Finiteness and Cardinality in Homotopy Type Theory

Donnacha Oisín Kidney

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

Abstract. We explore five notions of finiteness in Homotopy Type Theory [12]. We will classify these notions, and prove their closure properties, culminating in a proof that decidable Kuratowski-finite sets [10] form a topos.
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 [13], and we implement a library for proof search (including combinators for level-polymorphic fully generic currying), and demonstrate how it can be used to both prove properties and synthesise full functions given desired properties.

1 Introduction

1.1 Strong Finiteness Predicates

We will first explore the "strong" notions of finiteness (i.e. those at least as strong as Kuratowski finiteness [10]), with a special focus on cardinal finiteness (section 5), and manifest enumerability (section 4), which is new, to our knowledge.

Figure 1 organises the predicates according to their strength; i.e. how much information they provide about a conforming type. For instance, a proof that some type A is manifestly Bishop finite (the strongest of the notions, explored in section 3) also tells us that A is discrete (has decidable equality), and gives us a linear order on the type. A type that is Kuratowski finite (section 6) has no such extra features: indeed, we will see examples of Kuratowski finite types which are not even sets, never mind discrete ones.

We will go through each of the predicates, proving how to weaken each (i.e. we will provide a proof that every cardinally finite type is Kuratowski finite), and how to strengthen them, given the required property. In terms of figure 1, this amounts to providing proofs for each arrow. Our main proof in this regard is that we can derive manifest Bishop finiteness from Kuratowski finiteness plus a total order.

We will—through the use of containers [1]—formally prove the equivalence these predicates have with the usual function relations i.e. we will show that a proof of manifest enumerability is precisely equivalent to a surjection from a finite prefix of the natural numbers.

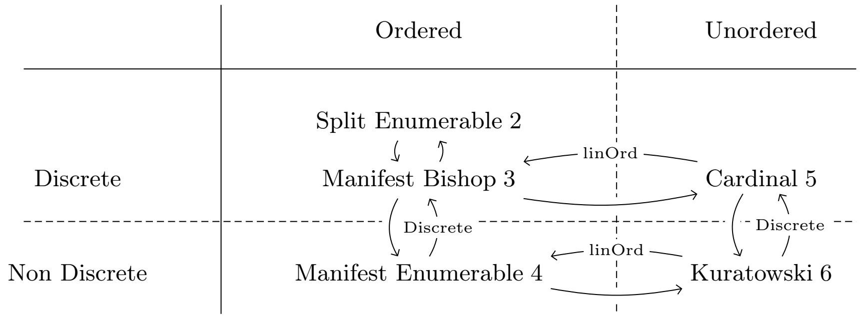


Fig. 1: Classification of the strong definitions of finiteness, according to whether they are discrete (imply decidable equality) and whether they induce a linear order.

For each predicate, we will also prove its closure over sums and products in both dependent and non-dependent forms, if such a closure exists. This will culminate in our main result for this section: the formal proof that decidable Kuratowski finite sets form a topos.

1.2 Infinite Types

In section 7, we will extend our study of finite types to infinite but countable types. We will see that the classification of finiteness predicates mirrors that of their countable counterparts, and we will prove closure under the Kleene star.

1.3 Practical Uses

Finally, in section 8 we will demonstrate some practical uses of the finiteness proofs in Cubical Agda. We will show how to use well-known techniques [5, 6] to automate proofs of functions with finite (and infinite) domains. We further show how to automate the synthesis of functions from desired properties.

2 Split Enumerability

We will start with the simplest definition of finiteness: we say a set is enumerable if there is a list of its elements which contains every element in the set.

Before giving the definition of the predicate, we will first define lists (and membership thereof): we have chosen a *container*-based definition for this work.

