形式化方法导引

第 6 章 案例分析

6.3 Coq: A Prover based on Higher-order Logic

6.3.1 Introduction | 6.3.2 Basics

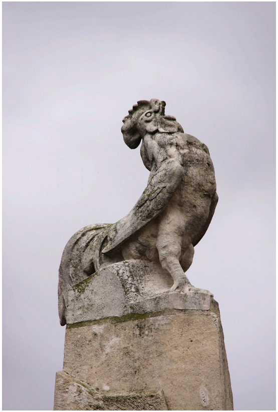
黄文超

https://faculty.ustc.edu.cn/huangwenchao

$\longrightarrow$ 教学课程 $\longrightarrow$ 形式化方法导引

本章代码链接

(Tested in v8.14.1, v8.15.1)



ROCQ

Coq proof assistant

- "Rooster": a symbol of France
- Calculus of constructions
- Thierry Coquand

Renamed: ROCQ

1. Introduction

推荐网站及书目:

- 【康奈尔】Coq 编程基础
- Coq'Art book
- Coq/ROCQ 官方网站
- Software Foundations
- Coq in a Hurry
- 形式化方法公开课之定理证明

Coq Features

- *Define* computable functions and logical predicates
- *State* mathematical theorems and software specifications
- *Develop* proofs of theorems, interactively
- *Machine-check* the proofs
- *Extract* certified code from the proofs, e.g., OCaml, Haskell, Scheme

Pure mathematics

- the Fundamental theorem of Algebra
  - every polynomial has a root in complex field
- Feit-Thompson theorem on finite groups
- the four-color theorem

Related systems

- *Lean*: a theorem prover and programming language

Formalizing programming environments

- *CompCert*: a certified optimizing compiler for C
- *Certicrypt*: an environment of formal proofs for computational cryptography
- JavaCard platform - the Gemalto and Trusted Logic companies
    - the *highest level* of certification (common criteria EAL 7)
- Ynot library: for proving imperative programs using separation logic

Related systems

- *Isabelle/HOL*: a generic proof assistant
    - *Sel4*: a microkernel that has been formally verified

Rich environment

- to help designing theories and proofs offering mechanisms
  - like user extensible notations, tactics for proof automation, libraries
- can be used and extended safely
  - ultimately any definition and proof is checked by a safe kernel

```
5=2+3
```

Small kernel based on a language with few primitive constructions

- functions, (co)-inductive definitions, product types, sorts
- a limited number of rules for type-checking and computation

```
@eq Z (Zpos (xI (xO xH))) (Zplus (Zpos (xO xH)) (Zpos (xI xH)))
```

1. Introduction

Program verification in Coq:

- One can *express the property* "the program p is correct" as a *mathematical statement*, and prove it is correct
- One can *develop* a specific program *analyzer* (model-checking, abstract interpretation, ...) in Coq, prove it correct and use it
- One can
  - represent the program p by a Coq term t
  - represent the specification by a type T
  - such that $t : T$ (which is *automatically checked*) implies p is correct
  - It works well for functional (possibly *monadic*) programs
- One can use an *external tool* to *generate proof obligations* and then use Coq to *solve obligations*.

Name and Type in Type Theory

A Coq object in the environment has a *name* and a *type*

Command: `Check` term

The command `Check` is used to check the type of an object

```
Check nat.
```

```
nat
     : Set
```

The object `nat` is a predefined type for natural numbers

- its *type* is a special constant `Set` called a *sort*.

Command: `Check term`

The command `Check` is used to check the type of an object

```
Check 0.
```

```
0
     : nat
```

The constant 0 has type `nat`.

Command: `Check term`

The command `Check` is used to check the type of an object

```
Check S.
```

```
S
     : nat -> nat
```

The object `S` is the successor function

- it has type `nat -> nat`

Command: `Check term`

The command `Check` is used to check the type of an object

```
Check plus.
```

```
Nat.add
     : nat -> nat -> nat
```

The binary function `plus` has type `nat -> nat -> nat`

- which should be read as `nat -> (nat -> nat)`

Application of functions

The application of a function to its arguments is written as

- `f x y` for a function `f` of type `A -> B -> C` and arguments `x` and `y` of type `A` and `B`

- The term `f x y` stands for `(f x) y`
- The natural number 10 is represented by the term `S (S (S (S (S (S (S (S (S (S O))))))))))`
- The usual infix notation `x+y` can be used instead of `plus x y`

```
Check (3+2).
```

```
3 + 2
     : nat
```

Propositions as terms

In Coq, *logical propositions* are also seen as *terms*.

