# The “Initiation to formal proofs” course

Pierre Rousselin with Marie Kerjean and Micaela Mayero

Université Paris XIII dite Sorbonne Paris Nord, Villetaneuse

Coq Workshop 2023, July $31^{\text{st}}$

The `.v` and `wacoq` `html` files of the courses are here:

`https://www.math.univ-paris13.fr/~rousselin/ipf.html`

But

- in French
- needs polishing (working on it at the moment)
- not possible at the moment to save and load with `wacoq` (Emilio Jesús Gallego Arias and Shachar Itzhaky are working on it)

Context

Organization of the course

Content

Impressions, feedbacks

# Public

- Université Sorbonne Paris Nord, Villetaneuse (northern suburbs of Paris, has sheeps)

- First year undergraduate students in maths+CS double major
- two groups of $\approx 25$ students each
- mandatory but only 1 credit out of 30 for the first semester

# History of the course

- first edition in fall 2021
- new specific course for these students
- at the interface of maths and CS
- focused on activity and rigour
- replaces a 18h methodology course, so only 18h (6 practice sessions of 3h each) for this course.

# Other courses in France using proof assistants

- Patrick Massot with Lean at Orsay. Uses a custom set of tactics (Lean-Verbose written by Patrick Massot).
- Frédéric Le Roux with Deaduction (a GUI built over Lean written by Frédéric Le Roux) at Sorbonne Université (Jussieu).
- Julien Narboux with Edukera (GUI built over Coq, commercial) at Strasbourg.
- Simon Modeste with Edukera at Montpellier.
- See: Utilisation des assistants de preuves pour l'enseignement en L1 : Retours d'expériences. La gazette des mathématiciens, 2022, 174
- A dedicated summer school this year: Proof Assistants for Teaching (PAT 2023).

# Goals of the course

- ??

# Goals of the course

- ??
- The aim is actually to make the students write formal proofs of mathematical statements...

# Goals of the course

- ??
- The aim is actually to make the students write formal proofs of mathematical statements...
- ... in the hope that it will help them in their maths+CS studies.

# Goals of the course

- ??
- The aim is actually to make the students write formal proofs of mathematical statements...
- ... in the hope that it will help them in their maths+CS studies.
- Work on rigour and problem-solving.
- Side goal: create a group dynamic in “double-licence” by giving the students a challenging specific course at the beginning of the year.

# What's this talk about?

# What's this talk about?

- ~~My indisputable expertise in Coq, dependent type theory and other secrets of the universe(s).~~

# What's this talk about?

- ~~My indisputable expertise in Coq, dependent type theory and other secrets of the universe(s).~~
- content of our course
- advice for teachers who would like to try this
- difficulties (from Math or Coq) for the students (or myself!)
- random thoughts about how it could get better
- random plans for the future (this course is a living thing)
- in the hope that it might be helpful and stimulate interesting discussions

Context

Organization of the course

Content

Impressions, feedbacks

# Choices

- We had to make some choices under many constraints (duration of the course, lack of time to prepare new content, etc) and external influences (people to work with, *Software Foundations* by Pierce and al., ...)
- These choices are certainly not the only possible ones and very likely not the best ones.
- Our goal in this talk is not to say “one should choose this”, but rather “we chose this” and explain, when possible, why we did so.

# Choices and preparation

- only hands-on practical sessions (no lectures) with homeworks
- As in *Software Foundations*, the course is a set of `.v` source files with examples, exercises and comments.
- The comments and the behavior of the tactics actually replace the lectures. In a way, Coq is one of the teachers.

# Choices (2)

- Do not hide (too much) stuff: it's ok to talk about intuitionistic logic, right-associativity of `->`, ...
- Passionate students should be able to write their own theorems and prove them (autonomy).
- However, writing your own functions or types is not an objective.
- The maths+CS side is embraced: it's ok to write ascii bytes in a file using a text editor.
- Restrictions:
  - no booleans (two logics would be too much)
  - no inductive propositions (too much to digest in such a small course)

# Starting point

- limits of sequences as final goal
- prerequisites: logics, natural numbers and real numbers.

