Finiteness in Cubical Type Theory

DONNACHA OISÍN KIDNEY, University College Cork
GREGORY PROVAN, University College Cork
NICOLAS WU, Imperial College London

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.

Additional Key Words and Phrases: Agda, Homotopy Type Theory, Cubical Type Theory, Dependent Types, Finiteness, Topos, Kuratowski finite

ACM Reference Format:
Donnacha Oisín Kidney, Gregory Provan, and Nicolas Wu. 2020. Finiteness in Cubical Type Theory. In *Proceedings of Workshop on Type-driven Development (TyDe 2020).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 (MLTT) [15] 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 (HoTT) [18] 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.

Manuscript submitted to ACM

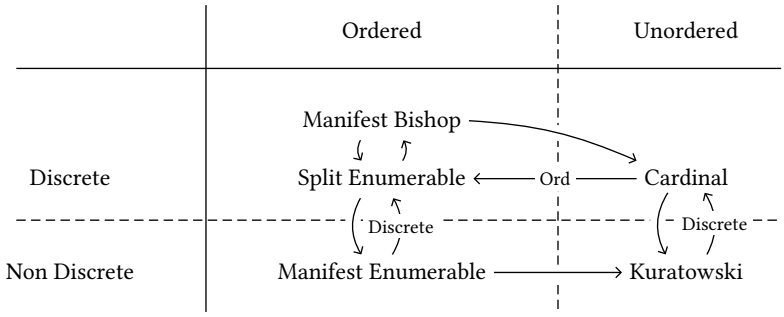

Fig. 1. Classification of finiteness predicates according to whether they are discrete (imply decidable equality) and whether they imply a total order.

Finally, for a computer scientist constructive mathematics has one invaluable feature missing from classical mathematics: computation. Cubical Type Theory (CuTT) [5], and its implementation in Cubical Agda [19], realise this property even in the presence of univalence, giving computational content to programs written in HoTT.

1.2 Contributions

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 3.5), for instance. The labelled arrows are strengthening proofs: every manifest enumerable set *with decidable equality* is split enumerable (lemma 3.4). 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 4.3). Our proofs follow the structure of [18, Chapters 9, 10] and [17].

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 [19]. 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. This library relies directly on the computational content of univalence, which allows us to (for instance) have Π types in the domain of the search space.

The full code of the formalisation is available at https://github.com/oisdk/finiteness-in-cubical-type-theory.

1.3 Notation and Background

1.3.1 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**.

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 ×, and Π as →.

Disjoint Union We define disjoint union as an inductive type.

$$\begin{aligned} A \uplus B \coloneqq \text{inl} \quad &: A \to A \uplus B \\ \mid \text{inr} \quad &: B \to A \uplus B \end{aligned} \tag{1}$$

It is also expressible with only Σ: $A \uplus B \simeq \Sigma(x : \mathbf{Bool}), \text{if } x \text{ then } A \text{ else } B$.

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.

1.3.2 Cubical Type Theory. Cubical Type Theory [5] is a constructive type theory with an implementation in Cubical Agda [19]. It allows us to do much of the same theory as in HoTT, but crucially the univalence "axiom" is a *theorem*, giving it computational content.

Definition 1.1 (Path Types). The equality type (which we denote with $\equiv$) in CuTT is the type of Paths[1]. 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 1.2 (Homotopy Levels). Types in HoTT and CuTT are not necessarily sets, as they are in MLTT. Some have higher homotopies (paths which aren't unique). We actually have a hierarchy of complexity of structure of path spaces in types, starting with the contractions [18, definition 3.11.1], then the mere propositions [18, definition 3.3.1], and the sets [18, definition 3.1.1].

$$\text{isContr}(A) \coloneqq \Sigma(x : A), \Pi(y : A), (x \equiv y) \tag{2}$$

$$\text{isProp}(A) \coloneqq \Pi(x, y : A), (x \equiv y) \tag{3}$$

$$\text{isSet}(A) \coloneqq \Pi(x, y : A), \text{isProp}(x \equiv y) \tag{4}$$

Definition 1.3 (Fibres). A fibre [18, definition 4.2.4] is defined over some function $f : A \to B$.

