Natural Deduction in Coq

(Based on [Pfenning 2004], [Bertot and Castéran 2004], and [Coq Reference Manual])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

Overview

- (Intuitionistic) Natural Deduction
- Proof Terms
- Typing/Inference Rules in Coq
- Inductive Definitions in Coq
- Introduction and Elimination in Coq

Natural Deduction in the Sequent Form

$$\overline{\Gamma, A \vdash A} \; (Hyp)$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \; (\wedge I)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \; (\wedge E_1)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \; (\wedge E_2)$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \; (\vee I_1)$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \; (\vee I_2)$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \; (\vee E)$$

Note: I: Introduction; E: Elimination.

Natural Deduction (cont.)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \, (\to I) \qquad\qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \, (\to E)$$

$$\frac{\Gamma, A \vdash B \land \neg B}{\Gamma \vdash \neg A} \, (\neg I) \qquad\qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash B} \, (\neg E)$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash \neg\neg A} \, (\neg\neg I) \qquad\qquad \frac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A} \, (\neg\neg E)$$

Intuitionistic Natural Deduction

$$\overline{A_1, \cdots, A_i, \cdots, A_n \vdash A_i} \, (Hyp^i)$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \, (\wedge I) \qquad\qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \, (\wedge E_1)$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \, (\wedge E_2)$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \, (\vee I_1)$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \, (\vee I_2) \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \, (\vee E)$$

Intuitionistic Natural Deduction (cont.)

$$\frac{}{\Gamma \vdash \top} \, (\top I)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \, (\to I) \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B} \, (\to E)$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A} \, (\bot E)$$

Note:

🔵 The last rule says that, if we can deduce $\bot$ (representing a contradiction), then we can deduce anything.

🔵 $\neg A$ is then represented (i.e., defined) as $A \to \bot$.

🔵 With the $\to$-elimination rule, one can deduce a contradiction (and hence anything) from $\neg A$ and A.

Quantifier Rules

$$\frac{\Gamma \vdash B[y/x]}{\Gamma \vdash \forall x B} \, (\forall I) \qquad\qquad \frac{\Gamma \vdash \forall x B}{\Gamma \vdash B[t/x]} \, (\forall E)$$

$$\frac{\Gamma \vdash B[t/x]}{\Gamma \vdash \exists x B} \, (\exists I) \qquad\qquad \frac{\Gamma \vdash \exists x B \qquad \Gamma, B[y/x] \vdash C}{\Gamma \vdash C} \, (\exists E)$$

Note: In the quantifier rules above, we assume that all substitutions are admissible and y does not occur free in Γ or B.

Equality Rules

Let t, t_1, t_2 be arbitrary terms and again assume all substitutions are admissible.

$$\frac{}{\Gamma \vdash t = t}\,(=I) \qquad \frac{\Gamma \vdash t_1 = t_2 \quad \Gamma \vdash A[t_1/x]}{\Gamma \vdash A[t_2/x]}\,(=E)$$

Note: The $=$ sign is part of the object language, not a meta symbol.

Intuitionistic ND with Proof Terms

$$\overline{x_1 : A_1, \cdots, x_i : A_i, \cdots, x_n : A_n \vdash x_i : A_i} \; (Hyp^i)$$

$$\frac{\Gamma \vdash t_1 : A \quad \Gamma \vdash t_2 : B}{\Gamma \vdash \mathbf{pair}(t_1, t_2) : A \wedge B} \; (\wedge I)$$

$$\frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \mathbf{fst}(t) : A} \; (\wedge E_1)$$

$$\frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash \mathbf{snd}(t) : B} \; (\wedge E_2)$$

Intuitionistic ND with Proof Terms (cont.)

