Finite Sets in Homotopy Type Theory

Dan Frumin Herman Geuvers Leon Gondelman
Niels van der Weide

Radboud University Nijmegen, The Netherlands

January 9, 2018, CPP

What Is Finiteness, Constructively?

- A set A is (Bishop)-finite if there are *exactly* $n \in \mathbb{N}$ elements in it.
- A set A is (Kuratowski)-finite if there are *at most* $n \in \mathbb{N}$ elements in it.
- Classically, these are equivalent, but constructively they are different.
- Goal: explore Kuratowski-finite sets in the setting of homotopy type theory.
- Material in this talk and more has been formalized in Coq using the Coq-HoTT library.

First Attempt: Sets as Lists

- First attempt: represent a set as a list of elements.
- This datatype has propositional equality different from sets.

$$l_1 \sim l_2 \iff \forall x, \mathsf{member}(x, l_1) \leftrightarrow \mathsf{member}(x, l_2)$$

- Operations on sets become operations on lists.
- Not all functions on lists are functions on sets (*e.g.*, length); functions have to respect $\sim$.
- This representation does not provide a useful proof principle.

Obtaining The Right Notion of Equality

- Setoids: no extra machinery required, but cumbersome, gives bigger proof terms.
- Quotients: some extra machinery required, but some extra work for lifting operations.
- Higher inductive types: give the right constructions, equations and proof principles immediately.

Our Approach

HoTT with Univalence and Higher Inductive Types.

- In the lingo of HoTT: types are spaces, terms are *points*, and proofs of equalities $x = y$ are *paths* between x and y.
- Univalence allows us to identify equivalent types.
- HITs: both point and path constructors allowed.
- The exact syntax and semantics of HITs is still up to some debate. We use the syntax from (Basold, Geuvers, Van der Weide (2017), Dybjer, Moeneclaey (2017))

Finite Sets as a Higher Inductive Type

$$
\begin{array}{l}
\texttt{Inductive}\ \mathcal{K}\ (\texttt{A} : \texttt{Type}) := \\
\mid\ \varnothing : \mathcal{K}\ \texttt{A} \\
\mid\ \{\cdot\} : \texttt{A} \to \mathcal{K}\ \texttt{A} \\
\mid\ \cup : \mathcal{K}\ \texttt{A} \to \mathcal{K}\ \texttt{A} \to \mathcal{K}\ \texttt{A}
\end{array}
\left.\right\}\ \text{Point constructors}
$$

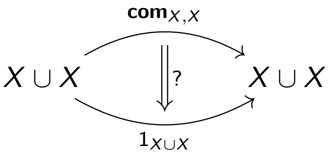
$$
\left.
\begin{array}{l}
\mid\ \textbf{nl} : \prod(x : \mathcal{K}(A)),\ \varnothing \cup x = x \\
\mid\ \textbf{nr} : \prod(x : \mathcal{K}(A)),\ x \cup \varnothing = x \\
\mid\ \textbf{idem} : \prod(a : A),\ \{a\} \cup \{a\} = \{a\} \\
\mid\ \textbf{assoc} : \prod(x, y, z : \mathcal{K}(A)),\ x \cup (y \cup z) = (x \cup y) \cup z \\
\mid\ \textbf{com} : \prod(x, y : \mathcal{K}(A)),\ x \cup y = y \cup x.
\end{array}
\right\}\ \text{Path constructors}
$$

Finite Sets as a Higher Inductive Type

$\texttt{Inductive}\ \mathcal{K}\ (\texttt{A} : \texttt{Type}) :=$
$|\ \varnothing : \mathcal{K}\ \texttt{A}$
$|\ \{\cdot\} : \texttt{A} \to \mathcal{K}\ \texttt{A}$
$|\ \cup : \mathcal{K}\ \texttt{A} \to \mathcal{K}\ \texttt{A} \to \mathcal{K}\ \texttt{A}$
} Point constructors

$|\ \mathbf{nl} : \prod(x : \mathcal{K}(A)),\ \varnothing \cup x = x$
$|\ \mathbf{nr} : \prod(x : \mathcal{K}(A)),\ x \cup \varnothing = x$
$|\ \mathbf{idem} : \prod(a : A),\ \{a\} \cup \{a\} = \{a\}$
$|\ \mathbf{assoc} : \prod(x, y, z : \mathcal{K}(A)),\ x \cup (y \cup z) = (x \cup y) \cup z$
$|\ \mathbf{com} : \prod(x, y : \mathcal{K}(A)),\ x \cup y = y \cup x$
$|\ \mathbf{trunc} : \prod(x, y : \mathcal{K}(A)),\ \prod(p, q : x = y),\ p = q.$
} Path constructors