Definition 1 (Container)**.** A container [1] is a pair $S \rhd P$ 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:

$$[\![S \rhd P]\!](A) = \Sigma(s : S), (P(s) \to A) \tag{1}$$

Membership of a container can be defined like so:

$$x \in xs = \mathrm{fiber}(\mathrm{snd}(xs), x) \tag{2}$$

Where fiber is from [12, definition 4.2.4].

Definition 2 (Lists)**.**

$$\mathbf{List} = [\![\mathbb{N} \triangleright \mathbf{Fin}]\!] \tag{3}$$

Definition 3 (Fin)**.** $\mathbf{Fin}(n)$ is the type of natural numbers smaller than n. We define it the standard way, where $\mathbf{Fin}(0) = \bot$ and $\mathbf{Fin}(n+1) = \top + \mathbf{Fin}(n)$.

Internally, in our formalisation, we actually use the standard inductive definition of lists more often (it tends to work better in more complex algorithms, and functions on it seems to satisfy the termination checker more readily). However, since both types are equivalent, univalence allows us to transport to whichever representation is more convenient in a given situation. For the higher-level proofs we present here, though, the container-based definition greatly simplifies certain steps, which is why we have chosen it as our representation.

Finally, we can define formally split enumerability.

Definition 4 (Split Enumerable Set)**.**

$$\mathcal{E}!(A) = \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), x \in xs \tag{4}$$

We call the first component of this pair the "support" list, and the second component the "cover" proof.

2.1 Split Surjections

Definition 5 (Surjections)**.**

$$A \twoheadrightarrow B = \Sigma(f : A \to B), \mathrm{surjective}(f) \tag{5}$$

$$A \twoheadrightarrow! \; B = \Sigma(f : A \to B), \text{split surjective}(f) \tag{6}$$

Where the definitions of surjective are taken from [12, definition 4.6.1].

Theorem 1. Split enumerability is equivalent to a split surjection from a finite prefix of the natural numbers.

$$\mathcal{E}!(A) \iff \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \twoheadrightarrow! \; A) \tag{7}$$

Proof. The proof is surprisingly short: after sufficient inlining, it emerges that our goal is simply a reassociation.

```
ℰ!⇔Fin↠! : ℰ! A ⇔ Σ[ n : ℕ ] (Fin n ↠! A)
ℰ!⇔Fin↠! =
  ℰ! A                                    ≋⟨⟩ - ℰ!
  Σ[ xs : List A ] Π[ x : A ] x ∈ xs      ≋⟨⟩ - ∈
```

$$
\begin{aligned}
&\Sigma[\ xs : \mathsf{List}\ A\ ]\ \Pi[\ x : A\ ]\ \mathsf{fiber}\ (xs\ .\mathsf{snd})\ x && \approx\langle\ \mathsf{reassoc}\ \rangle\\
&\Sigma[\ n : \mathbb{N}\ ]\ \Sigma[\ f : (\mathsf{Fin}\ n \to A)\ ]\ \Pi[\ x : A\ ]\ \mathsf{fiber}\ f\ x && \approx\langle\rangle\ \text{-}\ \twoheadrightarrow !\\
&\Sigma[\ n : \mathbb{N}\ ]\ (\mathsf{Fin}\ n \twoheadrightarrow !\ A)\ \blacksquare
\end{aligned}
$$

To be clear: in Agda, the proof could simply be reassoc; we have written out the extra lines for clarity alone.

Lemma 1. Any split enumerable type has decidable equality (is discrete).

Proof. We use a corollary that if there is a split-surjection from A to B, and A is discrete, then B is also discrete.

By Hedberg's theorem [8], we know that this implies split enumerable types must all be sets.

2.2 Closure

In this section we will prove closure under various operations for split enumerable sets. We are working towards a topos proof, which requires us to prove closure under a variety of operations: for now, we only have enough machinery to demonstrate the semiring operations, and dependent sums. In order to show closure under exponentials (function arrows), we will need an equivalence with **Fin**, which will be provided in section 3.

Lemma 2. **Bool**, $\top$, and $\bot$ are all split enumerable.

