiteness) is that we can often prove things like closure much more readily on the simpler predicates. The relations (which we will prove in the next section) then allow us to transfer those proofs onto Kuratowski finiteness.

2.1 Split Enumerability

Definition 9 (Split Enumerable Set)**.**

$$\mathcal{E}!(A) \coloneqq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in xs \tag{17}$$

We call the first component of this pair the "support" list, and the second component the "cover" proof. An equivalent version of this predicate was called `Listable` in [9].

We used some extra types in the above definition, which we will define here:

Definition 10 (Containers)**.** A container [1] is a pair S, P where S is a type, the elements of which are called the *shapes* of the container, and P is a type family on S, where the elements of $P(s)$ are called the *positions* of a container. We "interpret" a container into a functor defined like so:

$$[\![S, P]\!](A) \coloneqq \Sigma(s : S), (P(s) \to A) \tag{18}$$

Membership of a container can be defined like so:

$$x \in xs \coloneqq \mathrm{fib}_{\mathrm{snd}(xs)}(x) \tag{19}$$

Definition 11 (**List**)**.**

$$\mathbf{List} \coloneqq [\![\mathbb{N}, \mathbf{Fin}]\!] \tag{20}$$

Definition 12 (**Fin**)**.** $\mathbf{Fin}(n)$ is the type of natural numbers smaller than n. We define it the standard way, where $\mathbf{Fin}(0) \coloneqq \bot$ and $\mathbf{Fin}(n+1) \coloneqq \top \uplus \mathbf{Fin}(n)$.

We tend to prefer list-based definitions of finiteness, rather than ones based on bijections or surjections. This is purely a matter of perspective, however: the definition above is precisely equivalent to a split surjection from a finite prefix of the natural numbers.

Definition 13 (Surjections)**.** We define both surjections and *split* surjections here [19, definition 4.6.1].

$$\mathrm{surj}(f) \quad \coloneqq \Pi(y : B), \|\mathrm{fib}_f(y)\| \tag{21}$$
$$A \twoheadrightarrow B \quad \coloneqq \Sigma(f : A \to B), \mathrm{surj}(f) \tag{22}$$
$$\text{sp-surj}(f) \coloneqq \Pi(y : B), \mathrm{fib}_f(y) \tag{23}$$
$$A \twoheadrightarrow! B \quad \coloneqq \Sigma(f : A \to B), \text{sp-surj}(f) \tag{24}$$

Lemma 2.

$$\mathcal{E}!(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow! A) \tag{25}$$

Proof.

$$\begin{aligned}
\mathcal{E}!(A) &\simeq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in xs && \text{def. 9 } (\mathcal{E}!)\\
&\simeq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), \mathrm{fib}_{\mathrm{snd}(xs)}(x) && \text{eqn. 19 } (\in)\\
&\simeq \Sigma(xs : \mathbf{List}(A)), \text{sp-surj}(\mathrm{snd}(xs)) && \text{eqn. 23 (sp-surj)}\\
&\simeq \Sigma(xs : [\![\mathbb{N}, \mathbf{Fin}]\!](A)), \text{sp-surj}(\mathrm{snd}(xs)) && \text{def. 11 } (\mathbf{List})\\
&\simeq \Sigma(xs : \Sigma(n : \mathbb{N}), \Pi(i : \mathbf{Fin}(n)), A), \text{sp-surj}(\mathrm{snd}(xs)) && \text{eqn. 18 } ([\![\cdot]\!])\\
&\simeq \Sigma(n : \mathbb{N}), \Sigma(f : \mathbf{Fin}(n) \to A), \text{sp-surj}(f) && \text{Reassociation of } \Sigma\\
&\simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow! A) && \text{eqn. 24 } (\twoheadrightarrow!)
\end{aligned}$$

■

In our formalisation, the proof is a single line: most of the steps above are simple expansion of definitions. The only step which isn't definitional equality is the reassociation of Σ.

Split enumerability implies decidable equality on the underlying type. To prove this, we will make use of the following lemma, proven in the formalisation:

Lemma 3.

$$\frac{A \twoheadrightarrow! B \quad \text{Discrete}(A)}{\text{Discrete}(B)} \tag{26}$$

Lemma 4. Every split enumerable type is discrete.