$$\text{fib}_f(y) \coloneqq \Sigma(x : A), (f(x) \equiv y) \tag{5}$$

Definition 1.4 (Equivalences). We will take contractible maps [18, definition 4.4.1] as our "default" definition of equivalences.

$$\text{isEquiv}(f) \coloneqq \Pi(y : B), \text{isContr}(\text{fib}_f(y)) \tag{6}$$

$$A \simeq B \coloneqq \Sigma(f : A \to B), \text{isEquiv}(f) \tag{7}$$

Definition 1.5 (Decidable Types).

$$\mathbf{Dec}(A) \coloneqq A \uplus \neg A \tag{8}$$

Definition 1.6 (Discrete Types). A discrete type is one with decidable equality.

$$\text{Discrete}(A) \coloneqq \Pi(x, y : A), \mathbf{Dec}(x \equiv y) \tag{9}$$

By Hedberg's theorem [12] any discrete type is a set.

[1] 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.

Definition 1.7 (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 1.8 (Propositional Truncation). The type $\|A\|$ on some type A is a propositionally truncated proof of A [18, p. 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:

$$\begin{aligned} \|A\| \coloneqq |\cdot| &\quad : A \to \|A\|; \\ \mid \text{squash} &\quad : \Pi(x, y : \|A\|), x \equiv y; \end{aligned} \tag{10}$$

We will use two 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) 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" [14], 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 (decidable Kuratowski 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.

2.1 Split Enumerability

Definition 2.1 (Split Enumerable Set).

$$\mathcal{E}!(A) \coloneqq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in xs \tag{11}$$

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 2.2 (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:

$$[\![S, P]\!](A) \coloneqq \Sigma(s : S), (P(s) \to A) \tag{12}$$

Membership of a container can be defined like so:

$$x \in xs \coloneqq \text{fib}_{\text{snd}(xs)}(x) \tag{13}$$

*Definition 2.3 (**List**).*

$$\mathbf{List} \coloneqq [\![\mathbb{N}, \mathbf{Fin}]\!] \tag{14}$$

*Definition 2.4 (**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 2.5 (Surjections). We define both surjections and *split* surjections here [18, definition 4.6.1].

$$\mathrm{surj}(f) \quad :\equiv \Pi(y : B), \|\mathrm{fib}_f(y)\| \tag{15}$$

$$A \twoheadrightarrow B \quad :\equiv \Sigma(f : A \to B), \mathrm{surj}(f) \tag{16}$$

$$\text{sp-surj}(f) :\equiv \Pi(y : B), \mathrm{fib}_f(y) \tag{17}$$

$$A \twoheadrightarrow! B \quad :\equiv \Sigma(f : A \to B), \text{sp-surj}(f) \tag{18}$$

Lemma 2.6.

$$\mathcal{E}!(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow! A) \tag{19}$$

Proof.

$$\begin{aligned}
\mathcal{E}!(A) &\simeq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in xs && \text{def. 2.1 } (\mathcal{E}!)\\
&\simeq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), \mathrm{fib}_{\mathrm{snd}(xs)}(x) && \text{eqn. 13 } (\in)\\
&\simeq \Sigma(xs : \mathbf{List}(A)), \text{sp-surj}(\mathrm{snd}(xs)) && \text{eqn. 17 (sp-surj)}\\
&\simeq \Sigma(xs : [\![\mathbb{N}, \mathbf{Fin}]\!](A)), \text{sp-surj}(\mathrm{snd}(xs)) && \text{def. 2.3 (\textbf{List})}\\
&\simeq \Sigma(xs : \Sigma(n : \mathbb{N}), \Pi(i : \mathbf{Fin}(n)), A), \text{sp-surj}(\mathrm{snd}(xs)) && \text{eqn. 12 } ([\![\cdot]\!])\\
&\simeq \Sigma(n : \mathbb{N}), \Sigma(f : \mathbf{Fin}(n) \to A), \text{sp-surj}(f) && \text{Reassociation of } \Sigma\\
&\simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow! A) && \text{eqn. 18 } (\twoheadrightarrow!)
\end{aligned}$$

■ ■

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 2.7.

