

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

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

Natural Deduction in the Sequent Form

$$\frac{}{\Gamma, A \vdash A} \text{ (Hyp)}$$

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

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

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

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

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

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

Note: “ Γ, A ” is a shorthand for “ $\Gamma \cup \{A\}$ ”, I for Introduction, and E for Elimination.

Natural Deduction (cont.)

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} (\rightarrow I)$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} (\rightarrow E)$$

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

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash B} (\neg E)$$

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

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

$$\frac{}{A_1, \dots, A_i, \dots, A_n \vdash A_i} (\text{Hyp}^i)$$

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

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

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

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

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

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

Intuitionistic Natural Deduction (cont.)

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

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} (\rightarrow I)$$

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} (\rightarrow E)$$

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash A} (\perp E)$$

Note:

- 🌐 The last rule says that, if we can deduce \perp (representing a contradiction), then we can deduce anything.
- 🌐 $\neg A$ is then represented (i.e., defined) as $A \rightarrow \perp$.
- 🌐 With the \rightarrow -elimination rule, one can deduce a contradiction (and hence anything) from $\neg A$ and A .

Quantifier Rules

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

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

$$\frac{\Gamma \vdash B[t/x]}{\Gamma \vdash \exists x B} (\exists I)$$

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

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.

$$\frac{}{x_1 : A_1, \dots, x_i : A_i, \dots, x_n : A_n \vdash x_i : A_i} (\text{Hyp}^i)$$

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

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

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

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

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

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

Note: $\mathbf{case}(\mathbf{inl}(B, t'), x.t_1, y.t_2) \stackrel{\Delta}{=} t_1[t'/x]$;
 $\mathbf{case}(\mathbf{inr}(A, t'), x.t_1, y.t_2) \stackrel{\Delta}{=} t_2[t'/y]$.

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

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

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

Note: In the $(\rightarrow I)$ rule above, it is assumed that B does not contain x (so B does not depend on x).

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

An Example Proof

$$\frac{\frac{\frac{\overline{x : (A \vee B) \rightarrow C, y : A \vdash x : (A \vee B) \rightarrow C} \text{ (Hyp}^1)}{\alpha} (\rightarrow E)}{\frac{x : (A \vee B) \rightarrow C, y : A \vdash x \text{ inl}(B, y) : C} {x : (A \vee B) \rightarrow C \vdash \text{fun}(y : A) \Rightarrow x \text{ inl}(B, y) : A \rightarrow C} (\rightarrow I)}}{\vdash \text{fun}(x : (A \vee B) \rightarrow C) \Rightarrow \text{fun}(y : A) \Rightarrow x \text{ inl}(B, y) : ((A \vee B) \rightarrow C) \rightarrow (A \rightarrow C)} (\rightarrow I)$$

$$\alpha: \frac{\overline{x : (A \vee B) \rightarrow C, y : A \vdash y : A} \text{ (Hyp}^2)}{x : (A \vee B) \rightarrow C, y : A \vdash \text{inl}(B, y) : A \vee B} (\vee I)$$

Intuitionistic ND with Proof Terms (cont.)

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

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

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

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

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

Curry-Howard Correspondence

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

Main Typing/Inference Rules in Coq/CIC

- 🌐 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 (CIC, the type theory behind Coq):

$$\mathcal{S} = \{\text{SProp}, \text{Prop}, \text{Set}, \text{Type}(1), \text{Type}(2), \text{Type}(3), \dots\}$$

- ☀️ $\text{SProp} : \text{Type}(1)$.
- ☀️ $\text{Prop} : \text{Type}(1)$.
- ☀️ $\text{Set} : \text{Type}(1)$.
- ☀️ $\text{Type}(i) : \text{Type}(i + 1)$, for $i \geq 1$.

Main Typing/Inference Rules in CIC (cont.)