Proof. Let A be a split enumerable type. By lemma 2, there is a surjection from $\mathbf{Fin}(n)$ for some n. Also, we know that $\mathbf{Fin}(n)$ is discrete (proven in our formalisation). Therefore, by lemma 3, A is discrete. ■

2.2 Manifest Bishop Finiteness

Definition 14 (Manifest Bishop Finiteness)**.**

$$\mathcal{B}(A) \coloneqq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in! xs \tag{27}$$

The only difference between manifest Bishop finiteness and split enumerability is the membership term: here we require unique membership ($\in!$), rather than simple membership ($\in$).

Definition 15 (Unique Membership)**.**

$$x \in! xs \coloneqq \text{isContr}(x \in xs) \tag{28}$$

We use the word "manifest" here to distinguish from another common interpretation of Bishop finiteness, which we have called Cardinal finiteness in this paper. The "manifest" refers to the fact that we have a concrete, non-truncated list of the elements in the proof.

Where split enumerability was the enumeration form of a split surjection from **Fin**, manifest Bishop finiteness is the enumeration form of an *equivalence* with **Fin**.

Lemma 5.

$$\mathcal{B}(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \simeq A) \tag{29}$$

Proof.

$$\begin{aligned}
\mathcal{B}(A) &\simeq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in! \; xs && \text{def. 27 } (\mathcal{B}) \\
&\simeq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), \mathrm{isContr}\left(\mathrm{fib}_{\mathrm{snd}(xs)}(x)\right) && \text{def. 15 } (\in!) \\
&\simeq \Sigma(xs : \mathbf{List}(A)), \mathrm{isEquiv}(\mathrm{snd}(xs)) && \text{eqn. 11 (isEquiv)} \\
&\simeq \Sigma(xs : [\![\mathbb{N}, \mathbf{Fin}]\!](A)), \mathrm{isEquiv}(\mathrm{snd}(xs)) && \text{def. 11 } (\mathbf{List}) \\
&\simeq \Sigma(xs : \Sigma(n : \mathbb{N}), \Pi(i : \mathbf{Fin}(n)), A), \mathrm{isEquiv}(\mathrm{snd}(xs)) && \text{eqn. 18 } ([\![\cdot]\!]) \\
&\simeq \Sigma(n : \mathbb{N}), \Sigma(f : \mathbf{Fin}(n) \to A), \mathrm{isEquiv}(f) && \text{Reassociation of } \Sigma \\
&\simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \simeq A) && \text{eqn. 22 } (\twoheadrightarrow)
\end{aligned}$$

■

2.3 Cardinal Finiteness

Each finiteness predicate so far has contained an *ordering* of the underlying type. For our purposes, this is too much information: it means that when constructing the "category of finite sets" later on, instead of each type having one canonical representative, it will have $n!$, where n is the cardinality of the type[4].

To remedy the problem, we will use propositional truncation (def. 8).

Definition 16 (Cardinal Finiteness)**.**

$$\mathcal{C}(A) \coloneqq \|\mathcal{B}(A)\| \simeq \|\Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \simeq A)\| \tag{30}$$

At first glance, it might seem that we lose any useful properties we could derive from $\mathcal{B}$. Luckily, this is not the case: by eliminator 3 of def. 8, we need only show that the output is uniquely determined.

The following two lemmas are proven in [21] (Proposition 2.4.9 and 2.4.10, respectively), in much the same way as we have done here. Our contribution for this section is simply the formalisation.

Lemma 6. Given a cardinally finite type, we can derive the type's cardinality, as well as a propositionally truncated proof of equivalence with **Fin**s of the same cardinality.

$$\mathcal{C}(A) \to \Sigma(n : \mathbb{N}), \|\mathbf{Fin}(n) \simeq A\| \tag{31}$$

Lemma 7. Any cardinal-finite set has decidable equality.

[4] We actually do get a category (a groupoid, even) from manifest Bishop finiteness [21]: it's the groupoid of finite sets equipped with a linear order, whose morphisms are order-preserving bijections. We do not explore this particular construction in any detail.

2.4 Manifest Enumerability

Definition 17 (Manifest Enumerability)**.**

$$\mathcal{E}(A) \coloneqq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), \|x \in xs\| \tag{32}$$

As with manifest Bishop finiteness, the only difference with this type and split enumerability is the membership proof: here we have propositionally truncated it. This has two effects. First, it means that this proof represents a true surjection (rather than a split surjection) from **Fin**.

Lemma 8.

$$\mathcal{E}(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow A) \tag{33}$$

Secondly, it means the predicate does not imply decidable equality. More significantly, it allows the predicate to be defined over non-set types, like the circle.

