Formalizations of finite sets

Johannes Tantow

May 30, 2023

1 Formalizations

In [1] multiple ways to define finite sets via homotopy type theory in Coq are presented. The first approach are Kuratowski finite sets, as a higher order inductive type. This combines point constructors, that are elements of this type i.e. finite sets, and path constructors, that provide proofs that certain elements of this type are the same, where are proof is a path between these two points. Lean is not based on HoTT, but I believe it is still possible to simulate most of this. There used to be a library for Lean 2, but this seems to be no longer developed. Instead of a higher order inductive type we use a normal inductive type for the point constructors and create a type that consists of the empty set, singleton sets and the union of two sets. For the path constructors we can use axioms in lean to state that the union operation is commutative, associative or idempotent. It is unclear for me so far how to express the truncation condition, which states that all proofs of equality between two elements are the same, but its use outside of HoTT is also unclear. An alternative to this is to use the lean internal type of

If the type of the elements has decidable equality, one can define a member $\in$ function and intersection operations, so that Kuratowski-finite sets form a semi-lattice. In case of decidable equality it is equivalent to Bishop-finiteness. For Bishop-finiteness we define for every integer a canonical set of this size. A set is then bishop-finite if there exists a bijection from the set to one of the canonical sets. This makes only sense if we can decide equality to see if a map is really a bijection. This is not a trivial statement as equality on the real numbers with the sine function is undecidable.[2]

Kuratowski-finite sets serve in this paper as an interface for finite sets. We can have other implementations of finite sets (i.e. they have some $\emptyset$ object, singletons, a union and a member function $\in$), that fulfill some reasonable conditions like preservation of membership, so that they are a homomorphism to the Kuratowski-finite sets. This is not in general an isomorphism as the lists

```
[1,2]
```

and

```
[2,1]
```

are different, but in an easy conversion to Kuratowski sets the same. This can be solved by creating the quotient of this implementation, where 2 elements of the implementation

Mathlib already implements a type of finite sets.[3] Finite sets are lists together with a proof that there are no duplicates in that list (in contrast to finite multisets which are just lists). In contrast to Kuratowski sets it is very easy to define a length function, as the length for lists is easier to define than for Kuratowski sets, where an element can occur multiple times. In contrast some operations are a bit more cumbersome as we need to create new proofs that there are new duplicates.

There are further datastructures possible, as sets are implemented in many programming languages, which gives further inspiration. Sets are e.g. hashmaps in Python or red-black trees in C++, whereas Java offers both. These approaches are often faster, but have a more complicated implementations. The expressivity in contrast to Kuratowski sets should be the same, but it is open how this affects proofs, where the speed probably doesn't matter so much in the case of formal verification. In [4] is the formal implementation of red-black trees in Coq described together with correctness results like after insert all previous elements stay inside the tree. Additionally, it presents ways to define set-operations on red-black trees like union or intersection. This is again done to keep speed in mind, so the implementation of union that is selected depends on the relative size of both sets. In contrast to Kuratowski sets we need an order on our type, which is not always given. Lean itself already has trees inside the library, but no real results or functions to modify them like a data structure. In [5] AVL trees are defined and correctness results for them are proven, but with no application to sets, which would need further extension of this work. Additionally, some proofs are left incomplete in the implementation (i.e. they use sorry)

Are there further definitions possible that don't just depend on different datastructures. A look into constructive mathematics offers further defintions for finiteness. In [6] 4 ways for finiteness are presented and closure properties like under $\times$ or $\cup$ are proven.

1. enumerability: A set A is finite if there is a list that contains all its elements. This is equivalent to the other so implementations that were previously presented

2. bounded size: A set A is finite if there is a natural number N, s.t. all lists of size N with elements from A contains duplicates. This is different in constructive mathematics, since it is always preserved under subsets as we can take the same N again. In contrast, given a list of turing machines there is no (constructive) way to get a list of those turing machines that halt on the empty word, so that enumerated sets are not closed under substructures.

3. notherian: This condition is inspired from algebra and states informally there that every chain of subsets converges after a finite amount of steps.