$$\frac{A \twoheadrightarrow! B \quad \mathit{Discrete}(A)}{\mathit{Discrete}(B)} \tag{20}$$

Lemma 2.8. *Every split enumerable type is discrete.*

Proof. Let A be a split enumerable type. By lemma 2.6, 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 2.7, A is discrete. ■ ■

2.2 Manifest Bishop Finiteness

Definition 2.9 (Manifest Bishop Finiteness).

$$\mathcal{B}(A) :\equiv \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in! xs \tag{21}$$

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 2.10 (Unique Membership).

$$x \in! \; xs \coloneqq \text{isContr}(x \in xs) \tag{22}$$

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 2.11.

$$\mathcal{B}(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \simeq A) \tag{23}$$

This proof is effectively the same as that of lemma 2.6.

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[2].

To remedy the problem, we will use propositional truncation (def. 1.8).

Definition 2.12 (Cardinal Finiteness).

$$C(A) \coloneqq \|\mathcal{B}(A)\| \simeq \|\Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \simeq A)\| \tag{24}$$

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 2 of def. 1.8, we need only show that the output is uniquely determined.

The following two lemmas are proven in [20] (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 2.13. *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.*

$$C(A) \to \Sigma(n : \mathbb{N}), \|\boldsymbol{Fin}(n) \simeq A\| \tag{25}$$

Lemma 2.14. *Any cardinal-finite set has decidable equality.*

2.4 Manifest Enumerability

Definition 2.15 (Manifest Enumerability).

$$\mathcal{E}(A) \coloneqq \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), \|x \in xs\| \tag{26}$$

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**.

[2]We actually do get a category (a groupoid, even) from manifest Bishop finiteness [20]: 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.

Lemma 2.16.

$$\mathcal{E}(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow A) \tag{27}$$

The proof for this lemma is similar in structure to lemma 2.6 and lemma 2.11.

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

$$\begin{aligned} S^1 &:= \text{base} && : S^1; \\ &\mid \text{loop} && : \text{base} \equiv \text{base}; \end{aligned} \tag{28}$$

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 2.18. *The circle S^1 is manifestly enumerable.*

2.5 Kuratowski Finiteness

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

Definition 2.19 (Kuratowski Finite Subset). $\mathcal{K}(A)$ is the type of Kuratowski-finite subsets of A.

$$\begin{aligned} \mathcal{K}(A) &:= [] && : \mathcal{K}(A); \\ &\mid \cdot :: \cdot && : A \to \mathcal{K}(A) \to \mathcal{K}(A); \\ &\mid \text{com} && : \Pi(x, y : A), \Pi(xs : \mathcal{K}(A)), x :: y :: xs \equiv y :: x :: xs; \\ &\mid \text{dup} && : \Pi(x : A), \Pi(xs : \mathcal{K}(A)), x :: x :: xs \equiv x :: xs; \\ &\mid \text{trunc} && : \Pi(xs, ys : \mathcal{K}(A)), \Pi(p, q : xs \equiv ys), p \equiv q; \end{aligned} \tag{29}$$

We define it as a HIT (definition 1.7). 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 2.20 (Membership of $\mathcal{K}$). Membership is defined by pattern-matching on $\mathcal{K}$. The two point constructors are handled like so:

$$\begin{aligned} x \in \quad [] &:= \bot \\ x \in y :: ys &:= \|x \equiv y \uplus x \in ys\| \end{aligned} \tag{30}$$

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 2.21 (Kuratowski Finiteness).

$$\mathcal{K}^f(A) = \Sigma(xs : \mathcal{K}(A)), \Pi(x : A), x \in xs \tag{31}$$

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

Lemma 2.22. *$\mathcal{K}^f$ is a mere proposition.*

Lemma 2.23. *This circle S^1 is Kuratowski finite.*

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 predicates imply the other.

Lemma 3.1. *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. ■ ■

Lemma 3.2. *Any split enumerable set is manifest Bishop finite.*

This proof 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 2.8). From there, all we need to show is that the membership proof carries over appropriately.

3.2 Split Enumerability and Manifest Enumerability

Lemma 3.3. *Any split enumerable type is manifestly enumerable.*

This lemma is proven by truncating the membership proof in split enumerability.