Prod-Prop

$$\frac{E[\Gamma] \vdash A : s \quad s \in \mathcal{S} \quad E[\Gamma :: (a : A)] \vdash B : \text{Prop}}{E[\Gamma] \vdash \forall a : A, B : \text{Prop}}$$

Prod-Set /* Set is predicative. */

$$\frac{E[\Gamma] \vdash A : s \quad s \in \{\text{SProp}, \text{Prop}, \text{Set}\} \quad E[\Gamma :: (a : A)] \vdash B : \text{Set}}{E[\Gamma] \vdash \forall a : A, B : \text{Set}}$$

Prod-Type

$$\frac{E[\Gamma] \vdash A : s \quad s \in \{\text{SProp}, \text{Type}(i)\} \quad E[\Gamma :: (a : A)] \vdash B : \text{Type}(i)}{E[\Gamma] \vdash \forall a : A, B : \text{Type}(i)}$$

Main Typing/Inference Rules in CIC (cont.)

$$\mathbf{Lam} \frac{E[\Gamma] \vdash A \rightarrow B : s \quad E[\Gamma :: (v : A)] \vdash e : B}{E[\Gamma] \vdash \text{fun } v : A \Rightarrow e : A \rightarrow B}$$

where B does not depend on v .

$$\mathbf{App} \frac{E[\Gamma] \vdash e_1 : A \rightarrow B \quad E[\Gamma] \vdash e_2 : A}{E[\Gamma] \vdash e_1 e_2 : B}$$

$$\mathbf{Lam} \frac{E[\Gamma] \vdash \forall v : A, B : s \quad E[\Gamma :: (v : A)] \vdash t : B}{E[\Gamma] \vdash \text{fun } v : A \Rightarrow t : \forall v : A, B}$$

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

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

Main Typing/Inference Rules in CIC (cont.)

$$\mathbf{Lam} \frac{E[\Gamma] \vdash \forall v : A, B : s \quad E[\Gamma :: (v : A)] \vdash t : B}{E[\Gamma] \vdash \text{fun } v : A \Rightarrow t : \forall v : A, B}$$

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

$$\mathbf{Let} \frac{E[\Gamma] \vdash t_1 : A \quad E[\Gamma :: (x := t_1 : A)] \vdash t_2 : B}{E[\Gamma] \vdash \text{let } x := t_1 : A \text{ in } t_2 : B[t_1/x]}$$

The Example Proof in Coq

- The proposition to prove: $((A \vee B) \rightarrow C) \rightarrow (A \rightarrow C)$.

Formalization in Coq:

Variables `A B C: Prop.`

Lemma `t0: ((A \vee B) -> C) -> (A -> C).`

- A proof in Coq:

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

- The proof term in Coq:

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

cf.: **fun**($x : (A \vee B) \rightarrow C$) \Rightarrow **fun**($y : A$) \Rightarrow $x \text{ inl}(-, y)$
 : $((A \vee B) \rightarrow C) \rightarrow (A \rightarrow C)$

\wedge -Introduction in Coq

🌐 Definition of and (\wedge):

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:

Hypothesis t1: A.

Hypothesis t2: B.

Lemma and_I: A /\ B.

🌐 A proof in Coq:

split; assumption; assumption.

🌐 The proof term in Coq:

and_I = conj t1 t2
 : A /\ B

\wedge -Elimination in Coq

- Automatically generated `and_ind`:

```
and_ind =
fun (A B P : Prop) (f : A -> B -> P)
  (a : A /\ B) =>
match a with
| conj x x0 => f x x0
end
      : forall A B P : Prop,
        (A -> B -> P) -> A /\ B -> P
```

- Proposition to prove:

Hypothesis `t`: `A /\ B`.
 Lemma `and_E1`: `A`.