Definition 18 (S^1)**.** The circle, S^1, can be represented in HoTT as a higher inductive type.

$$\begin{aligned} S^1 \coloneqq \text{base} \quad &: S^1; \\ \mid \text{loop} \quad &: \text{base} \equiv \text{base}; \end{aligned} \tag{34}$$

We will use it here as an example of a non-set type, i.e. a type for which not all paths are equal. This also means that it does not have decidable equality.

Lemma 9. The circle S^1 is manifestly enumerable.

2.5 Kuratowski Finiteness

Much work has already been done on Kuratowski finiteness in HoTT in [11]. As a result, we will not needlessly repeat proofs, rather we will just give a brief introduction to the topic and point out where our treatment differs.

The first thing we must define is a representation of subsets.

Definition 19 (Kuratowski Finite Subset)**.** $\mathcal{K}(A)$ is the type of Kuratowski-finite subsets of A.

$$\begin{aligned} \mathcal{K}(A) \coloneqq [] \quad &: \mathcal{K}(A); \\ \mid \cdot :: \cdot \quad &: A \to \mathcal{K}(A) \to \mathcal{K}(A); \\ \mid \text{com} \quad &: \Pi(x, y : A), \Pi(xs : \mathcal{K}(A)), x :: y :: xs \equiv y :: x :: xs; \\ \mid \text{dup} \quad &: \Pi(x : A), \Pi(xs : \mathcal{K}(A)), x :: x :: xs \equiv x :: xs; \\ \mid \text{trunc} \quad &: \Pi(xs, ys : \mathcal{K}(A)), \Pi(p, q : xs \equiv ys), p \equiv q; \end{aligned} \tag{35}$$

We define it as a HIT (definition 7). The first two constructors are point constructors, giving ways to create values of type $\mathcal{K}(A)$. They are also recognisable as the two constructors for finite lists, a type which represents the free monoid.

The next two constructors add extra paths to the type: equations that usage of the type must obey. These extra paths turn the free monoid into the free *commutative* (com) *idempotent* (dup) monoid.

The final constructor enforces that the type $\mathcal{K}(A)$ must be a set.

The Kuratowski finite subset is a free join semilattice (or, equivalently, a free commutative idempotent monoid). More prosaically, $\mathcal{K}$ is the abstract data type for finite sets, as defined in the Boom hierarchy [3, 4]. However, rather than just being a specification, $\mathcal{K}$ is fully usable as a data type in its own right, thanks to HITs.

Other definitions of $\mathcal{K}$ exist (such as the one in [11]) which make the fact that $\mathcal{K}$ is the free join semilattice more obvious. We have included such a definition in our formalisation, and proven it equivalent to the one above.

Next, we need a way to say that an entire type is Kuratowski finite. For that, we will need to define membership of $\mathcal{K}$.

Definition 20 (Membership of $\mathcal{K}$)**.** Membership is defined by pattern-matching on $\mathcal{K}$. The two point constructors are handled like so:

$$\begin{aligned} x \in \quad [] &\coloneqq \bot \\ x \in y :: ys &\coloneqq \|x \equiv y \uplus x \in ys\| \end{aligned} \tag{36}$$

The com and dup constructors are handled by proving that the truncated form of $\uplus$ is itself commutative and idempotent. The type of propositions is itself a set, satisfying the trunc constructor.

Finally, we have enough background to define Kuratowski finiteness.

Definition 21 (Kuratowski Finiteness)**.**

$$\mathcal{K}^f(A) = \Sigma(xs : \mathcal{K}(A)), \Pi(x : A), x \in xs \tag{37}$$

We also have the following two lemmas, proven in both [11] and our formalisation.

Lemma 10. $\mathcal{K}^f$ is a mere proposition.

Lemma 11. This circle S^1 is Kuratowski finite.

The second of these in particular tells us that the "Kuratowski-finite types" are not necessarily sets; it also tells us that we cannot derive decidable equality from a proof of Kuratowski finiteness.

3 Relations Between Each Finiteness Definition

We will now look at the arrows in figure 1.

3.1 Split Enumerability and Manifest Bishop Finiteness

While manifest Bishop finiteness might seem stronger than split enumerability, it turns out this is not the case. Both types imply the other.

Lemma 12. Any manifest Bishop finite type is split enumerable.

Proof. To construct a proof of split enumerability from one of manifest Bishop finiteness, it suffices to convert a proof of $x \in! xs$ to one of $x \in xs$, for all x and xs. Since $\in!$ is defined as a contraction of $\in$, such a conversion is simply the fst function. ■

To derive $\mathcal{B}$ from $\mathcal{E}!$ takes significantly more work. The "unique membership" condition in $\mathcal{B}$ means that we are not permitted duplicates in the support list. The first step in the proof, then, is to filter those duplicates out from the support list of the $\mathcal{E}!$ proof: we can do this using the decidable equality provided by $\mathcal{E}!$ (lemma 4). From there, all we need to show is that the membership proof carries over appropriately.

