Coq: What is it doing, and the object-meta swap

CS3234
Martin Henz and Aquinas Hobor
Proofs

• We’ve now shown you two (related) ways of drawing proofs: trees and 3-column format

• Actually, the 3-column format is encoding a tree (really a directed acyclic graph, i.e., you can reuse lines without copying them out multiple times)
A proof (3 column)

• Consider this very simple proof:

1. $P \land Q$ (assumption)
2. $P$ ($\land e1, 1$)
3. $Q$ ($\land e2, 1$)
4. $Q \land P$ ($\land i, 3, 2$)
5. $(P \land Q) \rightarrow (Q \land P)$ ($\rightarrow i, 1\text{—}5$)
A proof (tree)

- The same proof in tree form:

```
1. \[ P \land Q \quad \text{assumption} \]
2. \[ Q \quad \land e_2 \]
3. \[ P \quad \land e_1 \]
4. \[ Q \land P \quad \land i \]
```

```
(P \land Q) \rightarrow (Q \land P) \quad \rightarrow i
```

\[ P \land Q \]
DAG

- The 3-column did not have this duplication:

\[
\begin{align*}
P \land Q & \quad \text{assumption} \\
Q & \quad \land e_2 \\
Q \land P & \\
\end{align*}
\]

\[
\begin{align*}
P \land Q & \quad \text{assumption} \\
P & \quad \land e_1 \\
Q \land P & \land i \\
( P \land Q ) \rightarrow ( Q \land P ) & \rightarrow i \\
( P \land Q ) & \rightarrow ( Q \land P ) \\
\end{align*}
\]
• In some sense the structure looks like this:
• Or more accurately like this:
Coq

- Coq is helping you build these trees/DAGs starting from the “bottom up”:

\[(P \land Q) \Rightarrow (Q \land P)\]
What it actually looks like

1 subgoal
v : Valuation
P : PropFormula
Q : PropFormula

\[ \text{holds } v \left( \text{propImpl} \left( \text{propConj} P Q \right) \left( \text{propConj} Q P \right) \right) \]

Now, we want to “apply Impl_I”
What it actually looks like

1 subgoal
v : Valuation
P : PropFormula
Q : PropFormula

holds v (propImpl (propConj P Q) (propConj Q P))

Now, we “apply Impl_I” (then intro)
What it actually looks like

1 subgoal
v : Valuation
P : PropFormula
Q : PropFormula
H : holds v (propConj P Q)

holds v (propConj Q P)

What happened to the tree?
Coq

\[
(P \land Q) \rightarrow (Q \land P)
\]
What it actually looks like

Now, we want to “apply Conj_I”

2 subgoals
v : Valuation
P : PropFormula
Q : PropFormula
H : holds v (propConj P Q)
___________________________ (1/2)
holds v Q

___________________________ (2/2)
holds v P
\((P \land Q) \rightarrow (Q \land P)\)
What it actually looks like

Now, we “apply Conj_E2 with P”

2 subgoals
v : Valuation
P : PropFormula
Q : PropFormula
H : holds v (propConj P Q)

(1/2)
holds v (propConj P Q)

(2/2)
holds v P
What it actually looks like

Now, we “apply Conj_E2 with P”

The old goal was “holds v Q”

Can’t guess P from the goal
What it actually looks like

Now, we “apply Conj_E2 with P”

2 subgoals
v : Valuation
P : PropFormula
Q : PropFormula
H : holds v (propConj P Q)
____________________________________________________________________(1/2)
holds v (propConj P Q)

____________________________________________________________________(2/2)
holds v P
Coq

\[ P \land Q \rightarrow (Q \land P) \]
What it actually looks like

Now, we “assumption”

1 subgoals
v : Valuation
P : PropFormula
Q : PropFormula
H : holds v (propConj P Q)

_________________________________________________________________________(1/1)
holds v P
\[ (P \land Q) \rightarrow (Q \land P) \]
What it actually looks like

Now, we “apply ConjE1 with Q” and “trivial”

Proof completed.
DAG

• More-or-less like this:
That’s how proofs work in Coq

- The details can get messy, but more-or-less that’s how Coq is working.

- Now we want to talk about something else you may have wondered about.
What it actually looks like

1 subgoal

v : Valuation
P : PropFormula
Q : PropFormula

_________________________________________________________________________(1/1)
holds v (propImpl (propConj P Q) (propConj Q P))

Now, we “apply Impl_I” (then intro)

Why do we need to do this?
We’ve been cheating

• We wanted to emphasize that formulas were just “data structures” – so we built them explicitly as (inductively defined) data:

Inductive PropFormula : Type :=
  | propTop : PropFormula
  | propBot : PropFormula
  | propAtom : Atom -> PropFormula
  | propNeg : PropFormula -> PropFormula
  | propConj : PropFormula -> PropFormula -> PropFormula
  | propDisj : PropFormula -> PropFormula -> PropFormula -> PropFormula
  | propImpl : PropFormula -> PropFormula -> PropFormula.
Built-in operators

• But Coq has all of these already built-in, and so we had to use some of the built-in operators from time-to-time.