Truncation **trunc** identifies higher paths in the type, *e.g.*, :



Recursion Principle for Finite Sets

$$Y : \mathrm{TYPE}$$
$$\varnothing_Y : Y$$
$$L_Y : A \to Y$$
$$\cup_Y : Y \to Y \to Y$$

$$\overline{\mathcal{K}(A)\, \mathbf{rec}(\varnothing_Y, L_y, \cup_Y \qquad\qquad) : \mathcal{K}(A) \to Y}$$

Recursion Principle for Finite Sets

$$\frac{\begin{array}{c} Y : \textsc{Type} \\ \varnothing_Y : Y \\ L_Y : A \to Y \\ \cup_Y : Y \to Y \to Y \\ \mathbf{nl}_Y : \prod(a : Y), \varnothing_Y \cup_Y a = a \\ \mathbf{nr}_Y : \prod(a : Y), a \cup_Y \varnothing_Y = a \\ \mathbf{idem}_Y : \prod(a : A), \{a\}_Y \cup_Y \{a\}_Y = \{a\}_Y \\ \mathbf{assoc}_Y : \prod(a, b, c : Y), a \cup_Y (b \cup_Y c) = (a \cup_Y b) \cup_Y c \\ \mathbf{com}_Y : \prod(a, b : Y), a \cup_Y b = b \cup_Y a \\ \mathbf{trunc}_Y : \prod(x, y : Y), \prod(p, q : x = y), p = q \end{array}}{\mathcal{K}(A)\, \mathbf{rec}(\varnothing_Y, L_y, \cup_Y, \mathbf{nl}_Y, \mathbf{nr}_Y, \mathbf{idem}_Y, \dots) : \mathcal{K}(A) \to Y}$$

The Need for Truncation

Suppose we are proving an equation of finite sets, *e.g.*,

$$\prod_{X:\mathcal{K}(A)} X \cup X = X$$

- For $\varnothing$ we provide $\mathbf{nr} : \varnothing \cup \varnothing = \varnothing$.
- For $X_1 \cup X_2$ we provide
  $(X_1 \cup X_1 = X_1) \to (X_2 \cup X_2 = X_2) \to$
  $(X_1 \cup X_2) \cup (X_1 \cup X_2) = (X_1 \cup X_2)$

$$p_{X_1,X_2}(H_1, H_2) = \mathbf{assoc} \cdot (\mathrm{ap} \cdots \mathbf{com}_{X_1,X_2}) \cdot (\mathrm{ap} \cdots \mathbf{assoc}^{-1}) \cdot \ldots$$

- Then for **nl** we need to provide a higher equality

$$\mathbf{nl}_*(p_{\varnothing,\varnothing}(\mathbf{nr}_\varnothing, \mathbf{nr}_\varnothing)) = \mathbf{nr}_\varnothing \,.$$

This is easy with the truncation **trunc**.

Propositional Truncation

Towards Propositional Membership

Definition (Propositions)

$\textsc{hProp}$ is the universe of proof-irrelevant types (propositions), *i.e.*, types A such that $\prod(x, y : A),\ x = y$.

Definition (Propositional Truncation)

$|| - || : \textsc{Type} \to \textsc{hProp}$

```
Inductive ||A|| :=
| tr : A → ||A||
| trunc : ∏(x, y : ||A||), x = y.
```

$||\mathtt{A}||$ is `A` with all elements identified.

Defining Propositional Membership

Application of the Recursion Principle

We define $\in: A \to \mathcal{K}(A) \to \text{hProp}$ by $\mathcal{K}(A)$-recursion.

$$\begin{aligned} a \in \varnothing &:= \bot, \\ a \in \{b\} &:= ||a = b||, \\ a \in (x_1 \cup x_2) &:= ||a \in x_1 + a \in x_2|| \end{aligned}$$

What about the path constructors?

- $(\text{hProp}, \bot, \vee)$ is a join semi-lattice;
- All paths between propositions are equal.

Requires *univalence* to show, *e.g.*,

$$||a \in x_1 + a \in x_2|| = ||a \in x_2 + a \in x_1||$$

Extensionality for Finite Sets

Theorem (Extensionality)

For all $x, y : \mathcal{K}(A)$ *the types* $(x = y)$ *and* $(\prod(a : A), (a \in x = a \in y))$ *are equivalent.*

Extensionality for Finite Sets

Theorem (Extensionality)

For all $x, y : \mathcal{K}(A)$ *the types* $(x = y)$ *and* $(\prod(a : A), (a \in x = a \in y))$ *are equivalent.*

Proof sketch.