Lemma 13. Any split enumerable set is manifest Bishop finite.

3.2 Split Enumerability and Manifest Enumerability

Lemma 14. Any split enumerable type is manifestly enumerable.

Lemma 15. For any type A, we can extract a value of type A from a decision over A in combination with a propositionally-truncated value of type $\|A\|$.

$$\frac{\mathbf{Dec}(A) \quad \|A\|}{A} \tag{38}$$

Proof. Let d be a value of type $\mathbf{Dec}(A)$, and p be a value of type $\|A\|$. We proceed by case analysis on d. In the case where d is inl, we have a value of type A, and our goal is satisfied. In the case of inr, we have a value of type $\neg A$. Using eliminator 1 from definition 8, and the fact that $\bot$ is a proposition, we can run the $\neg A$ proof on p, getting a contradiction. ■

Lemma 16. A manifestly enumerable type with decidable equality is split enumerable.

Proof. Similarly to the proof of lemma 12, our obligation for this proof is to provide a conversion function between membership proofs. In this case, we need to show:

$$\Pi(x : A), \Pi(xs : \mathbf{List}(A)), \|x \in xs\| \to x \in xs \tag{39}$$

This conversion function can be constructed from the two following functions, provided in the formalisation. First, for any type with decidable equality, decidable membership in a finite list is consequently decidable:

$$\text{Discrete}(A) \to \Pi(x : A), \Pi(xs : \mathbf{List}(A)), \mathbf{Dec}(x \in xs) \tag{40}$$

Then, we use lemma 15, allowing us to derive $x \in xs$ from $\|x \in xs\|$. ■

3.3 Manifest Bishop Finiteness and Cardinal Finiteness

Lemma 17. Any manifest Bishop finite type is cardinal finite.

Theorem 1. Any cardinal finite type with a total order is Bishop finite.

The proof for this particular theorem is quite involved in the formalisation, so we only give its sketch here. First, note that we actually convert to manifest enumerability first: this can be converted to split enumerability with decidable equality, which is provided by cardinal finiteness.

Next, we define permutations.

Definition 22 (List Permutations)**.** Two lists are permutations of each other if their membership proofs are all equivalent[5] [7].

$$xs \leftrightsquigarrow ys = \Pi(x : A), x \in xs \simeq x \in ys \tag{41}$$

Next, we define a sort function which relies on the provided total order. We further prove the following fact about this sort function:

$$\Pi(xs, ys : \mathbf{List}(A)), xs \leftrightsquigarrow ys \to \mathrm{sort}(xs) \equiv \mathrm{sort}(ys) \tag{42}$$

Next, notice that the support lists of any two proofs of manifest Bishop finiteness must be permutations of each other. This will allow us to sort the support list of a proof of cardinal finiteness in a coherently constant (definition 8, eliminator 3) way, pulling the support list out from the truncation. The cover proof emerges naturally from the definition of the permutation.

3.4 Cardinal Finiteness and Kuratowski Finiteness

Lemma 18.

$$\mathcal{C}(A) \simeq \mathcal{K}(A) \times \mathrm{Discrete}(A) \tag{43}$$

This proof is constructed by providing a pair of functions: one from $\mathcal{C}(A)$ to $\mathcal{K}(A) \times \mathrm{Discrete}(A)$, and one the other way. This pair implies an equivalence, because both source and target are propositions. The actual functions themselves are proven in our formalisation, as well as in [11].

4 Topos

In this section we will prove that decidable Kuratowski finite types form a Π-pretopos. Along the way we will provide closure proofs for a number of the other finiteness predicates. As we saw in theorem 18, decidable Kuratowski finite types are equivalent to cardinal finite types, so we will work with the latter from now on. Our first task is to show that cardinal finite types are closed under several operations.

[5] The definition in [7] and our formalisation is slightly different: we say permutations are lists with *isomorphic* membership proofs. The distinction, as it happens, does not affect our work here.

4.1 Closure

For the first three closure proofs, we only consider split enumerability: as it is the strongest of the finiteness predicates, we can derive the other closure proofs from it.

Lemma 19. $\bot$, $\top$, and **Bool** are split enumerable.

