WTF IS A COQ INTRODUCTION / ELIMINATION RULE?

CHT

Coq essentially follows intuitionistic logic. Writing a proof in coq is thus constructing a proof tree in intuitionistic logic from the conclusion to the axioms, i.e. *from the ground up, and not from the axioms to the conclusion*!

1. Conjunction

1.1. Introduction. The rule of intuitionistic logic goes as follows:

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \wedge\texttt{-intro}$$

The corresponding tactic in coq is `split`.

```
1 Lemma conj : forall P Q: Prop, P -> Q -> P /\ Q.
2 Proof.
3   intros P Q p q.
4   split.
5   assumption.
6   assumption.
7 Defined.
```

The proof state at line 3 is:

```
1 subgoal
P, Q : Prop
p : P
q : Q
______________________________________(1/1)
P /\ Q
```

whereas at line 4 it becomes:

```
2 subgoals
P, Q : Prop
p : P
q : Q
______________________________________(1/2)
P
______________________________________(2/2)
Q
```

Note that this lemma `conj` is already implemented in coq:

```
Lemma conj' : forall P Q: Prop, P -> Q -> P /\ Q.
Proof.
  intros P Q.
  apply conj.
Defined.
```

Also, note that tactic `split` is equivalent to `intros; apply conj`.

1.2. Elimination. The $\wedge$ connective has two elimination rules, one for the left and one for the right. We discuss only the former, the latter being similar.

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge\texttt{-elim-left}$$

The corresponding theorem is `proj1` (`proj2` for right elimination):

```
proj1: forall A B : Prop, A /\ B -> A
```

1.3. Induction. Induction for $\wedge$ corresponds to the following axiom:

$$\mathsf{ind}_\wedge : \prod_{A,B,P:\mathfrak{U}} ((A \to B \to P) \to (A \wedge B \to P))$$

which is coq is

```
and_ind =
fun A B P : Prop => and_rect (A:=A) (B:=B) (P:=P)
     : forall A B P : Prop, (A -> B -> P) -> A /\ B -> P
```

2. Disjunction

2.1. Introduction. The $\vee$ connective has two introduction rules, one for the left and one for the right. We discuss only the former, the latter being similar.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee\texttt{-intro-left}$$

In coq, `or` is an inductive type:

```
Result for command Print or . :
Inductive or (A B : Prop) : Prop :=
    or_introl : A -> A \/ B | or_intror : B -> A \/ B
```

and the left (resp. right) introduction rules correspond to `or_introl` (resp. `or_intror`). Note that coq tactic `left` (resp. `right`) correspond to the sequence `intros ; apply or_introl` (resp. `intros; apply or_intror`). For example:

```
Lemma disj_4_3 : forall A B C D: Prop, C -> A \/ B \/ C \/ D.
Proof.
 right.
 right.
 left.
 assumption.
Defined.
```

at the highlighted line, the proof state is

```
1 subgoal
A, B, C, D : Prop
H : C
______________________________________(1/1)
B \/ C \/ D
```

2.2. **Elimination.** This is where it gets confusing. Applying elimination rules removed a given connective on the proof tree, so when writing the proof in coq, it should make it appear since we write proofs in reverse. BUT if that connective is in the assumption, then it effectively makes it disappear:

```
1 Lemma disj_comm : forall A B: Prop, A \/ B -> B \/ A.
2 Proof.
3   intros A B d.
4   elim d.
5   right ; assumption.
6   left ; assumption.
7 Defined.
```

The proof state at line 3 is:

```
1 subgoal
A, B : Prop
d : A \/ B
______________________________________(1/1)
B \/ A
```

and at line 4:

```
2 subgoals
A, B : Prop
d : A \/ B
______________________________________(1/2)
A -> B \/ A
______________________________________(2/2)
B -> B \/ A
```

2.3. **Induction.**

$$\mathsf{ind}_\vee : \prod_{A,B,P:\mathfrak{U}} ((A \to P) \to (B \to P) \to (A \vee B \to P)).$$

```
Result for command Print or_ind . :
or_ind =
fun (A B P : Prop) (f : A -> P) (f0 : B -> P) (o : A \/ B) =>
match o with
| or_introl x => f x
| or_intror x => f0 x
end
     : forall A B P : Prop, (A -> P) -> (B -> P) -> A \/ B -> P
```

3. Implication

3.1. **Introduction.**

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \to Q} \to\texttt{-intro}$$

This corresponds exactly to tactic `intro`!