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathbf{inl}(B, t) : A \vee B} \ (\vee I_1)$$

$$\frac{\Gamma \vdash t : B}{\Gamma \vdash \mathbf{inr}(A, t) : A \vee B} \ (\vee I_2)$$

$$\frac{\Gamma \vdash t : A \vee B \qquad \Gamma, x : A \vdash t_1 : C \qquad \Gamma, y : B \vdash t_2 : C}{\Gamma \vdash \mathbf{case}(t, x.t_1, y.t_2) : C} \ (\vee E)$$

Note: $\mathbf{case}(\mathbf{inl}(B, t), x.t_1, y.t_2) \stackrel{\Delta}{=} t_1[t/x];$
$\mathbf{case}(\mathbf{inr}(A, t), x.t_1, y.t_2) \stackrel{\Delta}{=} t_2[t/y].$

Intuitionistic ND with Proof Terms (cont.)

$$\frac{}{\Gamma \vdash \mathbf{unit} : \top} \, (\top I)$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \mathbf{fun}(x : A) \Rightarrow t : A \to B} \, (\to I)$$

$$\frac{\Gamma \vdash t_1 : A \to B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 \ t_2 : B} \, (\to E)$$

Note: In the $(\to I)$ rule above, it is assumed that B does not contain x (so B is independent of A).

$$\frac{\Gamma \vdash t : \bot}{\Gamma \vdash \mathbf{abort}(A, t) : A} \, (\bot E)$$

An Example Proof

α:

$$\dfrac{\dfrac{}{x : (A \vee B) \to C, y : A \vdash y : A} \ (Hyp^2)}{x : (A \vee B) \to C, y : A \vdash \mathrm{inl}(B, y) : A \vee B} \ (\vee I)$$

$$\dfrac{\dfrac{\dfrac{}{x : (A \vee B) \to C, y : A \vdash x : (A \vee B) \to C} \ (Hyp^1) \qquad \alpha}{x : (A \vee B) \to C, y : A \vdash x \ \mathrm{inl}(B, y) : C} \ (\to E)}{\dfrac{x : (A \vee B) \to C \vdash \mathbf{fun}(y : A) \Rightarrow x \ \mathrm{inl}(B, y) : A \to C}{\vdash \mathbf{fun}(x : (A \vee B) \to C) \Rightarrow \mathbf{fun}(y : A) \Rightarrow x \ \mathrm{inl}(B, y) : ((A \vee B) \to C) \to (A \to C)} \ (\to I)} \ (\to I)$$

Intuitionistic ND with Proof Terms (cont.)

$$\frac{\Gamma, y : A \vdash t : B[y/x]}{\Gamma \vdash \mathbf{fun}(x : A) \Rightarrow t : \forall x : A, B} \ (\forall I)$$

$$\frac{\Gamma \vdash t_1 : \forall x : A, B \qquad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 \ t_2 : B[t_2/x]} \ (\forall E)$$

$$\frac{\Gamma \vdash t_1 : A \qquad \Gamma \vdash t_2 : B[t_1/x]}{\Gamma \vdash \mathbf{pair}(t_1, t_2) : \exists x : A, B} \ (\exists I)$$

$$\frac{\Gamma \vdash t' : \exists x : A, B \qquad \Gamma, y : A, z : B[y/x] \vdash t : C}{\Gamma \vdash \mathbf{open}(t', y.z.t) : C} \ (\exists E)$$

Note: All substitutions are admissible and y does not occur free in Γ or B; $\mathbf{open}(\mathbf{pair}(t_1, t_2), y.z.t) \stackrel{\Delta}{=} t[t_2/z][t_1/y]$.

Curry-Howard Correspondence

Logic	Programming (Typed λ-Calculus)
proof	program (λ term)
proposition	type
proof checking	type checking

Typing/Inference Rules in Coq/pCIC

- A type is expressed as a term.
- Every term has a type.
- The type of a type is called a *sort*.
- Sorts in predicative Calculus of Inductive Constructions (pCIC, the type theory behind Coq):

$$\mathcal{S} = \{\mathrm{Prop}, \mathrm{Set}, \mathrm{Type}(0), \mathrm{Type}(1), \mathrm{Type}(2), \cdots\}$$

  - $\mathrm{Prop} : \mathrm{Type}(i)$, for every i.
  - $\mathrm{Set} : \mathrm{Type}(i)$, for every i.
  - $\mathrm{Type}(i) : \mathrm{Type}(j)$, for $i < j$.