Proof. Each of these types clearly has a finite number of elements (2, 1, and 0, respectively), and furthermore has a straightforward enumeration.

Theorem 2. Split-enumerability is closed under Σ.

$$\frac{\mathcal{E}!(A) \quad \Pi(x : A), \mathcal{E}!(U(x))}{\mathcal{E}!(\Sigma(x : A), U(x))} \tag{8}$$

Proof. To obtain the support list, we concatenate the support lists of all the proofs of split-finiteness for U over the support list of E_A. In Agda:

```
sup-Σ : List A → (∀ x → List (U x)) → List (Σ A U)
sup-Σ xs ys = do x ← xs
                 y ← ys x
                 [ x , y ]
```

"do-notation" is available to us as we're working in the list monad.

Closure under disjoint union and Cartesian product both follow from Σ, as these types can be defined in terms of Σ.

$$A \times B = \Sigma(x : A), B \tag{9}$$

$$A + B = \Sigma(x : \mathbf{Bool}), \text{if } x \text{ then } A \text{ else } B \tag{10}$$

3 Manifest Bishop Finiteness

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**.

Definition 6 (Manifest Bishop Finiteness)**.**

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

The only difference between this predicate and split enumerability is the list membership term: we use $\in!$ here, where $x \in! \, xs$ is to be read as "x occurs exactly once in xs".

Definition 7 (Unique Membership)**.** We say an item x is "uniquely in" some container xs if its membership in that list is a *contraction* [12, definition 3.11.1]; i.e. its membership proof exists, and all such proofs are equal.

$$x \in! \, xs = \text{isContr}(x \in xs) \tag{12}$$

By the definition of a contraction, we can always recover the underlying membership proof, meaning that we can always derive split enumerability from manifest Bishop finiteness.

3.1 Equivalence

Lemma 3. A proof of manifest Bishop finiteness is equivalent to an equivalence with a finite prefix of the natural numbers.

$$\mathcal{B}(A) \simeq \Sigma(n : \mathbb{N}), (\mathbf{Fin}(n) \simeq A) \tag{13}$$

Proof. There are many equivalent definitions of equivalence in HoTT. Here we take the version preferred in the Cubical Agda library: contractible maps [12, definition 4.4.1]. Because of the parallels between contractible maps and split surjections, the proof proceeds much the same as 1. In other words, the definition of Bishop finiteness is itself a reassociation of a contractible map.

3.2 Relationship to Split Enumerability

We now show that manifest Bishop finiteness has equal strength to split enumerability.

Theorem 3. Any split enumerable set is manifest Bishop finite.

Proof. From proposition 1 we can derive decidable equality on A, and using this we can define a function (called uniques) which filters out duplicates from lists of As. This gives us our support list. To generate the cover proof it suffices now to prove the following:

$$\Pi(x : A), \Pi(xs : \mathbf{List}(A)), x \in xs \to x \in! \, \text{uniques}(xs) \tag{14}$$

3.3 Closure

Proving equal strength of split enumerability and manifest Bishop finiteness allows us to carry all of the previous proofs of closure over to manifest Bishop finite sets (and vice-versa). Missing from our previous proofs was a proof of closure of functions. We remedy that here.

Theorem 4. Manifest bishop finiteness is closed over dependent functions ($\prod$-types).

$$\frac{\mathcal{B}(A) \quad \Pi(x : A), \mathcal{B}\,(U(x))}{\mathcal{B}\,(\Pi(x : A), U(x))} \tag{15}$$

Proof. This proof is essentially the composition of two transport operations, made available to us via univalence.

First, we will simplify things slightly by working only with split enumerability. As this is equal in strength to manifest Bishop finiteness, any closure proofs carry over.

Secondly, we will replace A in all places with $\mathbf{Fin}(n)$. Since we have already seen an equivalence between these two types, we are permitted to transport along these lines. This is the first transport operation.

