Proof Assistants – TP. 1

Bruno Barras

September 13, 2019

1 Short overview of the Gallina specification language

1.1 Main commands

- `Definition c : ty := def.`
  Extends context with symbol *c* as a short-hand for term *def* of type *ty*. The type (and :) may be omitted.
- `Definition c (x1:ty1)(x2:ty2): ty := def.`
  The same for a parameterized definition. The parameter types can be omitted (`Definition c x1 x2 := ...`).
- `Lemma c : ty.`
  Starts a proof of statement (or type) *ty*. It is followed by a sequence of commands, called tactics, that incrementally build a proof (a term) of this type. When the proof is completed, the command `Qed.` (or `Defined.`) must be used and a symbol *c* is added to the context. Its definition is the term built by the tactics.
- `Axiom c : ty.` or `Parameter c : ty.`
  Extends the context with an uninterpreted symbol *c* of the given type (or which is a proof of statement `ty`).
- `Print c.`
  Prints the definition of symbol *c*.
- `Check trm.`
  Type-checks and prints the type of the given term. A variant `Check trm : ty.` allows to check that a term has a given type.
- `Eval compute in trm.`
  Type-checks and evaluates the given term to a normal form according to all reduction rules.

When Coq starts, its context already contains some useful definitions (called the prelude). It includes propositional logic and the definitions of natural numbers and booleans.

1.2 Terms and types

In Coq, terms and types belong to the same syntactic category. The constant `Type` plays the role of the type of types.

The syntax of term is an extension of the λ-calculus.

λ-abstraction	`fun (x:ty)=> body`
application	`f arg1 arg2`
arrow type	`A -> B`
dependent product	`forall (x:A), B`

The "dependent product" is a generalization of the arrow type that we will explain later.

As for the `Definition` command, it is possible to introduce several variables at once or omit the type of the variable in the λ-abstraction or dependent product.

Coq uses notations for legibility, whose display can be controlled using `Set/Unset Printing Notations`.

1.3 Propositions (formulae)

Lemma and theorem statements are terms of a specific type `Prop`.

Actually, as we shall see later, propositions are types. The inhabitants of a proposition A (i.e. the terms of type A) are said to be the proofs of that proposition A. So, any proposition (a term of type `Prop`) has also type `Type`. In other words, `Prop` is a subtype of `Type`.

The standard connectives of first-order logic are either primitive constructions of Coq, or defined in the prelude.

connective	FOL	Coq
trivial proposition	$\top$	`True`
absurd proposition	$\bot$	`False`
negation	$\neg A$	`~A` (`not`)
conjunction	$A \wedge B$	`A /\ B` (`and`)
disjunction	$A \vee B$	`A \/ B` (`or`)
implication	$A \Rightarrow B$	`A -> B`
universal quantification	$\forall x.A$	`forall x, A`
existential quantification	$\exists x.A$	`exists x, A`

The order above gives an indication on the relative precedence of the connectives (the first one bind more tightly than the last one). Beware, for instance, that `exists x, A -> B` means `exists x, (A->B)`.

Negation is actually derived from implication and absurdity, as you can see by doing:

```
Print not.
```

1.4 Proof mode

Using the command `Lemma`, one enters an interactive proof mode allowing to build a proof term for the type of the lemma incrementaly, using tactics. As proof terms correspond to logical rules, in this mode we focus on the use of logical rules and let the system build the proof term for us. A tactic corresponds to the application of one or more logical rules to a sequent.

The proof state is a list of sequents (called goals or subgoals) that remain to be proven in order to complete the proof of the lemma statement. Initially, Coq displays one goal corresponding to the lemma statemwnt, and applying tactics may replace this goal by zero, one or more subgoals.

The proof state is displayed as

```
  n : nat
  H : n > 2
  ================
  even n

Subgoal 2 is:
 odd n
```

meaning there are 2 subgoals. In the first one, the variables and hypotheses (also called goal context) are displayed above the bar, and the conclusion (what has to be proven) appears below the bar. For the other subgoals, only the conclusion is displayed.

1.5 Tactics implementing first-order logic

The introduction and elimination rules for the standard connectives are implemented by the following tactics:

`assumption`	Axiom
`destruct H`	$\wedge$-elim, $\vee$-elim, $\bot$-elim, $\neg$-elim
`split`	$\wedge$-intro, $\top$-intro
`left`, `right`	$\vee$-intro
`intro`, `intros`	$\Rightarrow$-intro, $\neg$-intro
`apply H`	$\Rightarrow$-elim, Axiom
`destruct H`	$\exists$-elim
`exists trm`	$\exists$-intro
`intro`, `intros`	$\forall$-intro
`apply H`	$\forall$-elim

Both `destruct` and `apply` can be used with universally quantified formulae. When some universally quantified variable cannot be guessed from the context, it is possible to supply their value using the `with` keyword.

