Proof Assistants – TP. 2

Bruno Barras

Dec 13, 2021

1 More on tactics

1.1 Making proof scripts more robust: naming hypotheses

Tactics such as `intro` or `destruct` introduce new hypotheses. Coq chooses a name for them, depending on their type. However, this leads to hypotheses with similar names (`H`, `H0`, `H1`, etc.) and this makes proof scripts more fragile: suppose we change the script and the hypothesis `H0` is not generated; then the one that was called `H1` is now called `H0`; `H2` becomes `H1`, etc. This requires to edit the name of hypotheses at many irrelevant places.

It is advised to use variants of the above tactics that let the user name the hypotheses:

- `intros hyp1 hyp2.` introduces two hypotheses with the given names
- `destruct hyp as (x,y,z)` applies an elimination rule to `hyp` which shall produce three hypotheses that will be named `x`, `y` and `z`. It is an error if the number of variables does not correspond to the number of hypotheses generated. Both conjunction and existential produce two hypotheses.
- `destruct hyp as [x1 y2|x2 H2].` is used when the elimination generates several subgoals (for instance, disjunction). Each list of names, separated by `|`, correspond to one subgoal.

The two ways of naming hypotheses generated by `destruct` can be combined to perform several destruction at once. An hypothesis `H:A\/(exists x, B x /\ C x)` can be completely decomposed by `destruct H as [a|(x,(b,c))]`. Beware that conjunction and disjunction are binary: `A/\B/\C` is understood as `A/\(B/\C)`, so the associated "naming pattern" will be of the form `(a,(b,c))`.

It is possible to let Coq choose the name of some hypothesis using `?`. Finally, the pattern _ indicates that the hypothesis should be discarded.

1.2 Combining tactics: tacticals

The expressivity of the tactic language can be considerably improved by introducing operators that combine atomic tactics in various ways. These operators are called "tacticals". Let us give a few tacticals:

- `T1 ; T2` applies `T1` to the current goal and then `T2` to all the subgoals produced by `T1`. If `T1` or one of the applications of `T2` fails, then the whole tactic `T1;T2` fails.
- `T1 || T2` applies `T1` to the current goal. If `T1` suceeds (i.e. does not produce an error), then `T2` is not used. If `T1` fails, then `T2` is applied to the current goal (any potential effect of `T1` is backtracked).
- `try T` applies `T`. If it produces an error, this error is caught and `try T` does nothing.
- `repeat T` applies repeatedly `T` as long as it suceeds.
- `do n T` is equivalent to `T;T;...;T` where the tactic is applied n times.
- `idtac` does nothing.
- `fail` always fails.

1.3 Automation

The tactic `auto` can be used to finish proofs that only require applying a small number of lemmas. The candidates are taken from a base of hints, which is formed of the hypotheses of the current goal and a set of lemmas declared by `Hint Resolve c.` It performs a depth-first-search (`auto n` performs search up to depth n). At each step it tries to apply one lemma of the base.

All the parameters of the hints must appear in the conclusion. For instance transitivity of a relation does not satisfy the criterion since it needs to know the intermediate term. This could be done by unification, but `auto` is not doing it. The tactic `eauto` overcomes this restriction.

In the prelude, the introduction rules of the main logical connectives are declared as hints. The elimination rules are not, because they would make the proof-search space too big.

Again, the proof scripts of TP1 can be made shorter by using automation.

2 Dependent types

The goal of this section is to get used to the typing rules of dependent products and sums, corresponding to universal and existential quantifiers.

First, look at the proofs of Socrates' syllogism, and try to figure out why the term has the expected type.