- A proof in Coq:

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

- The proof term in Coq:

```
and_E1 = and_ind (fun (x : A) (_ : B) => x) t
      : A
```

\wedge -Elimination in Coq (cont.)

$$\begin{array}{c}
 \text{Lam} \frac{\overline{t : A \wedge B, x : A, y : B \vdash x : A}}{t : A \wedge B, x : A \vdash \text{fun}(y : B) \Rightarrow x : B \rightarrow A} \\
 \text{Lam} \frac{\overline{t : A \wedge B \vdash \text{fun}(x : A)(y : B) \Rightarrow x : A \rightarrow B \rightarrow A}}{t : A \wedge B \vdash \text{and_ind}(\text{fun}(x : A)(y : B) \Rightarrow x) : A \wedge B \rightarrow A} \\
 \text{App } \alpha \frac{\text{App} \frac{\overline{t : A \wedge B \vdash \text{and_ind}(\text{fun}(x : A)(y : B) \Rightarrow x) : A \wedge B \rightarrow A} \quad \overline{t : A \wedge B \vdash t : A \wedge B}}{t : A \wedge B \vdash \text{and_ind}(\text{fun}(x : A)(y : B) \Rightarrow x) t : A}}{}
 \end{array}$$

α :

(Let a denote `and_ind`; further details omitted)

$$\begin{array}{c}
 \text{App} \frac{\overline{\cdot \vdash a : \forall A, B, P : \text{Prop}, (A \rightarrow B \rightarrow P) \rightarrow A \wedge B \rightarrow P} \quad \overline{\cdot \vdash A : \text{Prop}}}{\text{App} \frac{\overline{\cdot \vdash a A : \forall B, P : \text{Prop}, (A \rightarrow B \rightarrow P) \rightarrow A \wedge B \rightarrow P} \quad \overline{\cdot \vdash B : \text{Prop}}}{\text{App} \frac{\overline{\cdot \vdash a A B : \forall P : \text{Prop}, (A \rightarrow B \rightarrow P) \rightarrow A \wedge B \rightarrow P} \quad \overline{\cdot \vdash A : \text{Prop}}}{t : A \wedge B \vdash a A B A : (A \rightarrow B \rightarrow A) \rightarrow A \wedge B \rightarrow A}}}
 \end{array}$$

Note: the antecedent in each sequent should really be like $E[\Gamma :: (t : A \wedge B) \dots]$ and is sometimes elided as a dot “ \cdot ”. The upper occurrences of `and_ind` have A , B , and A as implicit arguments.

\wedge -Elimination in Coq (cont.)

- Automatically generated `and_ind`:

```
and_ind =
fun (A B P : Prop) (f : A -> B -> P)
  (a : A /\ B) =>
match a with
| conj x x0 => f x x0
end
      : forall A B P : Prop,
        (A -> B -> P) -> A /\ B -> P
```

- Proposition to prove:

Hypothesis `t`: `A /\ B`.

Lemma `and_E1_alt`: `A`.

- A proof in Coq:

`apply t.`

- The proof term in Coq:

```
and_E1_alt =
let H : A := match t with
  | conj x _ => x
end in
```

`H`

: `A`

\wedge -Elimination in Coq (cont.)

Apply $\frac{t : A \wedge B \vdash t : A \wedge B}{t : A \wedge B \vdash \text{let } H : A := \text{match } t \text{ with } | \text{conj } x _ \Rightarrow x \text{ end in } H : A}$

or

Apply $\frac{t : A \wedge B \vdash t : A \wedge B}{t : A \wedge B \vdash \text{match } t \text{ with } | \text{conj } x _ \Rightarrow x \text{ end} : A}$

Note : the term “match t with | conj $x _ \Rightarrow x$ end” plays the role of $\mathbf{fst}(t)$ as in the $\wedge E_1$ rule of the Intuitionistic ND with Proof Terms, as shown again below.

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

\vee -Introduction in Coq

Definition of or (\vee):

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

Note: “ $A \ \vee \ B$ ” is a convenient notation for “or $A \ B$ ”.

Proposition to prove:

Hypothesis t : A .

Lemma or_I1 : $A \ \vee \ B$.

