Evgeny Makarov’s modular axiomatisation of arithmetic in Coq
The status of arithmetical libraries in Coq

Reached a point where

- The same theories were implemented in many ways
- Each implementation came with its own same lemmas (commutativity, cancellation, transitivity, ...).
- Most theories were the result of an accumulation of small additions from various users, often by non-mathematician users, often anarchically.
The same theories were implemented in many ways

- **natural numbers:**
  
  nat: the simplest Peano definition, the most extended library (Arith)

  N: base-2 absolute value, under-developed (NArith)

- **integers:**
  
  Z: the oldest: sign + base-2 absolute value (ZArith)

  bigZ: quotient over sign + base-$2^{31}$ absolute value (Ints) [V8.2 only, by Spiwack]

- **cyclic integers:**
  
  znz: Grégoire and Théry’s bounded integers in tree form [V8.2 only]

  int31: efficient (because mapped to machine words) [V8.2 only, by Spiwack]
Makarov’s objectives

Factorisation of the proof of the properties shared by the different implementations.

(Re-)introducing a coherence and a conformity to the mathematical usage in the different implementations (see Freek’s comment on the “one-man” libraries)

- e.g. commutativity of addition was referred to as a symmetry property!
- addition was called plus but multiplication was called mult!
- less than and greater than were distinct definition, leading to many redundancies
- and often the same property had not the same name depending on the implementation
- use of variable names were anarchic in each implementation (n, a, x, ...)

Provide support for implementations defined by a congruence

Incidentally, testing the module system and setoid rewriting!

At the end, a new implementation of integers as pairs of natural numbers.
Extensions to the module system
Inlining definitions at functor application

Module Type T.
Parameter Inline T : Type.
End T.

Module F (X:T). Definition U := X.T. End F.

Module N. Definition T := nat. End N.

Module A := F N.

Print A.U.
(* A.U = nat : Type *)

(was based on previous work by Claudio Sacerdoti)
Parametric modules

Module Type T. End T.

Module Type S (M:T). . . . . End F.

Module N : T. . . . . End N.
Module A : S N. . . . . End A.

Not ready in time and finally not used.
Sharing submodules in a diamond configuration

Module F. Parameter a:nat. End F.

Module A.
  Module F1 := F.
  (* ... *)
End A.

Module B.
  Module F2 := F.
  (* ... *)
End B.

Module C.
  Module A_in_C := A.
  Module B_in_C := B.
  Goal A_in_C.F1.a = B_in_C.F2.a. reflexivity. Qed.
  (* OK, but everything doubled *)
End C.
Sharing applied subfunctors in a diamond configuration

Module Type T. End T.
Module F (X:T). Parameter a:nat. End F.

Module A (X:T).
  Module F1 := F X.
  (* ... *)
End A.

Module B (X:T).
  Module F2 := F X.
  (* ... *)
End B.

Module C (X:T).
  Module A_in_C := A X.
  Module B_in_C := B X.
  Goal A_in_C.F1.a = B_in_C.F2.a. reflexivity. (* fails *)
End C.
Module dependent over simple declarations

Module F (Parameter a:T).
...
End F.
Module inclusion

Module F (X:T)
    Definition u := X.t.
End F.

Module G (X:T)
    Module P := F X.
End G.

Module M
    Module N := G X.
    (* has N.P.u; we would prefer u *)
End M.
Module F (X:T)
   Definition u := X.t.
End F.

Module M
   Module N := F X renaming u into v.
End M.
About the axiomatics

see the development version of Coq at

https://gforge.inria.fr/plugins/scmsvn/viewcvs.php/trunk/theories/Numbers/?root=coq
A deliberate axiomatisation choice

Specification in pure first-order logic + induction schemes.

*Specifications are as "universal" as possible*

Implementation may use arbitrary specific features of the logic (inductive structures, well-founded fixpoint, ...).

*Implementations can exploit at best the features of the underlying programming language of the logic*
Granularity of the modularization

Incremental axiomatisation of $+$ and $\times$ was too complex with the module system.

Hence all of $\text{succ}$, $\text{pred}$, $+$, $-$, $\times$ and their properties are the axiomatics.

In a second step, an extended signature axiomatises $<$ and $\leq$ and their properties.
The arithmetic language as a language of choice for testing communication between PAs, CAS and ‘Wikipedia’

Simple and rather well understood but still some work to make it fitting the mathematician practice.

Incidentally, most interesting statements are $\Pi_2^0$ and the classical vs constructive discussion does not apply...
Examples from Wikipedia

\[ \binom{n}{p} = \frac{n!}{p!(n-p)!} \]

It relies on subtraction and division. How to represent them? I think that we can do better than working in \( \mathbb{Q} \) or \( \mathbb{R} \) so that division is exact, as shown in Herman’s picture. The same for subtraction. After all, number theory is about integer numbers and it should not need more concepts.

A mathematician-oriented number theory language would then typically declare

\[
\forall n : \mathbb{N} \quad n! : \mathbb{N} \\
\forall np : \mathbb{N} \quad n - p : \mathbb{N} \quad \text{with side condition: } n \geq p \\
\forall np : \mathbb{N} \quad \frac{n}{p} : \mathbb{N} \quad \text{with side condition: } p \neq 0 \land \exists q \ n = q \times p
\]

So that the definition above is mapped to the formal content

Definition of binomial coefficient is

Assumptions: \( np : \mathbb{N} , n \geq p : \text{PROP} \)
Proof obligation: \( n \geq p \)
Proof obligation: \( \exists q \ n! = q \times n!(n-p)! \)
Body: \( \binom{n}{p} = \frac{n!}{p!(n-p)!} \)