For instance, consider the goal

$$H : \forall x\, y. P(x, y) \Rightarrow Q(x) \vdash Q(t)$$

Using the tactic `apply H with (y:=u)` will produce the subgoal $P(t, u)$ (without affecting the context).

2 Propositional logic

2.1 Propositional tautologies

Using tactics, prove the following tautologies:

```
Parameter A B : Prop.
Lemma AimpA : A -> A.
...
Lemma imp_trans : (A->B)->(B->C)->A->C.
...
Lemma and_comm : A /\ B -> B /\ A.
...
Lemma or_comm : A \/ B -> B \/ A.
...
```

Using the `Print` command, print the proofs obtained for `AimpA` and `imp_trans`. What are these terms ? Does it help understanding why `->` is used both for logical implication and arrow types ?

If you have time, you can try to prove more tautologies:

- `A -> ~~A`
- `(A \/ B)/\ C -> A /\ C \/ B /\ C`
- `A <-> A`. First observe the definition of `iff` underlying the notation.

2.2 Classical logic

The theory of Coq is said "intuitionistic" (or constructive), which means it does not consider the rule of excluded-middle as valid.

The excluded-middle can be equivalently stated as `forall P:Prop, P \/ ~P` or `forall P:Prop, ~~P -> P`.

Prove that the 2 formulations are indeed equivalent.

Although the excluded-middle is not part of the logic og Coq, it is safe (i.e. it preserves consistency) to extend Coq with it. The standard library contains a module that assumes the excluded-middle and proves several consequences. Use `Require Import Classical.` to work with classical logic.

3 Working in a theory

3.1 Socrates is mortal

The goal here is to encode the famous syllogism: "Socrates is a man, all men are mortal, therefore Socrates is mortal", taking into consideration that there might exist persons that do not belong to mankind (e.g. gods and half gods).

Write the assumptions and the statement of the problem, including a type representing the persons, and predicates of persons that are men or that are mortal (those are axioms/parameters). An unary predicate on inhabitants of type A is an expression of type `A->Prop`, and `P a` is the proposition that a satisfies P.

Then prove the syllogism.

3.2 Drinker's paradox

Consider the following statement: "Consider a room with at least one person. There exists a person such that if he drinks, then everybody drinks".

Actually, this can be proven independently of the meaning of drinking. So we propose to prove: "There exists a person such that if he is immortal, then everybody is immortal". (We do not need to assume the non-emptyness because we already have a person – Socrates.)

The proof of this proposition requires the excluded-middle. You can choose either formulation of excluded-middle.

3.3 Equality

The prelude also defines the equality relation. The syntax of the formula is standard: `a = b`. The underlying constant is `eq`.

The introduction rule is reflexivity:

- The tactic `reflexivity`
- The proof-term is `eq_refl`

The elimination rule states that if $x = y$ holds, then for any predicate P, if $P(x)$ holds, then so does $P(y)$. This is the statement of the proof-term `eq_ind`. In other words, if $x = y$ holds, then it is allowed to replace x by y. It is possible to use the tactic `destruct` to apply this elimination rule. Note that when applying `destruct` it performs the opposite replacement y by x. Why ?

However, there are dedicated tactics

$$\frac{\vdash b=a}{\vdash a=b}\texttt{symmetry}$$

$$\frac{\vdash a=c \quad \vdash c=b}{\vdash a=b}\texttt{transitivity c}$$

$$\frac{H:a=b\vdash P(b)}{H:a=b\vdash P(a)}\texttt{rewrite H}$$

$$\frac{H:a=b\vdash P(a)}{H:a=b\vdash P(b)}\texttt{rewrite <- H}$$

3.4 Groups

Axiomatize the theory of a (non-commutative) group. This means introducing parameters and axioms that represent the type of the elements of the group, the basic elements (1) and group operations ($x\dot{y}$ and x^{-1}), and the equational theory (associativity, cancellation and inverse properties).

Show that $(x\dot{y})^{-1} = y^{-1}.x^{-1}$. Hint: consider the expression $y^{-1}.x^{-1}.x.y.(x.y)^{-1}$.