Constructively, we consider a list. If a list has duplicates, then we stop. Else we spawn subprocesses where we add some element from A to the list in each subprocess and continue this recursively until all subprocesses return duplicates. One starts this process with the empty list. Variants of this are explored in [7], where one modifies which elements to select or when to stop. This is similar to bounded size, but we don't explicitly show the number.

4. streamless: A stream is a map $\mathbb{N} \to A$. A set is streamless, if all streams contain duplicates. This is special, as this is the only type of finite criteria that is not closed under $\times$.

These are only criterias for finiteness, so we have to combine them with a formalization of a set. This has to be different to a list or else it the criteria would be useless, so one probably models the set as a function $f : A \to \mathit{bool}$, where A is the type of the elements the set shall contain.

[8] provide additional definitions of finite sets, but modelled in agda. They don't define sets, but instead define critierias that take a set instead of the inductive approach in [1], but also use lists without duplicates or bijections to canonical objects or a subsets of a finite type.

2 Applications

This sections collects results that use finite sets that can be formalized.

A well-known principle from finite set theory is the inclusion-exclusion principle, i.e. $|X \cup Y| + |X \cap Y| = |X| + |Y|$ that can be proven whenever we have decidable equality to define a member function.

Turing machines and automatas need finite sets in order to have meaningful results about languages (or else we encode everything in the state set and can recognise arbitrary languages). Those are encoded with lists in [9], with a custom defined finite type in [10] or theoretically viewed in [6].

Trakhtenbrots theorem states that finite decidability is undecidable. There we need to define a finite model and need a finite set for this. In [11] this is done again with a list. The proof is however quite large with many parts that don't care about finite sets so formalizing this is a large task.

More results on finite sets are provided in proofs from the book[12]. There are proofs of Sperners theorem (about antichains), the Erdős-Ko theorem about intersecting families and Halls marriage theorem. Those need further formalizations to do it, but maybe smaller than Trakhtenbrot, but they are only partially related to finite sets.

References

[1] Dan Frumin, Herman Geuvers, Léon Gondelman, and Niels van der Weide. Finite sets in homotopy type theory. In June Andronick and Amy P. Felty,

editors, *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, pages 201–214. ACM, 2018.

[2] Daniel Richardson. Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic*, 33(4):514–520, 1968.

[3] Lean community. Finite sets. `https://leanprover-community.github.io/mathlib_docs/data/finset/basic.html`.

[4] Andrew W. Appel. Efficient verified red-black trees. 2011.

[5] Sofia Konovalova. Verifying avl trees in lean. `https://lean-forward.github.io/pubs/konovalova_bsc_thesis.pdf`, 2021.

[6] Arnaud Spiwack and Thierry Coquand. Constructively finite? page 978, 2010.

[7] Denis Firsov, Tarmo Uustalu, and Niccolò Veltri. Variations on noetherianness. In Robert Atkey and Neelakantan R. Krishnaswami, editors, *Proceedings 6th Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2016, Eindhoven, Netherlands, 8th April 2016*, volume 207 of *EPTCS*, pages 76–88, 2016.

[8] Denis Firsov and Tarmo Uustalu. Dependently typed programming with finite sets. In Patrick Bahr and Sebastian Erdweg, editors, *Proceedings of the 11th ACM SIGPLAN Workshop on Generic Programming, WGP@ICFP 2015, Vancouver, BC, Canada, August 30, 2015*, pages 33–44. ACM, 2015.

[9] Yannick Forster, Fabian Kunze, and Maxi Wuttke. Verified programming of turing machines in coq. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 114–128. ACM, 2020.

[10] Jan Menz. A coq library for finite types. `https://www.ps.uni-saarland.de/~menz/bachelor.php`, 2016.

[11] Dominik Kirst and Dominique Larchey-Wendling. Trakhtenbrot's theorem in coq: Finite model theory through the constructive lens. *Log. Methods Comput. Sci.*, 18(2), 2022.

[12] Martin Aigner and Günter M. Ziegler. *Three famous theorems on finite sets*, pages 213–217. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.