# Plan of the course

- Propositional (intuitionnist) logic (+ additional exercises)
- Natural numbers and induction (+ additional exercises)
- Predicate calculus ("Set theory" à la Coq) (+ additional exercises)
- First homework assignment
- Real numbers as a field (algebra)
- Second homework assignment
- Real numbers as an ordered field
- Absolute value and distance on real numbers
- Convergence of real-valued sequences
- Final test

# Practical matters

- Practical Sessions rooms: about 15 computers with Debian, Coq already installed (`8.12`...) with CoqIDE, worked fine.
- The students also had to install it on their own machines for homework: links to installers in the Coq Platform worked fine on Windows, MacOS and Linux.
- Students used CoqIDE.
- The `.v` files where hosted on a private "moodle" page at the university (also used by students to upload their works).

# Contract

- Strong implicit contract between students and teachers:
- Every exercise should be feasible with what has previously been shown.
- Always start with an example, followed immediately by a very easy exercise.
- Reduced number of tactics.
- Keep the information flow manageable. (This is the hard part for the teacher.)
- For the files, we start with the solution and some tags (*(* Début Solution *)* and *(* Fin Solution *)*). Some bleeding edge technology (`sed`) is then used to generate both the subject and the solution.

Context

Organization of the course

Content

Impressions, feedbacks

# LogiquePropositionnelle.v

- The aim is to introduce the usual connectors of propositional (intuitionnist) logic.
- We always start with a commented example...
- ... followed by exercises (of which the first at least should be *very* easy).
- Always two sides: “how to prove it?” and “how to use it?”
- The order is `->`, `and`, `or`, `False` and `not`,
- In the end and in the additional exercises, the excluded middle is discussed and used (but we don’t really need it explicitly in the rest of the course...)

# Tactics for propositional logic

- `->` `intros` to prove
- `->` `apply` to use, only with backward reasoning at this point, in the form `apply H.` with `(H) : ?  -> E` and the goal of type `E`.
- `/\` `split` to prove, `destruct` to use
- `\/` `left` or `right` to prove, `destruct` to use (proof by exhaustion), students should choose the side carefully and at the last possible moment.
- `<->` `split` to prove, `destruct` to transform into two `->`
- `False` `destruct` to use and prove anything, `exfalso` to change any goal to `False`, (`unfold not`), `intros` to prove.

Conclude with `exact` or `assumption`.

# Some Coq issues

- We try to *restrict* the usage of tactics so that they have a predictable outcome close to natural deduction rules...
- ... but students try things (sometimes at random).

# Some Coq issues

- We try to *restrict* the usage of tactics so that they have a predictable outcome close to natural deduction rules...
- ... but students try things (sometimes at random).

Consider:

```
P, Q : Prop
H : P /\ Q
========================= (1 / 1)
Q
```

# Some Coq issues

- We try to *restrict* the usage of tactics so that they have a predictable outcome close to natural deduction rules...
- ... but students try things (sometimes at random).

Consider:

```
P, Q : Prop
H : P /\ Q
========================= (1 / 1)
Q
```

- Expected proof (at this point):
  ```
  destruct H as [H1 H2].
  exact H2.
  ```

# Some Coq issues

- We try to *restrict* the usage of tactics so that they have a predictable outcome close to natural deduction rules...
- ... but students try things (sometimes at random).

Consider:

```
P, Q : Prop
H : P /\ Q
========================= (1 / 1)
Q
```

- Expected proof (at this point):
  ```
  destruct H as [H1 H2].
  exact H2.
  ```
- A possible proof:
  ```
  apply H.
  ```

# Some Coq issues

- We try to *restrict* the usage of tactics so that they have a predictable outcome close to natural deduction rules...
- ... but students try things (sometimes at random).

Consider:

```
P, Q : Prop
H : P /\ Q
========================= (1 / 1)
Q
```

- Expected proof (at this point):
```
destruct H as [H1 H2].
exact H2.
```
- A possible proof:
```
apply H.
```
- Is `Set Poussin.` possible?

Par fir0002flagstaffotos [at] gmail.com, GFDL 1.2, https://commons.wikimedia.org/w/index.php?curid=135312

# Naturels.v

- Peano's natural numbers
- Coq can compute (`Fixpoint`, `simpl`, `Compute`, `discriminate`)
- `rewrite` and unification
- `induction`
- Natural number game (associativity and commutativity of multiplication from scratch)
- (`injection` and `f_equal`)

# Some Coq issues : `rewrite` and unification

Example about unification and `rewrite`:

```
n : nat
============================ (1 / 1)
0 + (1 + (1 + n)) = S (S n)
```

# Some Coq issues : `rewrite` and unification

Example about unification and `rewrite`:

```
n : nat
========================== (1 / 1)
0 + (1 + (1 + n)) = S (S n)
```

```
rewrite add_1_l.
```

```
n : nat
========================== (1 / 1)
0 + S (1 + n) = S (S n)
```

