The Book

Assia, Enrico, . . . (you are welcome)

November 20, 2014

Contents

0 The essence of math comp **5**

0.1 challenges faced and tools adopted 5

0.2 Computational Thinking 5

0.3 logic programming ... type inference 6

0.4 automation in tactics 6

0.5 discipline 6

0.6 trivial=implicit (for a trained mathematician) 7

I The art of formalizing **9**

1 Logics **11**

2 Programming **13**

3 Proofs **15**

4 Type inference/Genericity **17**

5 Mixing data and proofs **19**

6 Hierarchy **21**

7 Larger scale reflection (out of place) **23**

II Mathematics in Mathematical Components **25**

7.1 STYLE of these chapters 27

8 Numbers **29**

9 Polynomials, Linear algebra **31**

10 Quotients **33**

11 Finite group theory **35**

12 Representation theory, Character theory **37**

13 Galois Theory **39**

14 Real closed fields **41**

15 Algebraic closure **43**

III Annexes **45**

16 What is done where? **47**

17 How tos **49**

18 Naming conventions **51**

19 Index of notations **53**

Chapter 0

The essence of math comp

0.1 challenges faced and tools adopted

(tools in a broad sense, the logic is a tool, coq is a tool, the plugin is a tool, the ssr style is a tool,...)

Challenges:

- large body (scale up), make proofs small and robust. We need to say that we do use "deterministic automation". Use Laurent's data on de bruijn factor.
- model the use of math notations, their role in proofs, model proofs (also it is about reasoning, not just computations). Another way to say that is: model Bourbaki (rationalization of Math via structure/interfaces) but not the first book (set theory) that is replaced by CIC (link with section computational thinking).

We build on Coq and an extension. The main tools follow (in random order):

0.2 Computational Thinking

This section should motive the activity of formalizing mathematics with the **Coq** proof assistant, emphasizing its computing skills, and as opposed to other foundations like HOL. However the challenge is to keep mathematicians as the privileged target, while motivating the ssr approach to a CS oriented reader.

See `../coq/ch0.v`.

Aim of the chapter:

- should sound natural and easy to a CS person (but with the ssr twist)
- should sound different but well motivated to a Coq user (do show, maybe in the exercises, that leqn is 100 times better than "Inductive le"). Try to reproduce the shock we had the first time we used Boolean predicates. It

may help to compare, in the *advanced* section, the approach with the standard one, so that one sees two proof scripts in the same page.

0.3 logic programming ... type inference

to model proof search, give a meaning to notations, teach coq the work an informed reader does (contextualizing otherwise ambiguous notations, knowledge of interface/instance of algebraic structures).

relation with proof search: no "blind" proof search (easy, ad-hoc, pervasive v.s. advanced, generalistic, potentially expensive and unstable).

0.4 automation in tactics

the main points:

- 1/3 is rewrite, term selection/search (one does not need to reach a sub formula as a goal in order to make progress for example, no monkey puzzle as in GG terminology).
- Create a formula without writing it: some advanced forms of forward reasoning tactics to deal with symmetries, generalizations (boils down to syntesize the cut formula out of the minimum possible user input, as in a text where one says "similarly to that, we can also prove that". (this is not very pervasive, dunno if it is worth putting it here in this chapter). Also elim does that. (technically also rewrite, but we may want to separate things)

This section is were one talks about the plugin, and some of the main design points of tactics: compositional (a language, not a list of commands), predictable (documented!!!), finally compact (symbols for uninteresting steps).

0.5 discipline

MAKE an howto out of that. Maybe one should also add a few notes on the style in scripts? like:

- you must be able to model (at least) 1 proof step in a sentence (line), e.g. "rewrite preparegoal dostep ?cleanup."
- uninteresting/recurrent lemmas/steps should be small (short names, easy to gray out)
- lemma statements are designed, not just written, having in mind their use (forward, backward, implicit argumets, arguments order) and the class of trivial hypotheses since an extra hyp that is proable triviality (via //, hint resolve, cnaonical) is for free. E.g. "x
is a toto", "0 ¡= n", ...

- also not every possible lemma, but a few that combine well
- proofs/definitions are reworked many times, why (understand recurrent proof schemas, compact, factor, make more stable/robust) and what is needed (like meaningful names, clear structure)

0.6 trivial=implicit (for a trained mathematician)

The idea is to try to identify what is trivial (mathematically speaking) and be sure you can model it as such:

- (level basic) make explicit the trivialities of each theory (what one expects to be proved by //).
- (level advanced) when you do new stuff, you must decide what is trivial/implicitly proved.
- (hard) which technique to make Coq prove it automatically (hint resolve, canon, comput... in the type)

This may also be another way present the whole chapter

mantra: (basic) if you see toto=false you should perform the case analysys via fooP

Part I

The art of formalizing

Chapter 1

Logics

From calculability to proofs, hence the CC, and the fact that reasoning principles without a computational content become axioms.