The bulk of the proof now is concerned with proving the following:

$$(\Pi(x : \mathbf{Fin}(n)), \mathcal{E}!(A(x))) \to \mathcal{E}!\,(\Pi(x : \mathbf{Fin}(n)), A(x)) \tag{16}$$

Our strategy to accomplish this will be to consider functions from $\mathbf{Fin}(n)$ as n-tuples over some type family $T : \mathbb{N} \to \text{Type}$.

$$\begin{aligned} \mathbf{Tuple}(T, 0) &= \top \\ \mathbf{Tuple}(T, n+1) &= T(0) \times \mathbf{Tuple}(T \circ \text{suc}, n) \end{aligned} \tag{17}$$

This type is manifestly Bishop finite, as it is constructed only from products and the unit type.

We then prove an isomorphism between this representation and Π-types.

$$\mathbf{Tuple}(T, n) \iff \Pi(x : \mathbf{Fin}(n)), T(x) \tag{18}$$

This allows us to transport our proof of finiteness on tuples to one on functions from **Fin** (our second transport operation), proving our goal.

4 Manifest Enumerability

Both split enumerability and manifest Bishop finiteness are restricted to sets: because they both imply decidable equality, no non-set types can satisfy them as predicates. To find a more general predicate for finiteness, we *truncate* the membership proof.

Definition 8 (Manifest Enumerability)**.**

$$\mathcal{E}(A) = \Sigma(xs : \mathbf{List}(A)), \Pi(x : A), \|x \in xs\| \tag{19}$$

Definition 9 (Propositional Truncation)**.** a The type $\|A\|$ on some type A is a propositionally truncated proof of A [12, 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:

$$\begin{aligned} \|A\| = {} & |\cdot| : A \to \|A\|; \\ & \mid \text{squash} : \Pi(x, y : \|A\|), x \equiv y; \end{aligned} \tag{20}$$

We will use and consume values of the type $\|A\|$ in three main ways.

1. We can first eliminate from $\|A\|$ into any type which is a proposition. In other words, given a function $f : A \to B$, and a proof that B is a proposition, we can construct a function $\|A\| \to B$.
2. As a consequence of this first point, we can always eliminate into another propositionally truncated type. As a result, $\|\cdot\|$ forms a Monad: for our purposes, this simply means that we can work "under" a propositional truncation in an ergonomic way.
3. We can eliminate from $\|A\|$ with a function $f : A \to B$ iff f "doesn't care" about the choice of A.

$$\Pi(x, y : A), f(x) \equiv f(y) \tag{21}$$

   Formally speaking, f needs to be "coherently constant" [9], and B needs to be an n-type for some finite n.

We will use the circle as an example non-set type which is nonetheless manifestly enumerable.

Theorem 5. The circle S^1 is manifestly enumerable.

Proof. As the cover proof is a truncated proposition, we need only consider the point constructors, making this poof the same as the proof of split enumerability on $\top$.

4.1 Surjections

This predicates relation to surjectivity is much the same as split enumerability's relation to *split* surjectivity.

Lemma 4. A proof of manifest enumerability is equivalent to a surjection from a finite prefix of the natural numbers.

Proof. As with the other surjection proof (lemma 1), this is simply a reassociation.

4.2 Relation to Split Enumerability

Theorem 6. A manifestly enumerable type with decidable equality is split enumerable.

Proof. The support list stays the same between both enumerability proofs.

For the cover proof, we first need the following function which searches a list for a particular element (given decidable equality on A).

$$\in ? : \Pi(x : A), \Pi(xs : \mathbf{List}(A)), \mathbf{Dec}(x \in xs) \tag{22}$$

Where $\mathbf{Dec}(A)$ is a decision on some type A.

