The COQ Proof Assistant

coq.8.14.0, 2021 Revision r1 OCamlPRO

Useful Links

Main site: `https://coq.inria.fr/`
Install: `https://github.com/coq/platform`
Sources: `https://github.com/coq/coq`

Installing `coq` with `opam`

```
opam init
eval $(opam env)
opam switch create with-coq 4.05.0
opam pin add coq ${VERSION}
opam install coqide
opam repo add coq-released https://coq.inria.fr/opam/released
opam install coq-sudoku
```

Executables

`coqc --help`	Coq compiler
`coqtop --help`	Coq toplevel
`coqide --help`	Coq IDE

Vernacular Commands

`Check` *expression*.	Check the type of expression
`Locate "_ <= _".`	Find defintion of identifier
`Compute` *expression*.	Compute an expression
`Definition` *ident*:= *exp*.	Definition
`Definition` *f args*:= *exp*.	Function definition
`Reset` *ident*.	Forget definition
`Require Import` *Library*.	Import a library
`SearchPattern` *pattern*.	Search for a pattern (type)
`Search` *patterns*.	Search a combination of patterns
`Search (_ <= _) (_ + _).`	Search example
`Print` *ident*.	Print more information on *ident*
`Fixpoint` *f args*:= *exp*.	Recursive definition requires *structural recursion*
`Theorem` *ident* : *exp*.	Theorem definition
`Lemma` *ident* : *exp*.	Lemma definition

Expressions

`True, False`	`Prop`
`1`	`nat`
`1,1`	`nat * nat`
`1=1 / textbackslash1<=2`	`Prop`
`nat -> Prop`	`Type`
`forall A: Prop, ~A`	`Prop`
`fun x : nat => x = 3`	`nat -> Prop`
`forall x:nat, x<3 \/ exists y:nat, x=y+3`	`Prop`
`let x := 1 in x+x`	`nat`
`A -> B -> C`	`A` implies (`B` implies `C`)
`if cond then e1 else e2`	Conditional
`(match e with 0 => true` `| S p => false end)`	Pattern-matching

Libraries

`Bool`	Booleans
`Arith`	Natural Numbers: `0`, `1 = S 0`
`List`	Lists: `nil`, `1::nil`, `[1;2]`, `map f l`, `l++l`
`Omega`	Provide tactic `omega`
`ArithRing`	Provide tactic `ring`
`ZArith`	Integer Numbers

Tactics Table

	Hypothesis `H`	Conclusion
⇒	`apply H [with x:=E]`	`intros H`
¬	`elim H` `case H`	`intros H`
`False`	`elim H` `case H`	`intros H`
∀	`apply H`	`intros H`

∃	`elim H`
	`case H`
	`destruct H as [x H1]`
		`exists` *v*

∧	`elim H`
	`case H`
	`destruct H as [H1 H2]`
		`split`

∨	`elim H`
	`case H`
	`destruct H as [H1 | H2]`
		`left`
		`right`

`=`	`rewrite H [with x:=E]` `rewrite <- H [with x:=E]`	`reflexivity` `ring`

New Datatypes

```
Inductive bin : Type :=
  L : bin
| N : bin -> bin -> bin.
```

Tactics

`assumption`	Search goal in assumptions
`intuition, tauto`	Auto use of conj, disj and neg
`firstorder`	`tauto` with exist and forall
`Qed`	Proof finished, to be saved
`t1;t2`	Use `t1`, then `t2`
`apply le_trans with (m:=1)`	Apply, with free-var subst
`unfold` *ident*	Substitute with definition
`assert (H : P)`	introduce hypoth. `H` as `P`
`auto, eauto`	Use user-provided theorems
`ring`	equalities, add and mult
`omega`	linear inequations
`induction n`	induction on a natural number
`simpl`	substitute recursive call
`discriminate`	remove self-contradictory goals
`injection H`	instantiate `H` before goal
`rewrite H`	Use left-to-right rewrite of equality
`rewrite <- H`	Use right-to-left rewrite of equality
`case H`	Split between `H` and `~H`
`exact H`	?
`inversion H`	?

```
Theroem t1: forall x1 ... xk, A1 x1 ... xk
-> A2 x1 ... xk -> ... -> C x1 ... xk
================
C a1 ... ak
```
use: `apply t1`
```
================
A1 a1 ... ak, ..., An a1 ... ak
```

AAA **AAA** **AAA**