Developing sequence and tree fintypes in MathComp

Pierre-Léo Begay [1,2] **Pierre Crégut** [1] **Jean-François Monin** [2]

July 2020

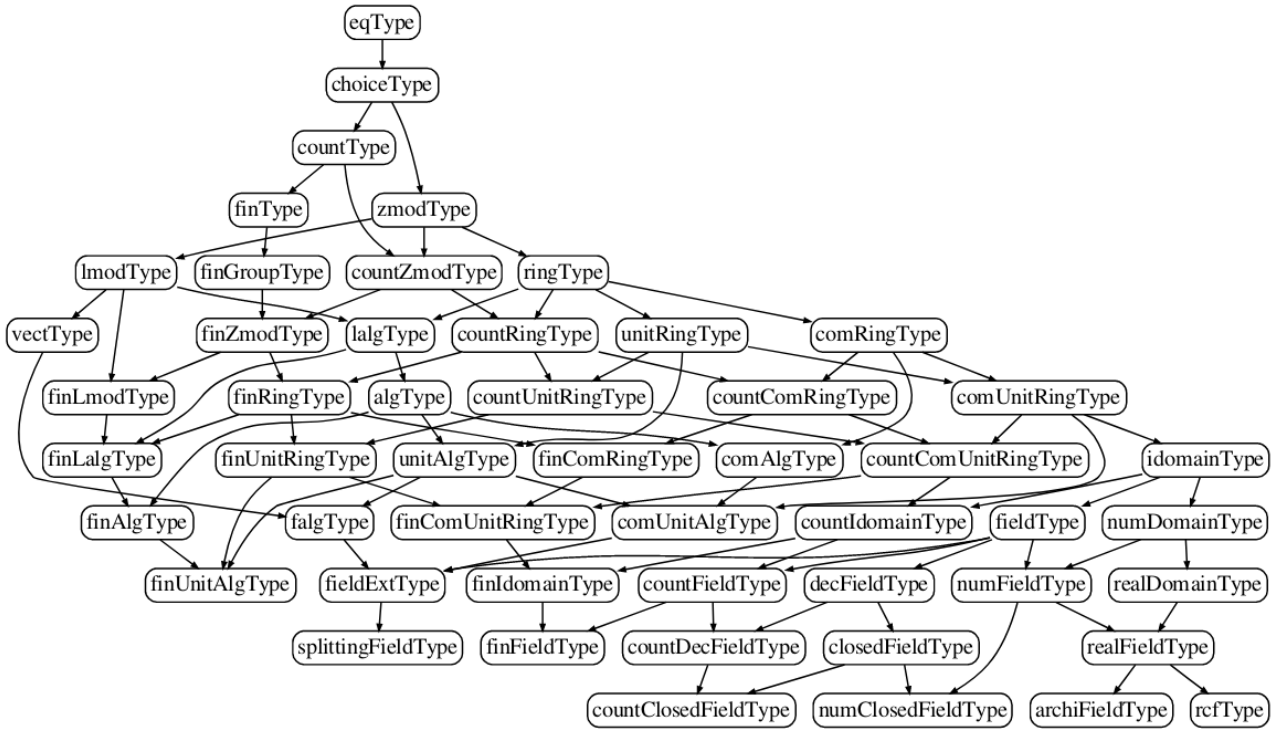
[1]Orange Labs [2]Verimag (Université Grenoble Alpes)

Plan

Plan

A type hierarchy for algebra

Algebraic structures inheriting properties and associated functions [Garillot et al., 2009, Sakaguchi, 2020]



The top of the tree can be used for general-purpose data structures

A type hierarchy for many cases

- `eqType`
  - A type *packaged* with a decidable equality
- `choiceType`
  - Choice function
- `countType`
  - Seen as an ordered and indexed list
- `finType`
  - Finite number of elements $\Rightarrow$ many new possibilities

A `finType` is a `countType` and so on.

Using finTypes

- Given a finType `A`, one can define the (fin)type `{set A}`.
- Usual set operations (union, intersection, difference, complement), as well as comprehension

```
[set f x | x in X & P x]
```

- Powerful tool that avoids dealing with circumbulated definitions and allows a more paper-like reasoning
- MathComp contains a list finType (`m.-tuple A`, lists with `m` elements of finType `A`) and no tree one

Plan

A finType heavy project

- We extend a Coq/MathComp formalization of Datalog [Benzaken et al., 2017b, Benzaken et al., 2017a] relying heavily on finTypes
- Datalog is `deeply finite`

$$\begin{array}{l} \mathit{linked}(X, Y) \text{ :- } \mathit{edge}(X, Y). \\ \mathit{linked}(X, Y) \text{ :- } \mathit{linked}(X, Z), \mathit{edge}(Z, Y). \end{array}$$