We then need to convert a value of type $\mathbf{Dec}(x \in xs)$ to $x \in xs$. We use the following to do that:

$$\text{recompute} : \mathbf{Dec}(A) \to \|A\| \to A \tag{23}$$

The function works via case-analysis on **Dec**: in the true case, we return the proof; in the false case, we can apply the proof of negation under the truncation, and then extract $\|\bot\|$ to $\bot$, as $\bot$ is a mere proposition, thus proving the goal via explosion.

4.3 Closure

Since we don't have an equivalence with **Fin**, we don't get closure under Π.

Lemma 5. Manifest enumerability is closed under Σ.

Proof. This closure proof is almost the same as theorem 2. The manipulation of the support lists can be carried over as-is; but the type of the cover proof has changed, so it will need to be updated. As it happens, the translation is straightforward: we effectively write the "monadic" version of the old function.

5 Cardinal Finiteness

For manifest enumerability, we removed the need for decidable equality: in these next two finiteness predicates, we remove the need for a total order on the underlying type.

Definition 10 (Cardinal Finiteness)**.** A type A is cardinally finite, $\mathcal{C}$, if it has a propositionally-truncated proof of bishop finiteness.

$$\mathcal{C}(A) = \|\mathcal{B}(A)\| \tag{24}$$

5.1 Closure

The closure properties of cardinal finiteness are effectively the non-dependent versions of manifest Bishop finiteness. To see why, consider equation 8. We can "lift" the proof (as a binary function) under a propositional truncation, giving us equation 25, but that doesn't give us the desired closure proof (equation 26).

$$\frac{\|\mathcal{B}(A)\| \quad \|\Pi(x:A),\mathcal{B}(U(x))\|}{\|\mathcal{B}(\Sigma(x:A),U(x))\|} \quad (25) \qquad \frac{\|\mathcal{B}(A)\| \quad \Pi(x:A),\|\mathcal{B}(U(x))\|}{\|\mathcal{B}(\Sigma(x:A),U(x))\|} \quad (26)$$

To remedy the mismatch we would need a function of the type:

$$(\Pi(x:A),\|\mathcal{B}(U(x))\|) \to \|\Pi(x:A),\mathcal{B}(U(x))\| \tag{27}$$

Unfortunately, this equation in particular is a form of the axiom of choice [12, equation 3.8.3]. This leaves us with closure under only the non-dependent operations.

Lemma 6. Cardinal finiteness is closed under $\times$, $+$, and $\to$.

Proof. All of these closure proofs can be lifted directly from their corresponding proofs on manifest Bishop finiteness.

5.2 Decidable Equality

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

Proof. We will use eliminator 1 from definition 9. Manifest Bishop finiteness implies decidable equality already, so our task here is to prove that decidable equality itself is a proposition.

We know that if a type A is a proposition, then the decision over that type is also a proposition. Then, via Hedberg's theorem, we know that any type with decidable equality is a set, meaning that paths in that type are themselves propositions. Therefore we can derive that a decision of equality on elements with decidable equality is a proposition, and by function extensionality we see that decidable equality is itself a proposition.

5.3 Cardinality

Theorem 8. 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.

$$\mathcal{C}(A) \to \Sigma(n:\mathbb{N}),\|\mathbf{Fin}(n) \simeq A\| \tag{28}$$

Proof. Our task here is to "pull out" the cardinality of the set from under the propositional truncation. In effect, we need the following function:

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

We will use eliminator 3 from definition 9. We eliminate with the following function:

```
alg : Σ[ n : ℕ ] (Fin n ≃ A) → Σ[ n : ℕ ] ∥ Fin n ≃ A ∥
alg (n , f≃A) = n , ∣ f≃A ∣
```

To show that alg is coherently constant, we first notice that the second element of the output pair is propositionally truncated, meaning that it is trivially equal to any other element of the same type. Our task, then, simplifies to demonstrating that the first element of the output pair is coherently constant.

$$\Pi(x : \Sigma(n : \mathbb{N}), \mathbf{Fin}(n) \simeq A), \Pi(y : \Sigma(m : \mathbb{N}), \mathbf{Fin}(m) \simeq A), n \equiv m \tag{30}$$

Notice that $\mathbf{Fin}(n)$ and $\mathbf{Fin}(m)$ are both equivalent to A: we can join these proofs together, giving us the following:

$$\mathbf{Fin}(n) \equiv \mathbf{Fin}(m) \tag{31}$$

All that remains now is to prove that **Fin** is injective. Though the proof is surprisingly complex, it is a well-known puzzle in Martin-Löf type theory. Our proof does not differ significantly from standard approaches, so we will not detail it here.