Typing/Inference Rules in Coq (cont.)

Prod$(\mathcal{S}, \mathrm{Prop}, \mathrm{Prop})$

$$\frac{E, \Gamma \vdash A : s \qquad E, \Gamma :: (a : A) \vdash B : \mathrm{Prop}}{E, \Gamma \vdash \forall a : A, B : \mathrm{Prop}}$$

where $s \in \mathcal{S}$.

Prod$(\{\mathrm{Prop}, \mathrm{Set}\}, \mathrm{Set}, \mathrm{Set})$ /* Set is predicative in pCIC. */

$$\frac{E, \Gamma \vdash A : s \qquad E, \Gamma :: (a : A) \vdash B : \mathrm{Set}}{E, \Gamma \vdash \forall a : A, B : \mathrm{Set}}$$

where $s \in \{\mathrm{Prop}, \mathrm{Set}\}$.

Prod$(\mathrm{Type}, \mathrm{Type}, \mathrm{Type})$

$$\frac{E, \Gamma \vdash A : \mathrm{Type}(i) \qquad E, \Gamma :: (a : A) \vdash B : \mathrm{Type}(j)}{E, \Gamma \vdash \forall a : A, B : \mathrm{Type}(k)}$$

where $i \leq k$ and $j \leq k$.

Typing/Inference Rules in Coq (cont.)

$$\textbf{Lam} \; \frac{E, \Gamma \vdash A \to B : s \qquad E, \Gamma :: (v : A) \vdash e : B}{E, \Gamma \vdash \text{fun } v : A \Rightarrow e : A \to B}$$

Note: B does not depend on v.

$$\textbf{App} \; \frac{E, \Gamma \vdash e_1 : A \to B \qquad E, \Gamma \vdash e_2 : A}{\Gamma \vdash e_1 \; e_2 : B}$$

$$\textbf{Lam} \; \frac{E, \Gamma \vdash \forall v : A, B : s \qquad E, \Gamma :: (v : A) \vdash t : B}{E, \Gamma \vdash \text{fun } v : A \Rightarrow t : \forall v : A, B}$$

Note: "$A \to B$" is just a shorthand for "$\forall v : A, B$", when B does not depend on v.

$$\textbf{App} \; \frac{E, \Gamma \vdash t_1 : \forall v : A, B \qquad E, \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 \; t_2 : B[t_2/v]}$$

The Example Proof in Coq

- The proposition to prove: $((A \lor B) \to C) \to (A \to C)$.
  Formalization in Coq:

  ```
  Variables A B C: Prop.
  Lemma t0: ((A \/ B) -> C) -> (A -> C).
  ```

- A proof in Coq:

  ```
  intro x; intro y; apply x; left; assumption.
  ```

- The proof term in Coq:

  ```
  fun (x : A \/ B -> C) (y : A) => x (or_introl B y)
      : (A \/ B -> C) -> A -> C
  ```

  cf.: $\mathbf{fun}(x : (A \lor B) \to C) \Rightarrow \mathbf{fun}(y : A) \Rightarrow x \, \mathrm{inl}(B, y)$
  $: ((A \lor B) \to C) \to (A \to C)$

∧-Introduction in Coq

- Definitions for ∧:

  ```
  Inductive and (A : Prop) (B : Prop) : Prop :=
    conj : A -> B -> A /\ B
  ```

  Note: "A /\ B" is a convenient notation for "and A B".

- Proposition to prove:

  ```
  Lemma and_intro: A -> B -> (A /\ B).
  ```

- A proof in Coq:

  ```
  intro t1; intro t2; split; assumption; assumption.
  ```

- The proof term in Coq:

  ```
  and_intro = fun (t1 : A) (t2 : B) => conj t1 t2
      : A -> B -> A /\ B
  ```

∧-**Elimination in Coq**

🔵 Definitions for ∧:
```
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
```

🔵 Proposition to prove:

Lemma and_elim: (A /\ B) -> A.

🔵 A proof in Coq:

intro t; elim t; intro x; intro y; assumption.

