CVFP (Software Design and Formal Verification)
TP 1 : Let's play with Coq

Coq is one of the most used proof assistant in the world. This tutorial aims to see how one can prove a program correct in such a tool.

CVFP's carnival Did you remember this `http://www.irisa.fr/prive/fschwarz/mit2_cvfp_2013/projects.html`? Please fill this formular with your name and group number: `http://www.framadate.org/studs.php?sondage=k6y3bdjzc8reydzm`.

Installation This should be relatively simple as there exists a lot of packages for Unix systems for it. For instance `opam` should also know about it. To install it directly, you can use its download page `http://coq.inria.fr/download`.

Graphical Interface Using Coq directly as a command-line tool can be hard and I strongly advise you to install CoqIDE or (if you're willing to use Emacs) Proof General. They should be relatively easy to install.

Exercise 1 `Basics`

You can download the file `http://people.rennes.inria.fr/Martin.Bodin/instruado/2013/CVFP/basics.v` and run it step by step.

The keyword `Inductive` is used to define types by constructors. It more or less behaves like the OCaml's `type` keyword. But in Coq there is no separation between types and objects…and the newly defined type `maybe` also has a type. You can see it using the command `Check`, while `Print` prints its definition.

The keyword `Definition` is the Coq version of the OCaml `let`. Note the use of `_` to represent a value inferable by Coq.

After defining some objects, we can prove things on them, for instance the `bool_elim` lemma. We usually use interactive mode for that, using tactics. Those tactics actually build a term: after proving a lemma, we can print the term that has been constructed as any other object using `Print`.

1.1. Try to understand how the shown tactics work.

1.2. Prove the following properties:

- `forall b : bool, exists b',~ b = b'`,
- `forall b1 b2 : bool, b = b' -> forall P : bool -> Prop, P b -> P b'`.

1.3. State an elimination scheme for the type `maybe` (in the same way than `bool_elim`) and prove it.

Exercise 2 `Adding Arithmetic`

We suggest to model a dichotomic search in a sorted array. Add to your file the content of the file `http://people.rennes.inria.fr/Martin.Bodin/instruado/2013/CVFP/arithmetic.v`. It contains a basic axiomatisation of an array.

We could programmed directly in Coq, but we'd have to deal with a restriction of it: every term has to be fully defined. There is no partial functions in Coq (that's why we had to defined the `maybe` type[1]), which yields that every function has to terminates. As we don't want to deal with termination right now, let's do it in another way.

The chosen formalisation is in small-step form: a state (containing every variable's values) is defined and a transition function is defined. For thé dichotomic search, the state is given by the value to be inserted, the array and the framing indexes of the goal value.

2.1. Complete the functions `is_final` and `next`. You can use the function `maybe_apply` for more readability[2].

Coq 's $\mathbb{Z}$ library already contains some already proven lemmae about arithmetic. You can search for lemmae using the `SearchAbout` keyword. For instance `SearchAbout Zdiv.` or `SearchAbout (_ / _).`[3] will list every lemmae about $\mathbb{Z}$ already proven in Coq 's opened libraries. There also exists a tactic `auto with zarith` trying some (but not all of them) of those lemmae on the current goal.

2.2. State and prove a simple lemma about arithmetic. For instance `forall n m p, n < p -> m < p -> (n + m) / 2 < p`.

Depending on the complexity of the formula, some elaborate tactics such as `ring` or `omega` can be useful (but probably not in this tutorial). It can also be possible to use external SMT-solvers in Coq.

2.3. Try to prove the next lemma, stating that every meaningful call to `next` will end to a defined state, or in other word that no undefined value were used.

Exercise 3 `From a real program`

Using Why3, we can actually extract a Coq formalisation of a real-world program. For instance from `http://people.rennes.inria.fr/Martin.Bodin/instruado/2013/CVFP/dichotomy.mlw`, we can extract the huge goal stored in `http://people.rennes.inria.fr/Martin.Bodin/instruado/2013/CVFP/from_dichotomy.v`.

3.1. proving such a goal would be tedious, but you can try to understand its meaning by linking every part of the formula with the annotations of the original `dichotomy.mlw` file.

[1] Which is basically a new implementation of Coq 's `option` type.

[2] Furthermore, its a monadic style can help proofs.

[3] `(_ / _)` is indeed a notation for `Zdiv`, which allows to have a more readable goal.