Coq cheat sheet

Notation

Propositions	**Coq**
$\top, \bot$	`True`, `False`
$p \wedge q$	`p /\ q`
$p \Rightarrow q$	`p -> q`
$p \vee q$	`p \/ q`
$\neg p$	`~ p`
$\forall x \in A \,.\, p(x)$	`forall x:A, p x`
$\forall x, y \in A \,.\, \forall u, v \in B \,.\, q$	`forall (x y:A) (u v:B), q`
$\exists x \in A \,.\, p(x)$	`exists x:A, p x`

Sets	**Coq**
1	`unit`
$A \times B$	`prod A B` or `A * B`
$A + B$	`sum A B` or `A + B`
B^A or $A \to B$	`A -> B`
$\{x \in A \mid p(x)\}$	`{x:A | p x}`
$\sum_{x \in A} B(x)$	`{x:A & B x}` or `sig A B`
$\prod_{x \in A} B(x)$	`forall x:A, B x`

Elements	**Coq**
$\star \in 1$	`tt : unit`
$x \mapsto f(x)$ or $\lambda x \in A \,.\, f(x)$	`fun (x : A) => f x`
$\lambda x, y \in A \,.\, \lambda u, v \in B \,.\, f(x)$	`fun (x y : A) (u v : B) => f x`
$(a, b) \in A \times B$	`(a,b) : A * B`
$\pi_1(t)$ where $t \in A \times B$	`fst t`
$\pi_2(t)$ where $t \in A \times B$	`snd t`
$\pi_1(t)$ where $t \in \sum_{x \in A} B(x)$	`projT1 t`
$\pi_2(t)$ where $t \in \sum_{x \in A} B(x)$	`projT2 t`
$\iota_1(t) \in A + B$ where $t \in A$	`inl t`
$\iota_2(t) \in A + B$ where $t \in B$	`inr t`
$t \in \{x \in A \mid p(x)\}$ because ρ	`exist t` ρ
$\iota(t)$ where $\iota : \{x \in A \mid p(x)\} \hookrightarrow A$	`projT1 t`

Basic tactics

When the goal is ...	**... use tactic**
very simple	`auto`, `tauto` or `firstorder`
`p /\ q`	`split`
`p \/ q`	`left` or `right`
`p -> q`	`intro`
`~p`	`intro`
`p <-> q`	`split`
an assumption	`assumption`
`forall x, p`	`intro`
`exists x, p`	`exists` t

To use hypothesis H **...**	**... use tactic**
`p \/ q`	`destruct` H `as` $[H_1 \mid H_2]$
`p /\ q`	`destruct` H `as` $[H_1\ H_2]$
`p -> q`	`apply` H
`p <-> q`	`apply` H
`~p`	`apply` H or `elim` H
`False`	`contradiction`
`forall x, p`	`apply` H
`exists x, p`	`destruct` H `as` $[x\ G]$
`a = b`	`rewrite` H or `rewrite <-` H

If you want to ...	**... then use**
prove by contradiction $p \wedge \neg p$	`absurd` p
simplify expressions	`simpl`
prove via intermediate goal p	`cut` p
prove by induction on t	`induction t`
pretend you are done	`admit`
import package P	`Require Import` P
compute t	`Eval compute in` t
print definition of p	`Print` p
check the type of t	`Check` t
search theorems about p	`SearchAbout` p

Inductive definitions

Inductive definition of X

```
Inductive X args :=
  | constructor1 : args1 -> X
  | constructor2 : args2 -> X
  ...
  | constructorN : argsN -> X.
```

Coq generates induction and recursion principles `X_ind`, `X_rec`, `X_rect`.

Construction of an object by cases

```
match t with
  | case1: result1
  | case2: result2
  ...
  | caseN: resultN
end
```

Recursive definition of f

```
Fixpoint f args := ...
```