Proof Assistants – TP. 1

Bruno Barras

September 13, 2018

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 type of the parameters can be omitted (`Definition c x1 x2 := ...`).
- `Axiom c : ty.` or `Parameter c : ty.`
  Extends the context with an uninterpreted symbol *c* of the given type.
- `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 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.
- `Print c.`
  Prints the definition of symbol *c*.
- `Check trm.`
  Type-checks and prints the type of the given term.
- `Eval compute in trm.`
  Type-checks and evaluates the given term to a normal form according to all reduction rules.

1.2 Terms

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`

Here again, the type of the variable introduced by the λ-abstraction or dependent product can be omitted.

The constant `Type` plays the role of the type of types.

Logical formulas are typed by the sort `Prop`, which is a subtype of `Type`, i.e. every term with type `Prop` also has type `Type`.

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

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

2 Propositional and predicate logic

The standard connectives (in `Prop`) are defined as follows:

connective	type	introduction rule	elimination rule
trivial proposition	`True`	`I`	`True_ind`
absurd proposition	`False`		`False_ind`
conjunction	`A /\ B` (and)	`conj`	`and_ind`
disjunction	`A \/ B` (or)	`or_introl` `or_intror`	`or_ind`
negation	`~A` (not)		

Using the `Print` command, obtain their definitions, then write the proof terms for the following definitions:

```
Definition imp_id (A : Prop) : A -> A := ..
Definition imp_trans (A B C : Prop) : (A -> B) -> (B -> C) -> (A -> C) := ..
Definition disj_comm (A B : Prop) : (A \/ B) -> (B \/ A) := ...
```

2.1 Tactics

Using the command `Lemma` instead of `Definition`, 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 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

2.2 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.
...
```

Print the proofs obtained for `AimpA` and `imp_trans`. What are these terms ?

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.

2.3 Drinker's paradox

The prelude also contains definitions for the predicate calculus:

universal quantification	`forall x:A, B`
existential qunatification	`exists x:A, B`

and the corresponding rules:

`destruct H`	∃-elim
`exists trm`	∃-intro
`intro`, `intros`	∀-intro
`apply H`	∀-elim

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

Write the assumptions and the statement of the problem, including a type representing the persons, and a predicate of persons that drink (those are axioms/parameters). The proof of this proposition requires the excluded-middle, which can be equivalently stated as `forall P:Prop, P \/ ~P` or `forall P:Prop, ~~P -> P`. Prove that the 2 formulations are indeed equivalent, and that the drinker's paradox can be proved using excluded-middle.