5.4 Relation to Manifest Bishop Finiteness

Cardinal finiteness tells us that there is an isomorphism between a type and **Fin**; it just doesn't tell us *which* isomorphism. To take a simple example, **Bool** has 2 possible isomorphisms with the set **Fin**(2): one where false maps to 0, and true to 1; and another where false maps to 1 and true to 0.

To convert from Cardinal finiteness to Bishop finiteness, then, requires that we supply enough information to identity a particular isomorphism. A total order is sufficient here: it will give us enough to uniquely order the support list invariant under permutations. This tells us what we already knew in the introduction: manifest Bishop finiteness is cardinal finiteness plus an order.

Theorem 9. Any cardinal finite type with a (decidable) total order is manifestly Bishop finite.

Proof. This proof is quite involved, and will rely on several subsequent lemmas, so we will give only its outline here.

- First, we will convert to manifest enumerability: knowing that the underlying type is discrete (theorem 7) we can go from manifest enumerability to split enumerability (lemma 6), and subsequently to manifest Bishop finiteness (lemma 3).
- To convert to manifest enumerability, we need to provide a support list: this cannot simply be the support list hidden under the truncation, since that would violate the hiding promised by the truncation. Instead, we sort the list (using insertion sort). We must, therefore, prove that insertion sort is invariant under all support lists in cardinal finiteness proofs.
- We do this by first showing that all support lists in cardinal finiteness proofs are permutations of each other, and then that insertion sort is invariant under permutations.

– Given our particular definition of permutations, cover proofs transfers naturally between lists which are permutations of each other.

Now we will build up the toolkit we need to perform the above steps. First, permutations.

Definition 11 (List Permutations)**.** We say that two lists are permutations of each other if there is an isomorphism between membership proofs [4].

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

Lemma 7. Insertion sort is invariant under permutations.

$$xs \leftrightsquigarrow ys \implies \mathrm{sort}(xs) \equiv \mathrm{sort}(ys) \tag{33}$$

Proof. Again, this proof is quite complex, so we only give a sketch here. First, we prove two properties about insertion sort:

1. It returns a sorted list.
2. It a returns a list that is a permutation of its input.

The second of these points allows us to show that $\mathrm{sort}(xs)$ is a permutation of $\mathrm{sort}(ys)$.

$$\mathrm{sort}(xs) \leftrightsquigarrow xs \leftrightsquigarrow ys \leftrightsquigarrow \mathrm{sort}(ys) \tag{34}$$

Then, we show that any lists which are both sorted and permutations of each other are equal. Both of these conditions are true for the output of sort.

6 Kuratowski Finiteness

Finally we arrive at Kuratowski finiteness [10].

Definition 12 (Kuratowski-Finite Set)**.** The Kuratowski finite set is a free join semilattice (or, equivalently, a free commutative idempotent monoid). HITs are required to define this type [3]:

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

The com and dup constructors effectively add commutativity and idempotency to the free monoid (the list), which is made by the first two constructors. The last constructor makes $\mathcal{K}(A)$ a set.

To eliminate from $\mathcal{K}(A)$, we have to provide equations for each of the point constructors which obey the equations of the path constructors. For com and dup, this means ensuring that the fold is commutative and idempotent, whereas trunc means we can only eliminate into sets.

