Developing sequence and tree fintypes in MathComp

Pierre-Léo Bégay[1,2], Pierre Crégut[1], and Jean-François Monin[2]

[1] Orange
[2] Verimag

1 Introduction

One of MathComp's main strengths is its ability to interpret a set of elements of a type fulfilling a given predicate as a regular type. Those types are organized as a hierarchy of structures verifying a set of properties: having decidable equality, having a choice function, being countable, and finiteness.

The MathComp library already provides ways to combine such structures using the classical sum and product operators, but the standard library does not provide any support for recursive types. Recursive types indeed usually lose the properties of their carrier types unless some finiteness properties are added.

We introduce two new types for explicitly bounded sequences and trees. We then study another, implicit way to bound the size of such types, by putting a constraint of no repetition along paths. This is typically used in program verification theory to limit the size of an abstract control stack.

2 Bounding sequences

2.1 Syntactically bounded sequences

MathComp already contains a type for sequences of exactly a given length, called `tuple`, which inherits the finiteness property of any finType used for the elements. We have developed `Wlist`, the type of lists (`seq`, in MathComp's nomenclature) bounded by a given `nat`. Unlike `tuple`, `Wlist` is not defined using a signature type, but an inductive:

```
Inductive Wlist (X: Type): nat -> Type :=
  Bnil  : forall n, (Wlist X n)
| Bcons : forall n, X -> (Wlist X n) -> (Wlist X n.+1).
```

The finiteness of `Wlist A n`, where `A` is a finType and `n` a `nat`, is shown via induction. In the inductive case, a `Wlist A n+1` is simply transformed as an element of type `unit + (Wlist A n * A)` using the following function:

```
Equations ff (n : nat) (x : Wlist A n.+1) : (unit + A * Wlist A n) :=
  ff (Bnil _)    := inl tt ;
  ff (Bcons a l) := inr (a,l).
```

This function uses [1], as the usual match failed to deal with the `n` argument. The sum or product of two finTypes is a finType itself, and the finiteness of `Wlist A n` is the induction hypothesis. The proof is shorter and simpler than the one for the finiteness of `tuple`. This probably stems from the fact that it is less precise, as we do not explicitely state or prove the cardinal of `Wlist`.

Using this type in practice can be cumbersome, as adding an element to a `Wlist` bounded by nat `m` would build a `Wlist m+1`, even though there may actually be much less than `m+1` elements in the list. We wrote functions that map elements of `Wlist` to usual sequences, and the other way around. In the second case, elements from the list can be lost if it was too long. However, we also wrote cancellation lemmas relying on properties about a given sequence's length. This controlled back and forth allowed us to use `Wlist` in practice, at the reasonable cost of tracking the size of the studied lists explicitly.

```
Lemma wlist_seqK (l : Wlist X m) : (seq_to_wlist m (wlist_to_seq l)) = l.
Lemma seq_wlistK (l : seq X) (H : size l <= m) : (wlist_to_seq (seq_to_wlist m l)) = l.
```

2.2 Sequences with unicity

We introduce a signature type, called `uniq_seq`, of sequences with unicity. It is shown that a `uniq_seq` over a finType `A` can be injected into a tuple of length bounded by `#|A|`, ie. the cardinal of `A`, which grants us finiteness. We provide two functions that add an element `x` at the head of an `uniq_seq l`. The first requires a proof of `x \notin l` to build a proof of `uniq (x :: l)`. The other function uses `sumbool` to check whether `x \notin l`, and extracts a proof if it is the case. If not, it simply returns `l`.

3 Bounding trees

3.1 Syntactically bounded trees

Using `Wlist`, we define the type of trees with a bounded height and number of successors, called `Htree`. This type is also defined using Inductive, its finiteness is shown similarly to that of `Wlist`, and [1] was also used, this time to map `Htree` to a general, unbounded tree type.

3.2 Semantically bounded trees using unicity

Having two different syntactic bounds makes the use of `Htree` quite tedious. Bounding the number of successors of nodes was not a problem in our use case, so we only had to deal with the height. Our trick is again to use unicity, by showing that trees over finType `A` with at most `n` successors and unicity across paths are a subtype of trees with a most `n` successors and a height of `#|A|`, making it a finType.

Trying to get this result directly proved tricky, it was easier to show a more general lemma stating that a tree with unicity and elements forming a subset of `E` had a height bounded by `|E|`. Replacing `E` by `A`, the subset condition is trivially true and we have our desired result. As for `uniq_seq`, we developed an insertion function that takes as an argument a proof of the absence of the top element in the given subtrees.

4 Conclusion and perspectives

MathComp provides a lot of useful meta theorems on types verifying properties ranging from the existence of a decidable equality to finiteness or simply being countable, but proving those properties can be tedious.

The finiteness proofs mentioned here rely on the fact that if a type `T` can be injected in a finite type `F` then `T` is also finite, on the preservation of finiteness by sums and products, and on the existence of a natural correspondence from non recursive datatypes with a given sum of products of more elementary types. Working on these finTypes, which we needed for another project, led us to work on a tactic that automates the simplest cases of derivation of "MathComp properties" from any user-defined type.

This work is still in very early stages, and more involved than expected. In particular, abstracting finTypes is made difficult by their encapsulation. We will discuss this point and present our current leads.

References

[1] Sozeau, M.: Equations: A dependent pattern-matching compiler. In: International Conference on Interactive Theorem Proving. pp. 419–434. Springer (2010)