• You’ve even seen some of them! For example,
  – \( \rightarrow \) instead of \texttt{propImpl}. Coq also has:
    – \( \&\& \) instead of \texttt{propConj}
    – \( \lor \) instead of \texttt{propDisj}
    – \( \neg \) instead of \texttt{propNeg}
    – \texttt{True}/\texttt{False} for \texttt{propTop}/\texttt{propBot}
Since it’s built in, it’s much shorter:

Lemma Conj_Comm:
    forall v P Q,
    holds v (propImpl
       (propConj P Q)
       (propConj Q P)).
Proof.
    intros.
    apply Impl_I. intro.
    apply Conj_I.
    apply Conj_E2 with P.
    trivial.
    apply Conj_E1 with Q.
    trivial.
    Qed.

Lemma Conj_Comm2:
    forall P Q,
    P \ Q \rightarrow Q \ P.
Proof.
    intros.
    destruct H.
    split.
    trivial.
    trivial.
    Qed.
Since it’s built in, it’s much shorter:

Lemma Conj_Comm:
  \[\forall v \ P \ Q,\]
  holds \( v \ (\text{propImpl} (\text{propConj} P Q) \ (\text{propConj} Q P)).\]
Proof.
intros.
apply Impl_I. intro.
apply Conj_I.
apply Conj_E2 with P.
trivial.
apply Conj_E1 with Q.
trivial.
Qed.

Lemma Conj_Comm2:
  \[\forall P \ Q,\]
  \( P \land Q \rightarrow Q \land P.\)
Proof.
intros.
destruct H.
split.
  trivial.
trivial.
Qed.
Since it’s built in, it’s much shorter:

Lemma Conj_Comm:
  forall v P Q,
  holds v (propImpl
    (propConj P Q)
    (propConj Q P)).
Proof.
intros.
apply Impl_I. intro.
apply Conj_I.
apply Conj_E2 with P.
trivial.
apply Conj_E1 with Q.
trivial.
Qed.

Lemma Conj_Comm2:
  forall P Q,
  P \land Q \rightarrow Q \land P.
Proof.
intros.
destruct H.
split.
trivial.
trivial.
Qed.
Since it’s built in, it’s much shorter:

Lemma Conj_Comm:  
  \[ \forall v. (\text{propImpl}(\text{propConj} P Q) \land \text{propConj} Q P). \]
  Proof. 
  intros. 
  apply Impl_I. intro. 
  apply Conj_I. 
  apply Conj_E2 with P. 
  trivial. 
  apply Conj_E1 with Q. 
  trivial. 
  Qed.

Lemma Conj_Comm2:  
  \[ \forall P Q. P \land Q \rightarrow Q \land P. \]
  Proof. 
  intros. 
  destruct H. 
  split. 
  trivial. 
  trivial. 
  Qed.
Or even (the wonders of automation)

Lemma Conj_Comm3:
  \( \forall P \forall Q, \)
  \( P \land Q \rightarrow Q \land P \).

Proof.
  intros.
  destruct H.
  split; trivial.
Qed.
Or even (the wonders of automation)

Lemma Conj_Comm3: 
for all \( P, Q, \)
\[ P \land Q \rightarrow Q \land P. \]
Proof.
intros.
destruct H.
split; trivial.
Qed.

Lemma Conj_Comm4: 
for all \( P, Q, \)
\[ P \land Q \rightarrow Q \land P. \]
Proof.
tauto.
Qed.
A quick tactic list (not complete!)

→i intro (intros)
→e apply (apply in, generalize, spec)
∧i split
∧e12 destruct
∨i1 left
∨i2 right
∨e destruct
Going “meta”

• From now on (not counting this week’s Coq HW or next week’s quiz), we will always present the Coq using the built-in operators

• This should save you considerable time and let you write shorter proofs...

• The real advantage comes when we move to predicate logic
Tactics for predicate logic

- Actually, you have already seen some of these... and they are not very hard.

\[ \forall i \text{ intros} \]
\[ \forall e \text{ spec, generalize (apply...)} \]
\[ \exists i \text{ exists} \]
\[ \exists e \text{ destruct} \]
Coq HW (not until next week)

• We will give some homework next week that will involve these tactics for your practice

• Don’t Panic: The quiz next week will not cover them (except intros which you have seen lots of), so it’s just a “sneak peek” now.

• Often it is much easier to do predicate logic in the theorem prove than on paper.
The horrible side of predicate logic

• You may recall the (horrible) issues of variable hiding and variable capture.

• *Almost guaranteed* to be on midterm & final.

• The great thing is Coq does all of this for you.
Demo

• We have two lemmas that demonstrate some of the issues involved in variable scoping.
Questions?