Lemma 20. Split enumerability is closed under Σ.

$$\frac{\mathcal{E}!(A) \quad \Pi(x : A), \mathcal{E}!(U(x))}{\mathcal{E}!(\Sigma(x : A), U(x))} \tag{44}$$

From this we can derive split enumerability of non-dependent sums and products, as both are definable in terms of Σ.

Lemma 21. Split enumerability is closed under dependent functions. (Π-types).

$$\frac{\mathcal{E}!(A) \quad \Pi(x : A), \mathcal{E}!\,(U(x))}{\mathcal{E}!\,(\Pi(x : A), U(x))} \tag{45}$$

Proof. Let A be a split enumerable type, and U be a type family from A, which is split enumerable over all points of A.

As A is split enumerable, we know that it is also manifestly Bishop finite (lemma 13), and consequently we know $A \simeq \mathbf{Fin}(n)$, for some n (lemma 5). We can therefore replace all occurrences of A with $\mathbf{Fin}(n)$, changing our goal to:

$$\frac{\mathcal{E}!(\mathbf{Fin}(n)) \quad \Pi(x : \mathbf{Fin}(n)), \mathcal{E}!\,(U(x))}{\mathcal{E}!\,(\Pi(x : \mathbf{Fin}(n)), U(x))} \tag{46}$$

We then define the type of n-tuples over some type family $T : \mathbf{Fin}(n) \to \mathbf{Type}$.

$$\begin{aligned} \mathbf{Tuple}(0, T) &:= \top \\ \mathbf{Tuple}(n+1, T) &:= T(0) \times \mathbf{Tuple}(n, T \circ \mathrm{suc}) \end{aligned} \tag{47}$$

We can show that this type is equivalent to functions (proven in our formalisation):

$$\Pi(x : \mathbf{Fin}(n)), U(x) \simeq \mathbf{Tuple}(n, U) \tag{48}$$

And therefore we can simplify again our goal to the following:

$$\frac{\mathcal{E}!(\mathbf{Fin}(n)) \quad \Pi(x : \mathbf{Fin}(n)), \mathcal{E}!\,(U(x))}{\mathcal{E}!\,(\mathbf{Tuple}(n, U))} \tag{49}$$

We can prove this goal by showing that $\mathbf{Tuple}(n, U)$ is split enumerable: it is made up of finitely many products of points of U, which are themselves split enumerable, and $\top$, which is also split enumerable. Lemma 20 shows us that the product of finitely many split enumerable types is itself split enumerable, proving our goal. ∎

The dependent cases of each of these proofs do not transfer simply to proofs on $\mathcal{C}$. In order to transfer them over, we need to prove a variant of the axiom of choice on finite sets.

Lemma 22.

$$\mathcal{C}(A) \to (\Pi(x : A), \|U(x)\|) \to \|\Pi(x : A), U(x)\| \tag{50}$$

This is proven in our formalisation.

4.2 The Category of Finite Sets

HoTT and CuTT seem to be especially suitable settings for formalisations of category theory. The univalence axiom in particular allows us to treat categorical isomorphisms as equalities, saving us from the dreaded "setoid hell".

We follow [19, chapter 9] in its treatment of categories in HoTT, and in its proof that sets do indeed form a category. We will first briefly go through the construction of the category *Set*, as it differs slightly from the usual method in type theory.

First, the type of objects and arrows:

$$\mathrm{Obj}_{Set} \coloneqq \Sigma(x : \mathbf{Type}), \mathrm{isProp}(x) \tag{51}$$

$$\mathrm{Hom}_{Set}(x, y) \coloneqq \mathrm{fst}(x) \to \mathrm{fst}(y) \tag{52}$$

As the type of objects makes clear, we have already departed slightly from the simpler $\mathrm{Obj}_{Set} \coloneqq \mathbf{Type}$ way of doing things: of course we have to, as HoTT allows non-set types. Furthermore, after proving the usual associativity and identity laws for composition (which are definitionally true in this case), we must further show $\mathrm{isSet}(\mathrm{Hom}_{Set}(x, y))$; even then we only have a precategory.

To show that *Set* is a category, we must show that categorical isomorphisms are equivalent to equivalences. In a sense, we must give a univalence rule for the category we are working in.