Lemma 3.4. *A manifestly enumerable type with decidable equality is split enumerable.*

3.3 Manifest Bishop Finiteness and Cardinal Finiteness

Lemma 3.5. *Any manifest Bishop finite type is cardinal finite.*

Theorem 3.6. *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 3.7 (List Permutations). Two lists are permutations of each other if their membership proofs are all equivalent[3] [7].

$$xs \leftrightsquigarrow ys = \Pi(x : A), x \in xs \simeq x \in ys \tag{32}$$

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

$$\Pi(xs, ys : \mathbf{List}(A)), xs \leftrightsquigarrow ys \to \mathrm{sort}(xs) \equiv \mathrm{sort}(ys) \tag{33}$$

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 1.8, eliminator 2) 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 3.8.

$$C(A) \simeq \mathcal{K}^f(A) \times \mathit{Discrete}(A) \tag{34}$$

This proof is constructed by providing a pair of functions: one from $C(A)$ to $\mathcal{K}^f(A) \times \mathrm{Discrete}(A)$, 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 3.7, 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.

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

Lemma 4.2. *Split enumerability is closed under* Σ.

$$\frac{\mathcal{E}!(A) \quad \Pi(x : A), \mathcal{E}!(U(x))}{\mathcal{E}!(\Sigma(x : A), U(x))} \tag{35}$$

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

[3] 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.

Lemma 4.3. *Split enumerability is closed under dependent functions. (Π-types).*

$$\frac{\mathcal{E}!(A) \quad \Pi(x : A), \mathcal{E}!\,(U(x))}{\mathcal{E}!\,(\Pi(x : A), U(x))} \tag{36}$$

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 3.2), and consequently we know $A \simeq \mathbf{Fin}(n)$, for some n (lemma 2.11). We can therefore replace all occurrences of A with $\mathbf{Fin}(n)$, changing our goal to:

$$\frac{\mathcal{E}!(\mathbf{Fin}(n)) \quad \Pi(x : \mathbf{Fin}(n)), \mathcal{E}!\,(U(x))}{\mathcal{E}!\,(\Pi(x : \mathbf{Fin}(n)), U(x))} \tag{37}$$

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

$$\begin{aligned} \mathbf{Tuple}(0, T) &\coloneqq \top \\ \mathbf{Tuple}(n + 1, T) &\coloneqq T(0) \times \mathbf{Tuple}(n, T \circ \mathrm{suc}) \end{aligned} \tag{38}$$

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

$$\Pi(x : \mathbf{Fin}(n)), U(x) \simeq \mathbf{Tuple}(n, U) \tag{39}$$

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

$$\frac{\mathcal{E}!(\mathbf{Fin}(n)) \quad \Pi(x : \mathbf{Fin}(n)), \mathcal{E}!\,(U(x))}{\mathcal{E}!\,(\mathbf{Tuple}(n, U))} \tag{40}$$

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 4.2 shows us that the product of finitely many split enumerable types is itself split enumerable, proving our goal. ■ ■

Lifting the above closure proofs to work on proofs of C is not necessarily straightforward. $\|\cdot\|$ forms a monad, giving us access to a powerful syntax for combining proofs. From this, we can derive closure under $\bot$, $\top$, **Bool**, $\times$, $\uplus$, and $\to$. The dependent case is not so straightforward. We need the following lemma, a version of the axiom of choice on finite sets, to prove closure under Σ and Π.

Lemma 4.4.

$$C(A) \to (\Pi(x : A), \|U(x)\|) \to \|\Pi(x : A), U(x)\| \tag{41}$$

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 [18, 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:

$$\mathrm{Obj}_{Set} \coloneqq \Sigma(x : \mathbf{Type}), \mathrm{isSet}(x) \tag{42}$$

$$\mathrm{Hom}_{Set}(x, y) \coloneqq \mathrm{fst}(x) \to \mathrm{fst}(y) \tag{43}$$

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

$$\begin{aligned} Obj_{FinSet} &\coloneqq \Sigma(x : \mathbf{Type}), C(x) \\ Hom_{FinSet}(x, y) &\coloneqq fst(x) \to fst(y) \end{aligned} \tag{44}$$

4.3 The Π-pretopos of Finite Sets

For this proof, we follow again the proof that *Set* forms a ΠW-pretopos from [18, chapter 10] and [17]. 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 4.6. *The category of finite sets, FinSet, forms a Π-pretopos.*

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: we will swap out lists for streams.