A proof in Coq:

```
left; assumption.
```

The proof term in Coq:

```
or_I1 = or_introl t  
      : A \\/ B
```

V-Elimination in Coq

🌐 Automatically generated `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

```

🌐 Proposition to prove:

```

Hypothesis t: A \\/ B.
Hypothesis t1: A -> C.
Hypothesis t2: B -> C.
Lemma or_E: C.

```

🌐 A proof in Coq:

```

elim t; assumption; assumption.

```

🌐 The proof term in Coq:

```

or_E = or_ind t1 t2 t
      : C

```

T-Introduction in Coq

🌐 Definition of True (\top):

```
Inductive True : Prop := I : True
```

🌐 Automatically generated True_ind:

```
True_ind =  
fun (P : Prop) (f : P) (t : True) =>  
match t with  
| I => f  
end  
: forall P : Prop, P -> True -> P
```

🌐 Proposition to prove:

```
Lemma true_I: True.
```

🌐 A proof in Coq:

```
exact I.
```

🌐 The proof term in Coq:

```
true_I = I  
: True
```

\perp -Elimination in Coq

🌐 Definition of False (\perp):

```
Inductive False : Prop :=
```

🌐 Automatically generated False_ind:

```
False_ind =
fun (P : Prop) (f : False) =>
match f return P with
end
      : forall P : Prop, False -> P
```

🌐 Proposition to prove:

```
Hypothesis t: False.
```

```
Lemma false_E: A.
```

🌐 A proof in Coq:

```
elim t.
```

🌐 The proof term in Coq:

```
false_E = False_ind A t
      : A
```

\forall -Introduction in Coq

🌐 The universal quantifier `forall` (\forall) is a primitive in Coq (CIC)

🌐 Proposition to prove:

```
Variable D: Set.
```

```
Variables P Q: D -> Prop.
```

```
Section All_I.
```

```
Hypothesis h: (forall x, P x).
```

```
Lemma all_I: (forall y, P y).
```

```
...
```

```
End All_I.
```

🌐 A proof in Coq:

```
intro; apply h.
```

🌐 The proof term in Coq:

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

\forall -Elimination in Coq

🌐 The universal quantifier forall (\forall) is a primitive in Coq (CIC)

🌐 Proposition to prove:

Hypothesis h: (forall x, P x).

Variable t: D.

Lemma all_E: P t.

🌐 A proof in Coq:

apply h.

🌐 The proof term in Coq:

```
all_E = h t
      : P t
```

\exists -Introduction in Coq

Definition of ex (\exists):

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

Notation for ex \exists :

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

Proposition to prove:

```
Variable t1: D.
Hypothesis t2: P t1.
Lemma exists_I: exists x, P x.
```

A proof in Coq:

```
exists t1; assumption.
```

The proof term in Coq:

```
exists_I =
ex_intro (fun x : D => P x) t1 t2
  : exists x : D, P x
```

∃-Elimination in Coq

- Automatically generated `ex_ind`:

```

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:

Hypothesis `t`: `exists x, P x`.
 Lemma `exists_E`: `(exists y, P y)`.

- A proof in Coq:

```
exact t.
```

- The proof term in Coq:

```

exists_E = t
: exists y : D, P y

```


=-Introduction in Coq

🌐 Definition of eq (=):

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

Note: “ $x = x$ ” is a convenient notation for “ $\text{eq } x \ x$ ”.

🌐 Proposition to prove:

Lemma eq_I: $t1=t1$.

🌐 A proof in Coq:

reflexivity.

🌐 The proof term in Coq:

```
eq_I = eq_refl  
      : t1 = t1
```

==Elimination in Coq

- Automatically generated `eq_ind`:

```
eq_ind =
fun (A : Type) (x : A) (P : A -> Prop)
  (f : P x) (y : A) (e : x = y) =>
match e in (_ = y0) return (P y0) with
| eq_refl => f
end
      : forall (A : Type) (x : A) (P : A -> Prop),
        P x -> forall y : A, x = y -> P y
```

- Proposition to prove:

Hypothesis H: $t_1=t_2$.

Lemma `eq_E`: $t_2=t_1$.

- A proof in Coq:

```
rewrite <- H; reflexivity.
```

- The proof term in Coq:

```
eq_E =
eq_ind t1 (fun d : D => d = t1) eq_refl t2 H
      : t2 = t1
```