Other representations of $\mathcal{K}$ [7] are more explicit constructions of the free join semilattice (i.e. there is a point constructors for union instead of cons, and then path constructors for the associativity and identity laws), but we have found this representation easier to work with. Nonetheless, the alternative representation is included in our formalisation, and proven equivalent to the representation here.

Definition 13 (Membership of K)**.** First, we need to provide equations for the two point constructors. $x \in [] = \bot$, and $x \in y :: xs = \|(x \equiv y) + (x \in xs)\|$. The com and dup constructors are handled by proving that the truncated form of $+$ is itself commutative and idempotent. The type of propositions is itself a set, satisfying the trunc constructor.

Definition 14 (Kuratowski Finiteness)**.** A type is Kuratowski finite iff there exists a Kuratowski Set which contains all of its elements.

$$\mathcal{K}^f(A) = \Sigma(xs : \mathcal{K}(A)), \Pi(x : A), x \in xs \tag{36}$$

Theorem 10. A proof of Kuratowski finiteness is equivalent to a propositionally truncated proof of enumerability.

$$\mathcal{K}^f(A) \simeq \|\mathcal{E}(A)\| \tag{37}$$

Proof. We prove by way of an isomorphism. In the first direction (from $\mathcal{K}$ to $\mathcal{E}$), because we are eliminating into a proposition, we need only deal with the point constructors. For these, we convert the $\mathcal{K}$ cons to its list counterpart, and similarly for the nil constructor.

The other direction is proven in [7], so we will not describe it here.

6.1 Topos

At this point, we see that a "decidable Kuratowski finite set" is precisely equivalent to a cardinal finite set. From this, we can lift over all of the properties of cardinal finite sets. In particular, we see that decidable Kuratowski finite sets form a *topos*. We already have most of the components we need: closure under $\bot$, $\top$, **Bool**, $+$, $\times$, and $\to$. What remains is the subobject classifier.

Definition 1.

7 Infinite Cardinalities

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.

7.1 Split Countable Types

Our first foray into the world of countable types will be a straightforward analogue to the split enumerable types. We need change only one element: instead of a support *list*, we instead have a support *stream*, which is its infinite.

Definition 15 (Stream)**.** As with lists, it is cleaner, theory-wise, to define streams as a container.

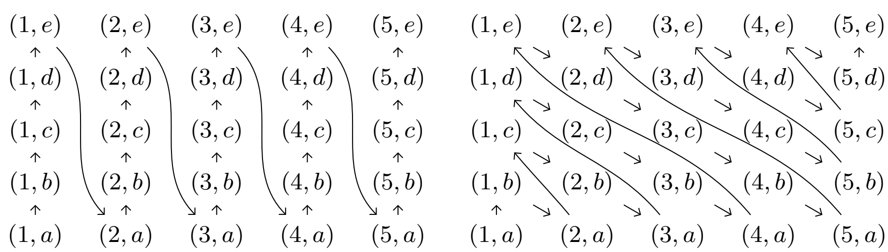
$$\mathbf{Stream}(A) = [\![\top \triangleright \mathrm{const}(\mathbb{N})]\!](A) \simeq \mathbb{N} \to A \tag{38}$$

Conceptually, a stream is like a list without an end.

Definition 16 (Split Countability)**.**

$$\mathcal{E}!(A) = \Sigma(xs : \mathbf{Stream}(A)), \Pi(x : A), x \in xs \tag{39}$$

***Σ* 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. To do this we have to take a slightly different approach to the functions we defined before. Figure 2 illustrates the reason why: previously, we used the "Cartesian" product pairing for each support list. This diverges if the first list is infinite, never exploring anything other than the first element in the second list. Instead, we use here the cantor pairing function, which performs a breadth-first search of the pairings of both lists.



(a) Cartesian (b) Cantor

Fig. 2: Two possible products for the sets $[1 \ldots 5]$ and $[a \ldots e]$

Theorem 11. Split countability is closed under Σ.

Proof. This final proof consists of the support stream, and the proof that the support stream covers the input.