Through a chain of equivalences

$$(x = y) \simeq ((y \cup x = x) \times (x \cup y = y)) \simeq (\prod_{a:A} a \in x = a \in y)$$

Equational reasoning

Nested induction on y and x

□

From Finite Sets to Finiteness

- Each $X : \mathcal{K}(A)$ gives a subobject

$$(\lambda x. x \in X) : A \to \text{HPROP}$$

- We think of $\mathcal{K}(A) \hookrightarrow \text{HPROP}^A$ as the collection of *Kuratowski-finite subobjects* of A.
- A type A is finite if the maximal subobject $\top_A$ is finite.

Definition (Kuratowski-finite types)

$$\mathsf{isKf}(A) := \sum (X : \mathcal{K}(A)), \prod (a : A), a \in X$$

NB: For every type A, $\mathsf{isKf}(A)$ is a mere proposition – has at most one inhabitant by extensionality.

Bishop-finiteness

Bishop-finiteness was previously explored in homotopy type theory

Definition (Bishop-finite types)

$$\mathsf{isBf}(A) := \sum (n : \mathbb{N}), ||A \simeq [n]||$$

- All finite cardinals $[n] = \{0, \ldots, n-1\}$ have decidable equality.
- It follows that every Bishop-finite type has decidable equality as well.
- This contrasts with Kuratowski-finite types, which need not have decidable equality.

Bishop-finiteness vs Kuratowski-finiteness

$$\mathsf{isKf}(A) := \sum (X : \mathcal{K}(A)), \prod (a : A), \ a \in X$$
$$\mathsf{isBf}(A) := \sum (n : \mathbb{N}), ||A \simeq [n]||$$

- Bishop-finite types are Kuratowski-finite.
- The other direction does not hold in general. (Counterexample: assuming univalence, S^1 is Kuratowski-finite, but not Bishop-finite because it doesn't have decidable equality).
- To better compare two notions we need to generalize Bishop-finite types to *finite subobjects*.

A subobject $P : A \to \mathrm{HPROP}$ is Bishop-finite if the subset $\sum_{x:A} P(x)$ is Bishop finite.

Comparison of Finite Subobjects

A type A has *decidable mere equality* if

$$\prod_{x,y:A} ||x = y|| + ||x \neq y||$$

	Bishop-finite subobjects $\sum_{X:A\to\text{hProp}} \mathsf{isBf}(X)$	**Kuratowski-finite subobjects** $\mathcal{K}(A)$
$\cup$	Iff A has decidable equality (given that A is an hSet)	Always definable
$\{-\}$	Iff A is an hSet	Always contains singletons
$\varnothing$	Always present	Always present
$\cap$	Iff A has decidable equality (given that A is an hSet)	Iff A has decidable mere equality
$\in_d$	Iff A has decidable equality (given that A is an hSet)	Iff A has decidable mere equality

Bishop-finiteness vs Kuratowski-finiteness (2)

Theorem

If A has decidable equality then $\mathsf{isKf}(A) \to \mathsf{isBf}(A)$.

Corollary

If A has decidable equality then $\mathsf{isKf}(A) \simeq \mathsf{isBf}(A)$.

An Interface for Finite Sets

Definition

A type T is an interpretation of finite sets over A if there are

- a term $\varnothing_T : T$;
- an operation $\cup_T : T \to T \to T$;
- for each $a : A$ a term $\{a\}_T : T$;
- a family of predicates $a \in_T - : T \to \text{hProp}$.

Definition

A homomorphism between interpretations T and R is a function $f : T \to R$ that commutes with all the operations.

$$f\,\varnothing_T = \varnothing_R \qquad f(x \cup_T y) = f\,x \cup_R f\,y$$

$$f\,\{a\}_T = \{a\}_R \qquad a \in_T x = a \in_R f\,x$$

Implementations of Finite Sets

Definition

An implementation of finite sets consists of

- a type family $T : \mathrm{Type} \to \mathrm{Type}$ such that each $T(A)$ is an interpretation of finite sets;
- homomorphisms $[\![\cdot]\!]_A : T(A) \to \mathcal{K}(A)$.

The maps $[\![\cdot]\!]_A$ are always surjective. Furthermore,

- functions on $\mathcal{K}(A)$ are carried over to any implementation of finite sets;
- properties of these functions carry over.

Relating Lists and $\mathcal{K}$

$$\textsc{List}(A) \xrightarrow{\mathsf{fold}(\varnothing, \lambda x \lambda y. \{x\} \cup y)} \mathcal{K}(A)$$

Lists implement finite sets with

- **nil** : $\textsc{List}(A)$,
- append : $\textsc{List}(A) \to \textsc{List}(A) \to \textsc{List}(A)$,
- member : $A \to \textsc{List}(A) \to \mathrm{hProp}$,
- and the homomorphism
  - $[\![\mathbf{nil}]\!] = \varnothing$,
  - $[\![h :: t]\!] = \{h\} \cup [\![t]\!]$

Lifting operations

We lift maps $\mathcal{K}(A) \to B$ to $\textsc{List}(A) \to B$ by composing with $[\![\cdot]\!]_A$.