We have provided formal proofs that *Set* does indeed form a category, and the following:

Theorem 2 (The Category of Finite Sets)**.** Finite sets form a category in HoTT when defined like so:

$$\begin{aligned} \mathrm{Obj}_{FinSet} &\coloneqq \Sigma(x : \mathbf{Type}), \mathcal{C}(x) \\ \mathrm{Hom}_{FinSet}(x, y) &\coloneqq \mathrm{fst}(x) \to \mathrm{fst}(y) \end{aligned} \tag{53}$$

4.3 The Π-pretopos of Finite Sets

For this proof, we follow again the proof that *Set* forms a ΠW-pretopos from [19, chapter 10] and [18]. The difference here is that clearly we do not have access to W-types, as they would permit infinitary structures.

We first must show that *Set* has an initial object and finite, disjoint sums, which are stable under pullback. We also must show that *Set* is a regular category with effective quotients. We now have a pretopos: the presence of Π types make it a Π-pretopos.

We have proven the above statements for both *Set* and *FinSet*. As far as we know, this is the first formalisation of either.

Theorem 3. The category of finite sets, *FinSet*, forms a Π-topos.

5 Countably Infinite Types

In the previous sections we saw different flavours of finiteness which were really just different flavours of relations to **Fin**. In this section we will see that we can construct a similar classification of relations to $\mathbb{N}$, in the form of the countably infinite types.

5.1 Two Countable Types

The two types for countability we will consider are analogous to split enumerability and cardinal finiteness. The change will be a simple one: in terms of surjections and bijections, we will swap out **Fin** for $\mathbb{N}$. In terms of support lists and cover proofs, we will swap out lists for streams. A stream can be defined many ways in type theory: we take rather a low-tech approach here, saying that a stream is a function from the natural numbers.

Definition 23 (Stream)**.**

$$\mathbf{Stream}(A) \coloneqq \mathbb{N} \to A \tag{54}$$

Streams are also definable as containers:

$$\mathbf{Stream}(A) \simeq [\![\top, \mathrm{const}(\mathbb{N})]\!] \tag{55}$$

However the two definitions are clearly equivalent, and furthermore it is simpler to work with the first.

From this, we can define "split countability".

Definition 24 (Split Countability)**.**

$$\mathcal{E}!(A) \coloneqq \Sigma(xs : \mathbf{Stream}(A)), \Pi(x : A), x \in xs \tag{56}$$

This type is definitionally equal to it surjection equivalent $(\mathbb{N} \twoheadrightarrow! A)$. We construct the unordered, propositional version of the predicate in much the same way as we constructed cardinal finiteness.

Definition 25 (Countability)**.**

$$\mathcal{E}(A) \coloneqq \|\mathcal{E}!(A)\| \tag{57}$$

From both of these types we can derive decidable equality.

Lemma 23. Any countable type has decidable equality.

5.2 Closure

We know that countable infinity is not closed under the exponential (function arrow), so the only closure we need to prove is Σ to cover all of what's left.

Theorem 4. Split countability is closed under Σ.

This proof does not mirror the proof of Σ closure on finite types, surprisingly. A different pattern is needed to pair up the members of each support stream: a *Cantor* pairing pattern. Using the simple finite pairing function would diverge, not properly exploring one of the two lists. Instead, our function works in two stages. First, it builds a stream of non-empty lists. From two support streams xs and ys, this intermediate stream $\mathcal{CV}$ is defined like so:

$$\mathcal{CV}_i \coloneqq [(xs_j, ys_k) \mid j, k \in \mathbb{N}; j + k = i] \tag{58}$$

This is then concatenated to give our result.

Finally, while we have lost certain closure proofs by allowing for infinite types, we also *gain* some: in particular the Kleene star.

Theorem 5. Split countability is closed under Kleene star.

$$\mathcal{E}!(A) \to \mathcal{E}!(\mathbf{List}(A)) \tag{59}$$

Again, this proof requires a particular pattern to ensure non divergence. The pattern here is slightly more complicated, but works on the same basic idea, building an intermediate stream $\mathcal{KV}$ of non-empty lists from the input support stream xs.

$$\mathcal{KV}_i \coloneqq [[xs_{j-1} \mid j \in js] \mid js \in \mathbf{List}(\mathbb{N}); \mathrm{sum}(js) = i; 0 \notin js] \tag{60}$$

Again, this is subsequently flattened.

6 Search

6.1 Omniscience

In this section we are interested in restricted forms of the limited principle of omniscience [17].

Definition 26 (Limited Principle of Omniscience)**.** For any type A and predicate P on A, the limited principle of omniscience is as follows:

$$(\Pi(x : A), \mathbf{Dec}(P(x))) \to \mathbf{Dec}\,(\Sigma(x : A), P(x)) \tag{61}$$

In other words, for any decidable predicate the existential quantification of that predicate is also decidable.