Definition 5.1 (Streams).

$$\mathbf{Stream}(A) \coloneqq (\mathbb{N} \to A) \simeq [\![\top, \mathrm{const}(\mathbb{N})]\!] \tag{45}$$

Definition 5.2 (Split Countability).

$$\mathcal{E}!(A) \coloneqq \Sigma(xs : \mathbf{Stream}(A)), \Pi(x : A), x \in xs \tag{46}$$

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 5.3 (Countability).

$$\mathcal{E}(A) \coloneqq \|\mathcal{E}!(A)\| \tag{47}$$

From both of these types we can derive decidable equality.

Lemma 5.4. *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 5.5. *Split countability is closed under* Σ.

This proof does not mirror the proof of Σ closure on finite types. The pattern we used to pair up the relevant support lists for the finite types would diverge for infinite types: instead, we need a *pairing* function.

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

Theorem 5.6. *Split countability is closed under Kleene star.*

$$\mathcal{E}!(A) \to \mathcal{E}!(\mathbf{List}(A)) \tag{48}$$

Again, this proof requires a particular pattern to ensure productivity. The pattern here builds an intermediate stream $\mathcal{KV}$ of non-empty lists from the input support stream xs, which is subsequently flattened.

$$\mathcal{KV}_i \coloneqq \left[\left[xs_{j-1} \mid j \in js\right] \mid js \in \mathbf{List}(\mathbb{N}); \mathrm{sum}(js) = i; 0 \notin js\right] \tag{49}$$

6 SEARCH

6.1 Omniscience

Definition 6.1 (Limited Principle of Omniscience). For any type A and predicate P on A, the limited principle of omniscience [16] is as follows:

$$(\Pi(x : A), \mathbf{Dec}(P(x))) \to \mathbf{Dec}\,(\Sigma(x : A), P(x)) \tag{50}$$

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, cardinal finite types are omniscient.

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

Definition 6.2 (Exhaustibility). We say a type A is exhaustible if, for any decidable predicate P on A, the universal quantification of the predicate is decidable.

$$(\Pi(x : A), \mathbf{Dec}(P(x))) \to \mathbf{Dec}\,(\Pi(x : A), P(x)) \tag{51}$$

All of the finiteness predicates we have seen imply exhaustibility. Omniscience is stronger than exhaustibility, as we can derive the latter from the former. All of the ordered finiteness predicates imply omniscience. For the unordered finiteness definitions, we have omniscience for prop-valued predicates.

6.2 Automating Proofs

One use for above constructions is the automation of certain proofs. In [9], which uses a similar approach to ours, 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 omniscience, itself derived from a finiteness predicate. 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. Our addition to this technique is a way to handle multiple arguments based on fully level-polymorphic dependent currying and uncurrying, building on [2].

