A single number type for Math education in Type Theory

Yves Bertot

June 2024

The context

- Efforts to use theorem provers like Lean, Isabelle, or Rocq in teaching
- language capabilities, documentation, error message
  - Strong inspiration: Waterproof
  - similar experiment on Lean (Lean Verbose, for instance)
- Our contention: the contents also play a role
  - Several types for numbers, several versions of each operation
  - Coercions may be hidden, they can still block some operations
  - Type theory forces developers to define functions where they should be underfined
- Typing helps young mathematicians, but not the type of natural numbers

Characteristics of the natural numbers

- Positive sides
  - An inductive type
  - computation by reduction (faster than rewriting)
  - Proof by induction as instance of a general scheme
  - Recusive definitions are mostly natural
- Negative sides
  - Subtraction is odd: the value of $3 - 5$ is counterintuitive
  - The status of function/constructor `S` is a hurdle for students
  - In Coq, `S 4` and `5` and interchangeable, but `S x` and `x + 1` are not
  - The time spent to learn pattern matching is not spent on math
  - Too much cognitive load

Numbers in the mind of math beginners

- Starting at age 12, kids probably know about integer, rational, and real numbers
- $3 - 5$ exists as a number, it is not 0
- Computing $127 - 42$ yields a natural number, $3 - 5$ an integer, and $1/3$ a rational
- $42/6$ yields a natural number
- These perception are *right*, respecting them is time efficient

Proposal

- Use only one type of numbers: real numbers
  - Chosen to be intuitive for studends at end of K12
  - Including the order relation
- View other known types as subsets
- Include stability laws in silent proof automation
- Strong inspiration: the PVS approach
  - However PVS is too aggressive on automation for education
- Natural numbers, integers, etc, still silently present in the background

Plan

- Review of usages of natural numbers and integers
- Defining subsets of $\mathbb{R}$ for inductive types
- From $\mathbb{Z}$ and $\mathbb{N}$ to $\mathbb{R}$ and back
- Ad hoc proofs of membership
- Recursive definitions and iterated functions
- Finite sets and big operations
- Minimal set of tactics
- Practical examples, around Fibonacci, factorials and binomials

Usages of natural numbers and integers

- A basis for proofs by induction
- Recursive sequence definition
- iterating an operation a number of time $f^n(k)$
- The sequence $0 \dots n$
- indices for finite collections,
- indices for iterated operations $\sum_{i=m}^{n} f(i)$
- Specific to Coq+Lean+Agda: constructor normal forms as targets of reduction
- In Coq real numbers, numbers 0, 1, ..., 37, ... rely on the inductive type of integers for representation
  - In Coq, you can define `Zfact` as an efficient equivalent of factorial and compute 100!

Defining subsets of $\mathbb{R}$ for inductive types

- Inductive predicate approach
  - Inherit the induction principle
  - Prove the existence of a corresponding natural or integer
- Existence approach
  - Show the properties normally used as constructors
  - Transport the induction principle from the inductive type to the predicate
  - Hurdle: not possible to use the induction tactics if the type of data is not inductive

Inductive predicate in Coq

```
Require Import Reals.
Open Scope R_scope.

Inductive Rnat : R -> Prop :=
  Rnat0 : Rnat 0
| Rnat_succ : forall n, Rnat n -> Rnat (n + 1).
```

Generated induction principle:

```
nat_ind
     : forall P : R -> Prop,
       P 0 ->
       (forall n : R, Rnat n -> P n -> P (n + 1)) ->
       forall r : R, Rnat r -> P r
```

from $\mathbb{N}$ and $\mathbb{Z}$ to $\mathbb{R}$ and back

- Reminder: the types $\mathbb{N}$ (`nat`) and $\mathbb{Z}$ (`Z`), should not be exposed
- Injections `INR` and `IZR` already exist
- New functions `IRN` and `IRZ`
- definable using Hilbert's choice operator
  - Requires `ClassicalEpsilon`
  - use the inverse image for `INR` and `IZR` when `Rnat` or `Rint` holds

Degraded typing

- Stability laws provide automatable proofs of membership

```
Existing Class Rnat.

Lemma Rnat_add x y : Rnat x -> Rnat y -> Rnat (x + y).
Proof. ... Qed.

Lemma Rnat_mul x y : Rnat x -> Rnat y -> Rnat (x * y).
Proof. ... Qed.

Lemma Rnat_pos : Rnat (IZR (Z.pos _)).
Proof. ... Qed.

Existing Instances Rnat0 Rnat_succ Rnat_add Rnat_mul Rnat_pos.
```

- `typeclasses eauto` or `exact _.` will solve automatically `Rnat x -> Rnat ((x + 2) * x).`

Ad hoc proofs of membership

- When $n, m \in \mathbb{N}, m \leq n$, $(n - m) \in \mathbb{N}$ can also be proved
- This requires an explicit proofs
- Probably good in a training context for students
- Similar for division