- Define $\forall : (A \to \textsc{hProp}) \to \mathcal{K}(A) \to \textsc{hProp}$ such that

$$\prod_{a:A} \prod_{X:\mathcal{K}(A)} (a \in X) \times \forall(P, X) \to P(a).$$

Lifting operations

We lift maps $\mathcal{K}(A) \to B$ to $\textsc{List}(A) \to B$ by composing with $[\![\cdot]\!]_A$.

- Define $\forall : (A \to \text{hProp}) \to \mathcal{K}(A) \to \text{hProp}$ such that

$$\prod_{a:A} \prod_{X:\mathcal{K}(A)} (a \in X) \times \forall(P, X) \to P(a).$$

- It lifts to $\forall_L : (A \to \text{hProp}) \to \textsc{List}(A) \to \text{hProp}$ such that

$$\prod_{a:A} \prod_{X:\textsc{List}(A)} (\text{member}(a, l) \times \forall_L(P, l)) \to P(a).$$

Because

$$(\text{member}(a, l) \times \forall_L(P, l)) = ((a \in [\![l]\!]) \times \forall(P, [\![l]\!]))$$
$$\implies P(a)$$

Lifting operations

We lift maps $\mathcal{K}(A) \to B$ to $\textsc{List}(A) \to B$ by composing with $[\![\cdot]\!]_A$.

- Define $\forall : (A \to \textsc{hProp}) \to \mathcal{K}(A) \to \textsc{hProp}$ such that

$$\prod_{a:A} \prod_{X:\mathcal{K}(A)} (a \in X) \times \forall(P, X) \to P(a).$$

- It lifts to $\forall_L : (A \to \textsc{hProp}) \to \textsc{List}(A) \to \textsc{hProp}$ such that

$$\prod_{a:A} \prod_{X:\textsc{List}(A)} (\mathsf{member}(a, l) \times \forall_L(P, l)) \to P(a).$$

- Similarly if A has decidable mere equality, we can define
  - $\mathsf{size} : \mathcal{K}(A) \to \mathbb{N}$ lifting to $\mathsf{size}_L : \textsc{List}(A) \to \mathbb{N}$
  - $\in_d : A \to \mathcal{K}(A) \to \textsc{Bool}$ lifting to $\in_d : A \to \textsc{List}(A) \to \textsc{Bool}$.

Summary

- Formalized development of finite sets using HITs.
- Comparative study of Bishop-finiteness and Kuratowski-finiteness in HoTT.
- Interface for finite sets suitable for data refinement.

`https://cs.ru.nl/~nweide/fsets/finitesets.html`

Thank you for listening.

Induction Principle for Kuratowski Sets

$$
\frac{\begin{array}{c}
Y : \mathcal{K}(A) \to \textsc{Type} \\
\varnothing_Y : Y[\varnothing] \\
L_Y : \prod(a : A), Y[\{a\}] \\
\cup_Y : \prod(x, y : \mathcal{K}(A)), Y[x] \times Y[y] \to Y[\cup(x, y)] \\
n_1 : \prod(x : \mathcal{K}(A)) \prod(a : Y[x]), \cup_Y(\varnothing_Y, a) =^Y_{\mathbf{nl}} a \\
n_2 : \prod(x : \mathcal{K}(A)) \prod(a : Y[x]), \cup_Y(a, \varnothing_Y) =^Y_{\mathbf{nr}} a \\
i_Y : \prod(a : A), \cup_Y(L_Y\, x, L_Y\, x) =^Y_{\mathbf{idem}} L_Y\, x \\
a_Y : \prod(x, y, z : \mathcal{K}(A)) \prod(a : Y[x]) \prod(b : Y[y]) \prod(c : Y[z]), \\
\cup_Y(a, (\cup_Y(b, c))) =^Y_{\mathbf{assoc}} \cup_Y(\cup_Y(a, b), c) \\
c_Y : \prod(x, y : \mathcal{K}(A)) \prod(a : Y[x]) \prod(b : Y[y]), \\
\cup_Y(a, b) =^Y_{\mathbf{com}} \cup_Y(b, a) \\
t_Y : \prod(x : \mathcal{K}(A)), Y[x] \in \textsc{hSet}
\end{array}}{\mathcal{K}(A)\, \mathbf{rec}(\varnothing_Y, L_y, \cup_Y, a_Y, n_{Y,1}, n_{Y,2}, c_Y, i_Y) : \prod(x : \mathcal{K}(A)), Y}
$$