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

Finally, we can derive decidable equality on functions over finite types. We can also use functions in our proof search. Here, for instance, is an 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 = ∃ℓ ⁿ 1 λ f → (f ∘ f ≟ id) && ! (f ≟ id)
```

7 RELATED WORK

The univalent foundations program is the main basis for this work [18]. In particular, our formalisation in section 4 relied heavily on [18, chapter 10], and [17], a paper which contains much of the same material.

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. 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.

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

ACKNOWLEDGMENTS

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

REFERENCES

[1] Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Containers: Constructing Strictly Positive Types". en. In: *Theoretical Computer Science*. Applied Semantics: Selected Topics 342.1 (Sept. 2005), pp. 3–27. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.06.002.

[2] Guillaume Allais. "Generic Level Polymorphic N-Ary Functions". en. In: *Proceedings of the 4th ACM SIGPLAN International Workshop on Type-Driven Development - TyDe 2019*. Berlin, Germany: ACM Press, 2019, pp. 14–26. ISBN: 978-1-4503-6815-5. DOI: 10.1145/3331554.3342604.

[3] H. J. Boom. "Further Thoughts on Abstracto". In: *Working Paper ELC-9, IFIP WG 2.1* (1981).

[4] Alexander Bunkenburg. "The Boom Hierarchy". en. In: *Functional Programming, Glasgow 1993*. Ed. by John T. O'Donnell and Kevin Hammond. Workshops in Computing. Springer London, 1994, pp. 1–8. ISBN: 978-3-540-19879-6 978-1-4471-3236-3. DOI: 10.1007/978-1-4471-3236-3_1.

[5] Cyril Cohen et al. "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom". en. In: *arXiv:1611.02108 [cs, math]* (Nov. 2016), p. 34. arXiv: 1611.02108 [cs, math].

[6] Thierry Coquand and Arnaud Spiwack. "Constructively Finite?" en. In: *Contribuciones Científicas En Honor de Mirian Andrés Gómez*. Universidad de La Rioja, 2010, pp. 217–230.

[7] Nils Anders Danielsson. "Bag Equivalence via a Proof-Relevant Membership Relation". en. In: *Interactive Theorem Proving*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Aug. 2012, pp. 149–165. ISBN: 978-3-642-32346-1 978-3-642-32347-8. DOI: 10.1007/978-3-642-32347-8_11.

[8] Dominique Devriese and Frank Piessens. "On the Bright Side of Type Classes: Instance Arguments in Agda". en. In: *ACM SIGPLAN Notices* 46.9 (Sept. 2011), p. 143. ISSN: 03621340. DOI: 10.1145/2034574.2034796.

[9] Denis Firsov and Tarmo Uustalu. "Dependently Typed Programming with Finite Sets". en. In: *Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming - WGP 2015*. Vancouver, BC, Canada: ACM Press, 2015, pp. 33–44. ISBN: 978-1-4503-3810-3. DOI: 10.1145/2808098.2808102.

[10] Denis Firsov, Tarmo Uustalu, and Niccolò Veltri. "Variations on Noetherianness". In: *Electronic Proceedings in Theoretical Computer Science* 207 (Apr. 2016), pp. 76–88. ISSN: 2075-2180. DOI: 10.4204/EPTCS.207.4. arXiv: 1604.01186.

[11] Dan Frumin et al. "Finite Sets in Homotopy Type Theory". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2018. Los Angeles, CA, USA: ACM, 2018, pp. 201–214. ISBN: 978-1-4503-5586-5. DOI: 10.1145/3167085.

[12] Michael Hedberg. "A Coherence Theorem for Martin-Löf's Type Theory". en. In: *Journal of Functional Programming* 8.4 (July 1998), pp. 413–436. ISSN: 0956-7968, 1469-7653. DOI: 10.1017/S0956796898003153.

[13] Frederik Hanghøj Iversen. "Univalent Categories: A Formalization of Category Theory in Cubical Agda". eng. Master's Thesis. Göteborg, Sweden: Chalmers University of Technology, 2018.

[14] Nicolai Kraus. "The General Universal Property of the Propositional Truncation". en. In: *arXiv:1411.2682 [math]* (Sept. 2015), 35 pages. DOI: 10.4230/LIPIcs.TYPES.2014.111. arXiv: 1411.2682 [math].

[15] Per Martin-Löf. *Intuitionistic Type Theory*. Padua, June 1980.

[16] John Myhill. "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." en. In: *The Journal of Symbolic Logic* 37.4 (Dec. 1972), pp. 744–747. ISSN: 0022-4812, 1943-5886. DOI: 10.2307/2272421.

[17] Egbert Rijke and Bas Spitters. "Sets in Homotopy Type Theory". en. In: *Mathematical Structures in Computer Science* 25.5 (June 2015), pp. 1172–1202. ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129514000553.

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

[19] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. "Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types". In: *Proc. ACM Program. Lang.* 3.ICFP (July 2019), 87:1–87:29. ISSN: 2475-1421. DOI: 10.1145/3341691.

[20] Brent Abraham Yorgey. "Combinatorial Species and Labelled Structures". en. PhD thesis. Pennsylvania: University of Pennsylvania, Jan. 2014.