# Some Coq issues : `rewrite` and unification

Example about unification and `rewrite`:

```
n : nat
========================== (1 / 1)
0 + (1 + (1 + n)) = S (S n)
```

```
rewrite add_1_l.
```

```
n : nat
========================== (1 / 1)
0 + S (1 + n) = S (S n)
```

```
rewrite add_1_l.
```

# Some Coq issues : `rewrite` and unification

Example about unification and `rewrite`:

```
n : nat
========================== (1 / 1)
0 + (1 + (1 + n)) = S (S n)
```

```
rewrite add_1_l.
```

```
n : nat
========================== (1 / 1)
0 + S (1 + n) = S (S n)
```

```
rewrite add_1_l.
```

```
Tactic generated a subgoal identical to the original goal.
```

# Some Coq issues : `rewrite` and unification

Example about unification and `rewrite`:

```
n : nat
========================= (1 / 1)
0 + (1 + (1 + n)) = S (S n)
```

```
rewrite add_1_l.
```

```
n : nat
========================= (1 / 1)
0 + S (1 + n) = S (S n)
```

```
rewrite add_1_l.
```

```
Tactic generated a subgoal identical to the original goal.
```

In practice it was mostly ok, but `rewrite` is not very predictable (at least for a beginner).

# Some Coq issues : `rewrite` and unification

Example about unification and `rewrite`:

```
n : nat
========================= (1 / 1)
0 + (1 + (1 + n)) = S (S n)
```

```
rewrite add_1_l.
```

```
n : nat
========================= (1 / 1)
0 + S (1 + n) = S (S n)
```

```
rewrite add_1_l.
```

```
Tactic generated a subgoal identical to the original goal.
```

In practice it was mostly ok, but `rewrite` is not very predictable (at least for a beginner).
`Set Keyed Unification.` ?
When in doubt, instantiate manually?

# Some Coq/Maths Issues (2) : `simpl`

- `simpl` is very sensitive to somewhat arbitrary choices of definition.
- `simpl` sometimes gives you a lot more than what you wished for.
- Some "encapsulation lemmas" (e.g. `add_succ_l`) could (maybe?) offer finer control using `rewrite` (à la *rewriting rules*).

# Some Coq/Maths Issues (3) : `induction`

- `induction` is sensitive to the order of universally quantified variables in the goal.
- One probably wants to avoid the `generalize dependent` tactic at this level (at least under these time constraints). You probably want to choose (or write) your exercise with this in mind.
- Another (Math) difficulty is that sometimes one should not draw `induction` immediately.
- Some exercises, not meant to be difficult, proved to be a lot harder in practice for the students, because of too early `induction` or `simpl`. So TODO: make this clearer for the students.

# `CalculPredicat.v`

- Existential quantifier: using (`destruct`) and proving (`exists`).
- "Subsets of a type `A`", actually `A -> Prop`.
- Injections, surjections, bijections
- This is an important part, because we know from experience that it is a strong mathematical difficulty for students.
- I would advise to stay in the intuitionnistic world a little bit before moving to the classical world (with more than 1 way to prove an existential formula).

# `CorpsOrdonné.v`

We start working with Coq's `Reals`.

- "Axioms" of an ordered field
- No more computation, only `rewrite`
- "Real numbers game": from "axioms" to $0 < 1$
- We progressively introduce forward reasoning (`apply ... in`, `rewrite ... in`, `assert`, `replace`).

# RInégalités.v and Rabs_R_dist.v

- New in 2022
- Students struggle with inequalities and absolute values.
- They needed more exercises before studying sequences.

# Suites.v

- Before studying sequences, automation is shown (file `Auto.v`).
- Actually some inequalities on $\mathbb{N}$ could **not** be proved manually by students.
- First analysis lemma:

```
Lemma small_zero: forall x,
  (forall eps, eps > 0 -> (Rabs x) < eps) -> x = 0.
```