2.1 Equality

The basic logical rules (defined in the prelude) for equality are:

- An introduction rule: reflexivity `eq_refl : forall (A:Type)(x:A), x=x`[1]
- An elimination rule: replacing equals
  `eq_ind : forall (A:Type)(x:A)(P:A->Prop), P x -> forall (y:A), x=y -> P y`

Using those two constants *define* (not using the proof mode) terms that are proofs of the following properties of equality:

- Symmetry: `forall (A:Type)(x y:A), x=y -> y=x`
- Transitivity: `forall (A:Type)(x y z:A), x=y -> y=z -> x=z`
- Congruence: `forall (A B:Type)(f:A->B)(x y:A), x=y -> f x = f y`

What is the problem with expressing the congruence property for *dependent* functions (i.e. with `(A:Type)(B:A->Type)(f:forall x:A, B x)`) ?

Advanced user of Coq can try to overcome this issue by using `eq_rect`, a generalization of `eq_ind` to all types (and not just propositions). Proving that property is for even more advanced users!

3 Universes

Observe the types of the following expressions:

- `Prop`
- `Type` (Remember that Coq hides some information)
- `Prop->Prop`

[1] Use `@eq_refl` to desactivate implicit arguments, so `eq_refl` has the type above. Try `Check @eq_refl`. The same remark applies to many constants of the prelude.

- `forall (P:Prop), P`
- `forall (P:Type), P`

Which of those expression uses the impredicativity ?

Give a piece of development showing that `Prop` is impredicative. That is, it should type-check and should not type-check when replacing `Prop` with `Type`. (Hint: use `Definition` to fix one universe.)

4 Impredicative encodings

The sort `Prop` of Coq is impredicative, which means that `forall x:A, P x` has type `Prop` as soon as `P x` has type `Prop`, regardless of the sort of `A`.

Impredicativity can be used to represent the logical connectives and, in a partial way, datatypes.

4.1 Logical connectives

The conjunction $A \wedge B$ can be encoded by `forall Q:Prop, (A->B->Q)->Q`. Prove the introduction and elimination rules of natural deduction.

Do the same for disjunction. Remember the elimination rule of the disjunction. The encoding will capture the fact that if we have an assumption $A \vee B$ and a goal Q, then it suffices to show Q assuming A, and to show Q assuming B.

The existential quantifier is a dependent generalization of the conjunction. Here again, define the connective and show its introduction and elimination rules.

Equality can be defined using Leibniz' characterization: two objects are equal if there exists no predicate that discriminates between them. In intuitionistic logic, it is better to express this positiviely: two objects x and y are equal if for all predicate that is satisfied by x is satisfied by y.

$$\frac{}{\Gamma \vdash \texttt{refl}\ M : M{=}M} \qquad \frac{\Gamma \vdash x{=}y \quad \Gamma \vdash P\ x \quad \Gamma \vdash P{:}A{\to}\texttt{Prop}}{\Gamma \vdash P\ y}$$

Prove that the equality is reflexive, symmetric, transitive and congruent (that is $x = y$ implies $f(x) = f(y)$ for all function f).

4.2 Booleans

The encoding of booleans is `forall Q:Prop, Q->Q->Q`.

Write the booleans true and false and the the if/then/else. Then write the boolean and function, and prove that it satisfies the truth table.

Note that we cannot prove that a member of this type is either true or false.

4.3 Natural numbers

The encoding of natural numbers is `forall Q:Prop, Q->(Q->Q)->Q`. The idea is that n is going to be encoded by $\lambda x.\lambda f.f^n(x)$ (the n-th iteration of f on x).

Write the encoding for 0 and the successor function. Then write the addition and multiplication.

Adapt the negative result of previous section to natural numbers. Which Peano axioms are and are not provable ?

5 Predicative polymorphic encodings

The idea it to define the natural numbers as before, but replacing `Prop` by `Type`.

1. Define the type and its constructors (0 and successor)
2. Try to define addition as before. What happens ?

3. Prove $0 \neq 1$
4. Is it possible to prove the induction scheme, or a simpler property such as

$$\forall n \in \mathbb{N}.\, n = 0 \vee \exists m.n = S(m) \ ?$$

5. Define the subset of $\mathbb{N}$ (as a predicate of type $\mathbb{N} \to$`Prop`) of numbers that enjoy the induction scheme.
6. Show that the above subset contains 0 and is closed by successor
7. Prove the Peano axioms on this subset.