🔵 The proof term in Coq:
```
fun t : A /\ B => and_ind (fun (x : A) (_ : B) => x) t
    : A /\ B -> A
```

∧-Elimination in Coq (cont.)

α:

$$
\textbf{App} \cfrac{
\textbf{App} \cfrac{
\textbf{App} \cfrac{
\cfrac{\text{(Let } a \text{ denote and\_ind)}}{\cdot \vdash a : \forall A, B, P : \text{Prop}, (A \to B \to P) \to A \wedge B \to P} \qquad \cfrac{}{\cdot \vdash A : \text{Prop}}
}{\cdot \vdash a\ A : \forall B, P : \text{Prop}, (A \to B \to P) \to A \wedge B \to P} \qquad \cfrac{}{\cdot \vdash B : \text{Prop}}
}{\cdot \vdash a\ A\ B : \forall P : \text{Prop}, (A \to B \to P) \to A \wedge B \to P} \qquad \cfrac{}{\cdot \vdash A : \text{Prop}}
}{t : A \wedge B \vdash a\ A\ B\ A : (A \to B \to A) \to A \wedge B \to A}
$$

$$
\textbf{App} \cfrac{
\textbf{App} \cfrac{
\alpha \qquad \textbf{App} \cfrac{
\textbf{Lam} \cfrac{
\textbf{Lam} \cfrac{
\cfrac{}{t : A \wedge B, x : A, y : B \vdash x : A}
}{t : A \wedge B, x : A \vdash \text{fun}(y : B) \Rightarrow x : B \to A}
}{t : A \wedge B \vdash \text{fun}(x : A)(y : B) \Rightarrow x : A \to B \to A}
}{t : A \wedge B \vdash a\ (\text{fun}(x : A)(y : B) \Rightarrow x) : A \wedge B \to A} \qquad \cfrac{}{t : A \wedge B \vdash t : A \wedge B}
}{t : A \wedge B \vdash a\ (\text{fun}(x : A)(y : B) \Rightarrow x)\ t : A}
}{\textbf{Lam} \cfrac{}{\vdash \text{fun}(t : A \wedge B) \Rightarrow a\ (\text{fun}(x : A)(y : B) \Rightarrow x)\ t : A \wedge B \to A}}
$$

🔵 Some assumptions are elided with a dot "·".

🔵 Lower occurrences of a have A, B, and A as implicit arguments.

∨-**Introduction in Coq**

🔵 Definitions for ∨:

```
Inductive or (A : Prop) (B : Prop) : Prop :=
    or_introl : A -> A \/ B | or_intror : B -> A \/ B
```

Note: "A \/ B" is a convenient notation for "or A B".

🔵 Proposition to prove:

```
Lemma or_intro: A -> (A \/ B).
```

🔵 A proof in Coq:

```
intro x; left; assumption.
```

🔵 The proof term in Coq:

```
or_intro = fun x : A => or_introl B x
    : A -> A \/ B
```

∨-**Elimination in Coq**

🔵 Definitions for ∨:

```
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
```

∨-**Elimination in Coq (cont.)**

🔵 Proposition to prove:

Lemma or_elim: (A -> C) -> (B -> C) -> ((A\/B) -> C).

🔵 A proof in Coq:

intro xt1; intro xt2; intro t; elim t;
assumption; assumption.

🔵 The proof term in Coq:

fun (xt1 : A -> C) (xt2 : B -> C) (t : A \/ B)
      => or_ind xt1 xt2 t
   : (A -> C) -> (B -> C) -> A \/ B -> C

⊤-**Introduction in Coq**

- Definitions for ⊤:
  ```
  Inductive True : Prop :=  I : True
  True_ind =
  fun P : Prop => True_rect (P:=P)
      : forall P : Prop, P -> True -> P
  ```
- Proposition to prove:

  ```
  Lemma True_intro: True.
  ```
- A proof in Coq:

  ```
  exact I.
  ```
- The proof term in Coq:
  ```
  True_intro = I
      : True
  ```