3.2. **Elimination (modus ponens).**

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash A \to B}{\Gamma \vdash B} \to\texttt{-elim}$$

There is two ways to proceed:

```
Lemma modus_ponens : forall A B: Prop, A -> (A -> B) -> B.
Proof.
  intros A B a f.
  exact (f a).
Defined.
```

or use intermediate lemmas:

```
Lemma id : forall A: Prop, A -> A.
Proof.
  intros; assumption.
Defined.

Lemma impl_perm : forall A B P: Prop, (A -> B -> P) -> (B -> A -> P).
Proof.
  intros A B P f b a; exact (f a b).
Defined.

Lemma modus_ponens' : forall A B: Prop, A -> (A -> B) -> B.
Proof.
  intros A B.
  exact (impl_perm (A -> B) A B (id (A -> B))).
Defined.
```

3.3. **Induction.**

4. False

4.1. **Introduction.**

4.2. **Elimination.**

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A} \bot\texttt{-elim}$$

```
Lemma false_ind : forall A: Prop, False -> A.
Proof.
  intros A f.
  elim f.
Defined.
```

4.3. Induction.

```
Result for command Print False_ind . :
False_ind = fun P : Prop => False_rect P
     : forall P : Prop, False -> P
```

```
2 subgoals
A, B : Prop
d : A \/ B
______________________________________(1/2)
A -> B \/ A
______________________________________(2/2)
B -> B \/ A
```

5. Negation

The ¬ operator is *defined* as:

$$\frac{\Gamma \vdash A \to \bot}{\Gamma \vdash \neg A}$$

In coq, the elimination corresponds to tactic `unfold not`:

```
1 Lemma absurd : forall A: Prop, A -> ~A ->
  ↪ False.
2 Proof.
3   intro A.
4   unfold not.
5   apply modus_ponens.
6 Defined.
```

The proof state at line 3 is:

```
1 subgoal
A : Prop
______________________________________(1/1)
A -> ~ A -> False
```

whereas at line 4 it is:

```
1 subgoal
A : Prop
______________________________________(1/1)
A -> (A -> False) -> False
```

6. Universal quantification

6.1. Elimination.

In coq, `forall` really acts like a function, and this can be eliminated with `intro`.

7. Existential quantification

7.1. Introduction.

$$\frac{\Gamma, a : A \vdash Pa}{\Gamma \vdash \exists x : A, Px}\ \exists\text{-intro}$$

```
1 Lemma forall_exists : forall (A: Set) (P: A ->
  ↪ Prop), A -> (forall x: A, P x) -> (exists
  ↪ x: A, P x).
2 Proof.
3   intros A P a p.
4   exists a.
5   exact (p a).
6 Defined.
```

Before line 4:

```
1 subgoal
A : Set
P : A -> Prop
a : A
p : forall x : A, P x
______________________________________(1/1)
exists x : A, P x
```

at line 4:

```
1 subgoal
A : Set
P : A -> Prop
a : A
p : forall x : A, P x
______________________________________(1/1)
P a
```

7.2. Elimination.

$$\frac{\Gamma, (\exists x : A, Px) \vdash Q}{\Gamma \vdash (\forall x : A, Px \to Q)}\ \exists\text{-elim}$$

```
1 Lemma not_forall : forall (A: Set) (P: A ->
  ↪ Prop), (forall x: A, P x) -> ~(exists x:
  ↪ A, ~P x).
2 Proof.
3   unfold not.
4   intros A P p q.
5   elim q.
6   intros a r.
7   exact (r (p a)).
8 Defined.
```

Proof state at line 4:

```
1 subgoal
A : Set
P : A -> Prop
p : forall x : A, P x
q : exists x : A, P x -> False
______________________________________(1/1)
False
```

at line 5:

```
1 subgoal
A : Set
P : A -> Prop
p : forall x : A, P x
q : exists x : A, P x -> False
______________________________________(1/1)
forall x : A, (P x -> False) -> False
```

at line 6:

```
1 subgoal
A : Set
P : A -> Prop
p : forall x : A, P x
q : exists x : A, P x -> False
a : A
r : P a -> False
______________________________________(1/1)
False
```