Recursive functions

- recursive sequences are also a typical introductory subject
- As an illustration, let us consider the *Fibonacci* sequence
  The Fibonacci sequence is the function F *such that* $F_0 = 0$, $F_1 = 1$, *and* $F_{n+2} = F_n + F_{n+1}$ *for every natural number* n
- Proof by induction and the defining equations are enough to *study* a sequence
- But *defining* is still needed
- Solution: define a recursive definition command using Elpi

Definition of recursive functions

- We can use a *recursor*, mirror of the recursor on natural numbers
- `Rnat_rec : ?A -> (R -> ?A -> ?A) -> R -> ?A`
- Multi-step recursion can be implemented by using tuples of the right size

```
(* fib 0 = 0  fib 1 = 1               *)
(* fib n = fib (n - 1) + fib (n - 2) *)

Definition fibr := Rnat_rec [0; 1]
   (fun n l => [nth 1 l 0; nth 0 l 0 + nth 1 l 0]).
```

Meta-programming a recursive definition command

- The definition in the previous slide can be generated
- Taking as input the equations (in comments)
- The results of the definition are in two parts
  - The function of type `R -> R`
  - The proof the logical statement for that function

```
Recursive (def fib such that
             fib 0 = 0 /\
             fib 1 = 1 /\
             forall n : R, Rnat (n - 2) ->
                fib n = fib (n - 2) + fib (n - 1)).
```

Compute with real numbers

- `Compute 42 - 67.` yields a puzzling answer
  - Tons of `R1`, `+`, `*`, and parentheses.
- `Compute (42 - 67)%Z.` yields the right answer
  - Except it is in the wrong type

Compute with real numbers: our solution

- New command `R_compute`.
- `R_compute (42 - 67).` succeeds and displays -25
- `R_compute (fib (42 - 67)).` fails!
- `R_compute (fib 42) th_name.` succeeds and saves a proof of equality.
  - Respecting the fact that `fib` is only defined for natural number inputs
- Implemented by exploiting a correspondance between elementary operations on `R`, `Z` (with proofs)
- ELPI programming
- Mirror a recursive definition in `R` to a definition in `Z`
- Correcness theorem of the mirror process has a `Rnat` on the input.

Finite sets of indices

- Usual mathematical idiom : $1 \ldots n$, $0 \ldots n$, $(v_i)_{i=1 \ldots n}$
- Provide a `Rseq : R -> R -> R`
  - `Rseq 0 3 = [0; 1; 2]`
- Using the inductive type of lists here
- This may require explaining structural recursive programming to students
- At least `map` and `cat` (noted `++`)

Big sums and products

- Taking inspiration from Mathematical components
- `\sum_(a <= i < b) f(i)`
  - Also `\prod`
- Well-typed when `a` and `b` are real numbers
- Relevant when $\mathtt{a} < \mathtt{b}$
- This needs a hosts of theorems
  - Chipping off terms at both ends
  - Cutting in the middle
  - Shuffling the indices
- Mathematical Components `bigop` library provides a guideline

Iterated functions

- Mathematical idiom : f^n, when $f : A- > A$
- We provide `Rnat_iter` whose numeric argument is a real number
- Only meaning full when the real number satisfies `Rnat`
- Useful to define many of the functions we are accustomed to see
- Very few theorems are needed to explain its behavior
  - $f^{n+m}(a) = f^n(f^m(a)) \quad f^1(a) = f(a) \quad f^0(a) = a$

Minimal set of tactics

- `replace`
  - `ring` and `field` for justifications
  - No need to massage formulas step by step through rewriting
- `intros`, `exists`, `split`, `destruct` to handle logical connectives (as usual)
- `rewrite` to handle the behavior of all defined functions (and recursors)
- `unfold` for functions defined by students
  - But we should block unfolding of recursive functions
- `apply` and `lra` to handle all side conditions related to bounds
- `typeclasses eauto` to prove membership in `Rnat`
  - Explicit handling for subtraction and division
- Possibility to add ad-hoc computing facilities for user-defined
  - Relying on mirror functions computing on inductive `nat` or `Z`

Demonstration time

- A study of factorials and binomial numbers
  - Efficient computation of factorial numbers
  - Proofs relating the two points of view on binomial numbers, ratios or recursive definition
  - A proof of the expansion of $(x + y)^n$
- A study the fibonacci sequence
  - $\mathcal{F}(i) = \frac{\phi^i - \psi^i}{\phi - \psi}$ (ϕ golden ratio)

The "17" exercise

- Prove that there exists an n larger than 4 such that

$$\binom{n}{5} = 17 \binom{n}{4}$$

(suggested by S. Boldo, F. Clément, D. Hamelin, M. Mayero, P. Rousselin)

- Easy when using the ratio of factorials and eliminating common sub-expressions on both side of the equality

$$\frac{\cancel{n}!}{\cancel{(n-5)!}\cancel{5!}_5} = 17 \frac{\cancel{n}!}{\cancel{(n-4)!}_{(n-4)}\cancel{4!}}$$

- They use the type of natural numbers and equation

$$\binom{n}{p+1} \times (p+1) = \binom{n}{p} \times (n-p)$$