⊥-Elimination in Coq

- Definitions for ⊥:

  ```
  Inductive False : Prop :=  /* nothing */
  False_ind = fun P : Prop => False_rect P
       : forall P : Prop, False -> P
  ```

- Proposition to prove:

  ```
  Lemma False_elim: False -> A.
  ```

- A proof in Coq:

  ```
  intro bot; elim bot.
  ```

- The proof term in Coq:

  ```
  False_elim = fun bot : False => False_ind A bot
       : False -> A
  ```

∀-**Introduction and Elimination in Coq**

- ∀ is a primitive in Coq (pCIC)
- Proposition to prove:

  ```
  Variable D: Set.
  Variables P Q: D -> Prop.
  Section All.
  Hypothesis h: (forall x, P x).
  Lemma all_intro: (forall y, P y).
  End All.
  ```
- A proof in Coq:

  ```
  intro; apply h.
  ```
- The proof term in Coq:

  ```
  all_intro = fun y : D => h y
       : forall y : D, P y
  ```

∀-**Elimination in Coq**

🌐 ∀ is a primitive in Coq (pCIC)

🌐 Proposition to prove:

```
Section All_Elim.
Hypothesis h: (forall x, P x).
Variable t: D.
Lemma all_elim: P t.
End All_Elim.
```

🌐 A proof in Coq:

```
apply h.
```

🌐 The proof term in Coq:

```
all_elim = h t
     : P t
```

∃-Introduction in Coq

🔵 Definitions for ∃:

```
Inductive ex (A : Type) (P : A -> Prop) : Prop :=
    ex_intro : forall x : A, P x -> ex P

Notation "'exists' x , p" := (ex (fun x => p))
  (at level 200, x ident, right associativity)
  : type_scope.
```

🔵 Proposition to prove:

```
Variable t: D.
Lemma exists_intro: P t -> exists x, P x.
```

∃-Introduction in Coq (cont.)

- A proof in Coq:

  ```
  intro; exists t; assumption.
  ```

- The proof term in Coq:

  ```
  exists_intro =
  fun H : P t => ex_intro (fun x : D => P x) t H
      : P t -> exists x : D, P x
  ```

∃-**Elimination in Coq**

🔵 Definitions for ∃:

```
ex_ind =
fun (A : Type) (P : A -> Prop) (P0 : Prop)
  (f : forall x : A, P x -> P0)
  (e : ex P) => match e with
                | ex_intro x x0 => f x x0
                end
  : forall (A : Type) (P : A -> Prop) (P0 : Prop),
    (forall x : A, P x -> P0) -> ex P -> P0
```

🔵 Proposition to prove:

```
Lemma exist_elim: (exists x, P x /\ Q x)
                    -> (exists x, P x).
```

∃-**Elimination in Coq (cont.)**

🔵 A proof in Coq:
```
intro; elim H; intro; intro; elim H0;
intro; intro; exists x; assumption.
```
🔵 The proof term in Coq:
```
exists_elim =
fun H : exists x : D, P x /\ Q x =>
ex_ind
  (fun (x : D) (H0 : P x /\ Q x) =>
   and_ind (fun (H1 : P x) (_ : Q x) =>
       ex_intro (fun x0 : D => P x0) x H1) H0) H
  : (exists x : D, P x /\ Q x) -> exists x : D, P x
```

=-Introduction in Coq

- Definitions for $=$:

  ```
  Inductive eq (A : Type) (x : A) : A -> Prop :=
      refl_equal : x = x
  ```

  Note: "x = x" is a convenient notation for "eq x x".

- Proposition to prove:

  ```
  Lemma eq_intro: t1=t1.
  ```

- A proof in Coq:

  ```
  reflexivity.
  ```

- The proof term in Coq:

  ```
  eq_intro = refl_equal t1
      : t1 = t1
  ```

=-Elimination in Coq (cont.)

🟡 The proof term in Coq:

```
eq_elim =
fun H : t1 = t2 => eq_ind t1 (fun d : D => d = t1)
                               (refl_equal t1) t2 H
    : t1 = t2 -> t2 = t1
```