The limited principle of omniscience is non-constructive, but individual types can themselves satisfy omniscience. In particular, *finite* types are omniscient.

There is also a universal form of omniscience, which we call exhaustibility.

Definition 27 (Exhaustibility)**.** We say a type A is exhaustible if, for any decidable predicate P on A, the universal quantification of the predicate is decidable.

$$(\Pi(x : A), \mathbf{Dec}(P(x))) \to \mathbf{Dec}\,(\Pi(x : A), P(x)) \tag{62}$$

All of the finiteness predicates we have seen justify exhaustibility. We will only prove it once, then, for the weakest:

Theorem 6. Kuratowski-finite types are exhaustible.

Omniscience is stronger than exhaustibility, as we can derive the latter from the former:

Lemma 24. Any omniscient type is exhaustible.

All of the ordered finiteness predicates imply omniscience. We will again only prove it for the weakest.

Theorem 7. Manifest enumerable types are omniscient.

Finally, we do have a form of omniscience for prop-valued predicates, as they do not care about the chosen representative.

Theorem 8. Kuratowski finite types are omniscient about prop-valued predicates.

6.2 Synthesising Pattern-Matching Proofs

In particular, they can automate large proofs by analysing every possible case. In [9], the Pauli group is used as an example.

```
data Pauli : Type₀ where X Y Z I : Pauli
```

As Pauli has 4 constructors, n-ary functions on Pauli may require up to 4^n cases, making even simple proofs prohibitively verbose.

The alternative is to derive the things we need from $\mathcal{E}!$. From here we can already derive decidable equality, a function which requires 16 cases if implemented manually.

For proof search, the procedure is a well-known one in Agda [8]: we ask for the result of a decision procedure as an *instance argument*, which will demand computation during typechecking.

6.3 Multiple Arguments

The automation machinery above only deals with single-argument predicates. This is not a problem, as we know that we can work with multiple arguments by currying and uncurrying, since all of the finiteness predicates are closed under $\times$. To automate away the curry/uncurry noise we will use instance search, building on [2] to develop a small interface to generic n-ary functions and properties.

Our generic representation can handle dependent Σ and $\prod$ types (rather than their non-dependent counterparts, $\times$ and $\to$). This extension was necessary for our use case: it is mentioned in the paper as the obvious next step. We also implement the curry-uncurry combinators as (verified) isomorphisms.

A full explanation of our implementation is beyond the scope of this work, so we only present the finished interface, which is used like so:

```
assoc-· : ∀ x y z → (x · y) · z ≡ x · (y · z)
assoc-· = ∀↓ⁿ 3 λ x y z → (x · y) · z ≟ x · (y · z)
```

6.4 Synthesising Functions

Finally, thanks to extensionality provided to us by HoTT, and the computation properties provided by CuTT, we can derive decidable equality on functions over finite types. We can also use functions in our proof search. Here, for instance, is a automated procedure which finds the not function on **Bool**, given a specification.

```
not-spec : Σ[ f ∶ (Bool → Bool) ] (f ∘ f ≡ id) × (f ≢ id)
not-spec = ∃↓ⁿ 1 λ f → (f ∘ f ≟ id) && ! (f ≟ id)
```

7 Related Work

The univalent foundations program is the main basis for this work [19]. In particular, our formalisation in section 4 relied heavily on [19, chapter 10], and [18], a paper which contains much of the same material.

In [11] the topic of Kuratowski finite sets in HoTT is studied extensively: we have focused more on the non-truncated versions of finiteness (the "manifest" predicates), and we have provided the missing Π-pretopos proof of decidable Kuratowski finite sets.

[14] provided a starting point for our categorical formalisation: it contains a proof, for instance, that homotopy sets form a category.

Finite sets in a constructive setting has been studied extensively before: In [6] four separate predicates for finiteness were considered (split-enumerable being the only one explored in this work), and [10] explores Noetherianness. [9] explored what we have called split enumerability and manifest Bishop finiteness (although they are stated slightly differently), and they use these to build a library for proof search.

Our formalisation is made possible by Cubical Type Theory [5] and Cubical Agda [20].

Acknowledgement

This work has been supported by the Science Foundation Ireland under the following grant: 13/RC/2D94 to Irish Software Research Centre.