This is a non technical chapter and message should be:

- instantiation of a universal statement is application (also the pair)
- Excluded middle is not available by default (choice?)
- Conversion as a pervasive indistinguishably, what inside (beta, definition unfolding,...)
- Dependent types: eq, sigma (which example?)

One options is: avoid relating type theory and other logics. We say: we have a formal game where the basic elements are programs/functions that come with types to avoid confusion. full stop. (no relation with proof theory, set theory). maybe mention that roots are in calculability (hence the choice to pick functions as primitive and not sets). This is lucky because (computable) functions are today executable by a computer. Still not all concepts are ”computable” hence some principles are problematic: EM,.... we mainly stay in the lucky fragment (again no propaganda on intuitionistic logic, constructive math; just a mention).

Chapter 2

Programming

Presentation of inductive data structures, recursive programs on these data. Bool, Boolean connectives, Boolean reflection (cf ch. 0.2), views. Examples of equality tests (==, with a forward reference to explain the magic if needed), operations on sequences, nats, exercises on prime, div.

Chapter 3

Proofs

Where one learns to do proofs. Boolean reflection in practice, views, discussion on the definition of leq, proofs on things defined in the previous chapter, associated tactics, exercises on prime, div, binomial, etc.

spec? A new vernacular to declare specs without typing coinductive and by writing explicitly the equations.

Chapter 4

Type inference/Genericity

put here a minimalistic presentation of records.

Implicit arguments (Is it possible to survive without implicit argument up to this point? We should probably use them plus forward references to here), canonical structures. Combining with notations. Now the real ==.

Chapter 5

Mixing data and proofs

Boolean sigma types, records, coercions, UIP. Examples: ordinals, tuples (not their use which requires CS). This chapter might come after some presentation of basic canonical structures, i.e. reorganize the content between this chapter and the next.

example of tuples here? better ordinals?

GG: makes many points here (not fully understood by Enrico):

- ordinals/tuples are easy to use but hard to build
- it is a tradeoff, but is not clear if we can give hints on when a specific datatype like ordinals is better that unpackaged stuff.

UIP is advanced, the basic user should just be told that putting bools inside a record is just fine.

- record (flat)

Chapter 6

Hierarchy

Packaging records, the bigop hierarchy. Scaling with packed classes and mixins, to the ssralg hierarchy. Presentation of the content of ssralg in terms of structures and of the theory? Should the latter be a separate chapter.

Maybe a plugin for a new vernacular to script the creation/declaration of structures/instances so that the level basic can touch the argument easily.

Explain what the abstraction barrier is (like unfolding a GRing projection)

gotcha: if you see GRing.toto then you broke an abstraction barrier

mantra: if you have a proof in mind, don't let the system drive you to another, less clean and abstract, proof.

Declaring an instance is hard... we need to document the multuiple processes for each structure in each hierarchy and possibily make a program out of it.

Chapter 7

Larger scale reflection (out of place)

Four colours, decomposition in primes, example in peterfalvi. In particular example where a case analysis does not follow the constructors of an existing inductive type: then craft an ad hoc one to hint the proof. This topic is probably at a wrong place.

Part II

Mathematics in Mathematical Components

7.1 STYLE of these chapters

These chapters hopefully in the following style:

- first math mode to review the (not so standard) definitions and
- then touch some real proof script to show why/how the definition make it possible to model the proofs (as they are in math)
- use the (few) examples to illustrate a cool proof strategy if any (not necessarily typical of the math subject, but that happens to be there, like in OOTHM paper: circular leq, symmetries, ad hoc decision procedure, ...)
- try hard to show how CIC helps (which feature: HO, computation, dependent types . . .), so that at the end we can sum up and make a synthesis of all that.

Maybe it is simpler to do it in 2 steps: 1) in this second part one identifies where CIC or the SSR style plays a crucial role 2) then we advertise these use cases in the first part to motivate the techniques, the complexity of the logic...

Chapter 8

Numbers

What are the numbers available in the library? How to use them, casts between types... Non trivial point in the formalisation: axiomatization, order is defined from norm, partial orders (in particular for complex numbers).

Chapter 9

Polynomials, Linear algebra

2-stage presentation: interface plus explicit. Expansion of Georges'ITP paper. Here is one of the main application of the choice operator (complement a base).

Chapter 10

Quotients

Chapter 11

Finite group theory

Data-structures, how to craft the set of variants of a same theorem to make the formalization handy. Permutations. Presentations?

Chapter 12

Representation theory, Character theory

As an example of application, in particular of the linear algebra theory.

Chapter 13

Galois Theory

Chapter 14

Real closed fields

Chapter 15

Algebraic closure

Part III

Annexes

Chapter 16

What is done where?

Chapter 17

How tos

suggestions:

- Get more information when you do not understand the error message
- Search in the library
- Canonical structures: define a new instance
- Canonical structures: add a new structure
- Give a relevant name to the lemma I just stated
- Forbid unwanted expansions
- Choose a notation (what not to do...)
- Compute “for real” (Natbin, Coqeal)
- MathComp script style

Chapter 18

Naming conventions

Chapter 19

Index of notations