- The given example is:

```
Theorem UL_sequence (Un : nat -> R) (l1 l2 : R) :
  Un_cv Un l1 → Un_cv Un l2 → l1 = l2.
Proof.
  unfold Un_cv.
  intros Hl1 Hl2.
  (* On va montrer que la distance entre l1 et l2
     est aussi petite qu'on veut. *)
  apply small_dist_equal.
  (* Soit eps > 0. *)
  intros eps Heps.
  (* Soit n1 tel que pour tout n >= n1, |Un - l1| < eps / 2. *)
  destruct (Hl1 (eps / 2)) as [n1 Hn1]. lra.
  (* Soit n2 tel que pour tout n >= n2, |Un - l2| < eps / 2. *)
  destruct (Hl2 (eps / 2)) as [n2 Hn2]. lra.
  (* Soit n3 = max(n1, n2). *)
  remember (max n1 n2) as n3 eqn:n3_max.
  (* ... *)
Qed.
```

# A remark on multiple definitions

We may use many different definitions to say the same things.
Consider: $(u_n)$ goes to $+\infty$.

```
Definition cv_infty (Un:nat -> R) : Prop :=
  forall M, exists N : nat, (forall n:nat, (N <= n)%nat -> M < Un n).
```

Other equivalent definitions:

- $\forall M > 0, \exists N \in \mathbb{N}, \forall n \geq N, u_n > M$ (restriction on $M$).
- $\forall M > 0, \exists N \in \mathbb{N}, \forall n \geq N, u_n \geq M$ (restriction on $M$ + weaker conclusion).

This might look innocent... but it feels really weird, when for instance one actually has to consider the case $M \leq 0$ to prove that $n \to \infty$.

# A remark on multiple definitions

In general, when we want

- to *prove* that a property holds, we want the strongest hypotheses and the weakest conclusion; here, for instance, $\forall M > 0, \exists N \in \mathbb{N}, \forall n \geq N, u_n \geq M$.
- to *use* the fact that some property holds, we want the weakest hypotheses and the strongest conclusion.
  Here, for instance, $\forall M, \exists N \in \mathbb{N}, \forall n \geq N, u_n > M$.
- Could we imagine some Coq support for multiple equivalent definitions? For instance `unfold cv_infty%2.` to select from a list of (proved to be) equivalent defintions?

Context

Organization of the course

Content

Impressions, feedbacks

# In the end

- Propositional logic with natural deduction is well understood
- Almost all students can prove simple equalities in $\mathbb{N}$ by induction
- Some difficulties with predicate calculus, but knowing that it would be hard helped this year
- Working with real numbers is mostly ok with equations, inequalities are harder (but this gets better with practice).
- In 2021, only one student managed to prove a non-trivial analysis theorem. In 2022, about 6 of them proved a significant part of the `Suites.v` file. Can this be increased with more hours? more polishing?

# In the end (2)

- It is not really possible at this point to quantify the impact of this course on the students.
- It is clear though that it helped create a solid "double-licence" group.
- During the second semester, the avarage grade in double-licence this year was about 14/20 (in France this is *really good*), usually 5 more points than computer science students, with the same courses.
- A group of students was very willing to continue with formal proofs (unfortunately, I didn't manage to find time to write more exercises...)

# What should be a Coq file for teaching?

- `jscoq` or `wacoq`?
- `html`? `markdown`?
- Mathematical formulas?
- Figures?
- Multilingual document?
- Better (multilingual) error messages?

# What should be a Coq file for teaching?

- `jscoq` or `wacoq`?
- `html`? `markdown`?
- Mathematical formulas?
- Figures?
- Multilingual document?
- Better (multilingual) error messages?

  No product even after head-reduction.

# What should be a Coq file for teaching?

- `jscoq` or `wacoq`?
- `html`? `markdown`?
- Mathematical formulas?
- Figures?
- Multilingual document?
- Better (multilingual) error messages?

No product even after head-reduction.

versus

No more variables or hypotheses to introduce.

# What should be a Coq file for teaching?

- `jscoq` or `wacoq`?
- `html`? `markdown`?
- Mathematical formulas?
- Figures?
- Multilingual document?
- Better (multilingual) error messages?

No product even after head-reduction.

versus

No more variables or hypotheses to introduce.

or if the selected language is French:

# What should be a Coq file for teaching?

- `jscoq` or `wacoq`?
- `html`? `markdown`?
- Mathematical formulas?
- Figures?
- Multilingual document?
- Better (multilingual) error messages?

No product even after head-reduction.

versus

No more variables or hypotheses to introduce.

or if the selected language is French:

Il n'y a plus ni variable ni hypothèse à introduire.

# Food for thoughts

- What is the real value of this course for students?
- When and how should we introduce forward reasoning?
- When and how should we introduce automation?
- How to go from Coq proofs to pen and paper proofs?
- What's next?