References

1. Abbott, M., Altenkirch, T., Ghani, N.: Containers: Constructing strictly positive types. Theoretical Computer Science **342**(1), 3–27 (Sep 2005). https://doi.org/10.1016/j.tcs.2005.06.002
2. Allais, G.: Generic level polymorphic n-ary functions. In: Proceedings of the 4th ACM SIGPLAN International Workshop on Type-Driven Development - TyDe 2019. pp. 14–26. ACM Press, Berlin, Germany (2019). https://doi.org/10.1145/3331554.3342604
3. Boom, H.J.: Further thoughts on Abstracto. Working Paper ELC-9, IFIP WG 2.1 (1981)
4. Bunkenburg, A.: The Boom Hierarchy. In: O'Donnell, J.T., Hammond, K. (eds.) Functional Programming, Glasgow 1993, pp. 1–8. Workshops in Computing, Springer London (1994). https://doi.org/10.1007/978-1-4471-3236-3_1
5. Cohen, C., Coquand, T., Huber, S., Mörtberg, A.: Cubical Type Theory: A constructive interpretation of the univalence axiom. arXiv:1611.02108 [cs, math] p. 34 (Nov 2016)
6. Coquand, T., Spiwack, A.: Constructively finite? In: Contribuciones Científicas En Honor de Mirian Andrés Gómez. pp. 217–230. Universidad de La Rioja (2010)
7. Danielsson, N.A.: Bag Equivalence via a Proof-Relevant Membership Relation. In: Interactive Theorem Proving. pp. 149–165. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32347-$8_1$1
8. Devriese, D., Piessens, F.: On the bright side of type classes: Instance arguments in Agda. ACM SIGPLAN Notices **46**(9), 143 (Sep 2011). https://doi.org/10.1145/2034574.2034796
9. Firsov, D., Uustalu, T.: Dependently typed programming with finite sets. In: Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming - WGP 2015. pp. 33–44. ACM Press, Vancouver, BC, Canada (2015). https://doi.org/10.1145/2808098.2808102
10. Firsov, D., Uustalu, T., Veltri, N.: Variations on Noetherianness. Electronic Proceedings in Theoretical Computer Science **207**, 76–88 (Apr 2016). https://doi.org/10.4204/EPTCS.207.4
11. Frumin, D., Geuvers, H., Gondelman, L., van der Weide, N.: Finite Sets in Homotopy Type Theory. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 201–214. CPP 2018, ACM, Los Angeles, CA, USA (2018). https://doi.org/10.1145/3167085
12. Girard, J.Y.: Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur. PhD Thesis, PhD thesis, Université Paris VII (1972)
13. Hedberg, M.: A coherence theorem for Martin-Löf's type theory. Journal of Functional Programming **8**(4), 413–436 (Jul 1998). https://doi.org/10.1017/S0956796898003153
14. Iversen, F.H.: Univalent Categories: A Formalization of Category Theory in Cubical Agda. Master's Thesis, Chalmers University of Technology, Göteborg, Sweden (2018)
15. Kraus, N.: The General Universal Property of the Propositional Truncation. arXiv:1411.2682 [math] p. 35 pages (Sep 2015). https://doi.org/10.4230/LIPIcs.TYPES.2014.111
16. Martin-Löf, P.: Intuitionistic Type Theory. Padua (Jun 1980)

17. Myhill, J.: Errett Bishop. Foundations of constructive analysis. McGraw-Hill Book Company, New York, San Francisco, St. Louis, Toronto, London, and Sydney, 1967, xiii + 370 pp. - Errett Bishop. Mathematics as a numerical language. Intuitionism and proof theory, Proceedings of the summer conference at Buffalo N.Y. 1968, edited by A. Kino, J. Myhill, and R. E. Vesley, Studies in logic and the foundations of mathematics, North-Holland Publishing Company, Amsterdam and London1970, pp. 53–71. The Journal of Symbolic Logic **37**(4), 744–747 (Dec 1972). https://doi.org/10.2307/2272421
18. Rijke, E., Spitters, B.: Sets in homotopy type theory. Mathematical Structures in Computer Science **25**(5), 1172–1202 (Jun 2015). https://doi.org/10.1017/S0960129514000553
19. Univalent Foundations Program, T.: Homotopy Type Theory: Univalent Foundations of Mathematics. https://homotopytypetheory.org/book, Institute for Advanced Study (2013)
20. Vezzosi, A., Mörtberg, A., Abel, A.: Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types. Proc. ACM Program. Lang. **3**(ICFP), 87:1–87:29 (Jul 2019). https://doi.org/10.1145/3341691
21. Yorgey, B.A.: Combinatorial Species and Labelled Structures. Ph.D. thesis, University of Pennsylvania, Pennsylvania (Jan 2014)