- Need new finTypes to
  - preserve this *finiteness spirit* (introduction of a trace semantics)
  - develop an analysis with built-in termination

Plan

Syntactically bounded lists

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

- Finiteness shown by induction on `n`
- a `Blist A n+1` is transformed as a `unit + (Blist A n * A)`

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

- The usual match failed to deal with `n`, hence the use of Equations [Sozeau, 2010]

Semantically bounded lists

```
Structure uniq_seq {A : eqType} :=
  {useq :> seq A ; buniq : uniq useq}.
```

- Given a finType `A`, the finiteness of `uniq_seq A` is shown by injecting its elements into `m.-tuple A`, where `m` is bounded by the cardinal of `A`
- Alternatively, injection into `Blist A #|A|`
- Two `cons` functions : use a `\notin` proof or try to compute one, and build a new proof of uniqueness

Syntactically bounded trees

```
Inductive Btree: nat -> Type :=
  BLeaf : forall n, B -> (Btree n)
| BNode : forall n, A -> (Blist (Btree n) w)
                      -> (Btree n+1).
```

- Height bounded by `n`, width by `w`
- Different types for nodes and leaves, required by our use case
- Finiteness again shown by induction on `n` and a translation from `Btree A B n+1 w` to `B + (A * Blist (Btree A B n w) w)`
- Equations again required, this time when translating to a generic, unbounded tree type

Syntactically bounded trees

```
Equations b_to_tree {A B : Type} {w n : nat}
          (t : @Btree w A B n) : @tree A B by wf n :=
  b_to_tree (@BLeaf _ x) := @Leaf A B x ;
  b_to_tree (@BNode h y l) :=
    Node y
      (map (fun tb : @Btree w A B h => @b_to_tree A B w h tb)
           (blist_to_seq l)).
```

More complex than for `Blist`, but Equations handles it

(Partially) Semantically bounded trees

```
Definition uw_pred {A B : eqType} {w : nat} (t : @tree A B) :=
  ((tree_uniq t) && (tree_width t <= w)).

Structure Utree {A B : eqType} (w : nat) :=
  Wht {wht :> @tree A B ; Hwht : @uw_pred A B w wht}.
```

- Height bounded with unicity, width still syntactically (project-driven choice)
- Proof of finiteness :
  - trees with unicity and elements forming a subset of `E` have a height bounded by `|E|`.
  - if `E` is a finType, this bounds the height by `E`'s cardinal
  - any `Utree A B w` can then be injected into `Btree A B w #|A|`

Plan

Using bounded lists

- Back and forth between `seq` and `Blist`

```
Lemma blist_seqK (l : Blist X m) :
  (seq_to_blist m (blist_to_seq l)) = l.

Lemma seq_blistK (l : seq X) (H : size l <= m) :
  (blist_to_seq (seq_to_blist m l)) = l.
```

- Usable, but awkward
- "my first intuition would have been to go in a totally different direction and simply code bounded sequences as `{s : seq; Hs : size s <= n}`" $\Rightarrow$ probably a welcome (and now almost free) abstraction

Syntactic bounds as a backbone

- A simple plan :
  - Develop a type with built-in finiteness
  - Inject a more abstract type in it
- *Feels* very naive compared to MathComp (cf. proof of `enumP` in `tuple.v`)
- Longer but easier and more readable proofs
- Strategy used in reaction to MathComp's opaqueness

Future works

- Integration into MathComp ?
- A tactic to automatically derive eq/choice/count/finType properties of an Inductive
  - Very early stage, struggling with type packing in MathComp

Future works

- Integration into MathComp ?
- A tactic to automatically derive eq/choice/count/finType properties of an Inductive
  - Very early stage, struggling with type packing in MathComp
  - Already done by Arthur Azevedo de Amorim ?

Benzaken, V., Contejean, É., and Dumbrava, S. (2017a).
Certifying Standard and Stratified Datalog Inference Engines in SSReflect.
In *International Conference on Interective Theorem Proving*, Brasilia, Brazil.

Benzaken, V., Contejean, É., and Dumbrava, S. (2017b).
Datalogcert.
https://framagit.org/formaldata/datalogcert/.

Garillot, F., Gonthier, G., Mahboubi, A., and Rideau, L. (2009).
Packaging mathematical structures.
In *International Conference on Theorem Proving in Higher Order Logics*, pages 327–342. Springer.

Sakaguchi, K. (2020).
Validating mathematical structures.
arXiv preprint arXiv :2002.00620.

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