As mentioned, we will have to use a more sophisticated pairing function than the Cartesian product we used before. We instead will mirror the pattern in figure 2b. To greatly simplify the algorithm, we will produce an intermediate stream of lists which consists of the diagonals in the diagram. We then concatenate these streams into the final support stream.

Lemma 8. Split countability is closed under non-dependent product and sum.

Proof. Follows from theorem 11.

Kleene Star While we lose some closures with the inclusion of infinite types, we gain some others. In particular, we have the Kleene star. This means, in effect, that we have closure under lists.

Theorem 12. Split countability is closed under Kleene star.

$$\mathcal{E}!(A) \to \mathcal{E}!(\mathbf{List}(A)) \tag{40}$$

Proof. As with the proof of closure under Σ, our main task here is to figure out a way to arrange the indices such that

7.2 Manifest Countability

As we can quotient out the position information with finite types, so can we with countable types.

fill in rest here

8 Practical Uses

8.1 Omniscience

In this section we are interested in restricted forms of the limited principle of omniscience [11].

Definition 17 (Limited Principle of Omniscience)**.** For any type A and predicate P on A, the limited principle of omniscience is as follows:

$$(\Pi(x : A), \mathbf{Dec}(P(x))) \to \mathbf{Dec}\,(\Sigma(x : A), P(x)) \tag{41}$$

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

$$(\Pi(x : A), \mathbf{Dec}(P(x))) \to \mathbf{Dec}\,(\Pi(x : A), P(x)) \tag{42}$$

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

Theorem 13. Kuratowski-finite types are exhaustible.

Proof.

Proof

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

Lemma 9. Any omniscient type is exhaustible.

Proof. For decidable propositions, we know the following:

$$\Pi(x : A), P(x) \leftrightarrow \neg\Sigma(x : A), \neg P(x) \tag{43}$$

To derive exhaustibility from omniscience, then, we run the predicate in its negated form, and then subsequently negate the result. The resulting decision over $\neg\Sigma(x : A), \neg P(x)$ can be converted into $\Pi(x : A), P(x)$.

We cannot derive, however, that any exhaustible type is omniscient, as we do not have the inverse of equation 43:

$$\Sigma(x : A), P(x) \leftrightarrow \neg\Pi(x : A), \neg P(x) \tag{44}$$

Such an equation would allow us to pick a representative element from any type, which is therefore non-constructive. In a sense, equation 43 requires a form of LEM on the proposition (i.e. requires it to be decidable), whereas equation 44 requires a form of choice. Those finiteness predicates which are ordered do in fact give us this form of choice, so the conversion is valid. As such, all of the ordered finiteness predicates imply omniscience. Again, we will prove it only for the weakest.

Theorem 14. Manifest enumerable types are omniscient.

Proof.

Proof

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

Theorem 15. Kuratowski finite types are omniscient about prop-valued predicates.

Proof.

Proof

8.2 Synthesising Pattern-Matching Proofs

In particular, they can automate large proofs by analysing every possible case. In [6], 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 $\mathcal{E}!$ somehow. As Pauli is a simple finite type, the instance can be defined in a similar way to those in lemma 2. 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 [5]: we ask for the result of a decision procedure as an *instance argument*, which will demand computation during typechecking.

8.3 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 $\rightarrow$). 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)
```

Synthesise functions

Partial search over infinite types

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. Altenkirch, T., Anberrée, T., Li, N.: Definable Quotients in Type Theory (2011)
4. 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
5. 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
6. 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
7. 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
8. 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
9. 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
10. Kuratowski, C.: Sur la notion d'ensemble fini. Fundamenta Mathematicae **1**(1), 129–131 (1920)
11. 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
12. Univalent Foundations Program, T.: Homotopy Type Theory: Univalent Foundations of Mathematics. https://homotopytypetheory.org/book, Institute for Advanced Study (2013)
13. 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