- the type of a proposition is called its *sort* `Prop`

The command `Check` verifies a proposition is well-formed

- but does not say if it is true or not

```
Check (1+2=3).
```

```
1 + 2 = 3
     : Prop
```

```
Check (forall x:nat, exists y, x=y+y).
```

```
forall x : nat, exists y : nat, x = y + y
     : Prop
```

Introducing goals as propositions:

Commands:

- `Lemma id: prop.`
- `Theorem id: prop.`
- `Goal prop.`

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
```

Type of proofs: The *type* of a proof is the *proposition* it proves.

Curry-Howard isomorphism (柯里-霍华德同构)

It proposes a deep connection between the world of *logic* and the world of *computation*:

- *propositions* are *types*
- *proofs* are *programs*
    - *proofs* are *computations*

Specific explanation of the *Curry-Howard correspondence*:

There is only one form of judgment $\Gamma \vdash p : A$

- The environment Γ is a *list* of *names* associated with types
- When A is a type of objects, p is a *term* of type A
  - e.g., $x : \mathrm{nat} \vdash x : \mathrm{nat}$
  - p is well-formed in the environment Γ and has type A
- When A is a proposition, p is a *proof* of A
  - e.g., $x : \mathrm{nat},\ h : x = 1 \vdash \ldots : x \neq 0$
  - A is provable under the assumption of Γ and p is a *witness* of that proof

Specific explanation of the *Curry-Howard correspondence*:

There is only one form of judgment $\Gamma \vdash p : A$

- The environment Γ is a *list* of *names* associated with types
- When A is a type of objects, p is a *term* of type A
  - e.g., $x : \mathrm{nat} \vdash x : \mathrm{nat}$
  - p is well-formed in the environment Γ and has type A
- When A is a proposition, p is a *proof* of A
  - e.g., $x : \mathrm{nat},\ h : x = 1 \vdash \ldots : x \neq 0$
  - A is provable under the assumption of Γ and p is a *witness* of that proof

Specific explanation of the *Curry-Howard correspondence*:

There is only one form of judgment $\Gamma \vdash p : A$

- The environment Γ is a *list* of *names* associated with types
- When A is a type of objects, p is a *term* of type A
  - e.g., $x : \mathrm{nat} \vdash x : \mathrm{nat}$
  - p is well-formed in the environment Γ and has type A
- When A is a proposition, p is a *proof* of A
  - e.g., $x : \mathrm{nat},\ h : x = 1 \vdash ... : x \neq 0$
  - A is provable under the assumption of Γ and p is a *witness* of that proof

柯里-霍华德同构的作用

- 让“写程序”变成“写证明”
  - 编程变成构造逻辑证明，程序等于正确性。
- 支持“程序 = 证明”的语言和工具
  - 可机器验证 (*Type System*) 的形式化证明和可信软件系统。

Producing a proof

To establish that a proposition is true, we construct a proof using the following steps:

- State the proposition using `Lemma`, `Theorem`, or `Goal`.
- Enter the proof mode using the `Proof.` command.
- Apply tactics to manipulate the goal and subgoals until all are resolved.
- Conclude the proof with `Qed.` or `Admitted.` (if incomplete).

```
Lemma ex0: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
intros. apply H.
- assumption.
- apply H0. assumption.
Qed.
```

Proving process:

- Lexing and Parsing
- Proof Environment Initialization
- Tactic Interpretation and Execution
- Proof Term Construction
- *Type Checking*
    - extracts a term p and the trusted kernel has to check that
    - $\Gamma \vdash p : A$ is a valid judgment by elementary rules
- Subgoal Management
- QED and Globalization

Rules of proofs:

- *axioms* rules: introduce new assumptions
- *introduction* rules: introduce new variables
- *elimination* rules: manipulate the goal

Default proof routine: *Backward reasoning* with tactics

- A tactic transforms a goal into a set of subgoals
  - solving these subgoals is sufficient to solve the original goal

Axiom rules:

- `assumption` tactic: solves the goal if it is an assumption
- `exact` tactic: solves the goal if it is exactly equal to a hypothesis

$\dfrac{h : A \in \Gamma}{\Gamma \vdash h : A}$	`exact` h or `assumption`

Introduction and elimination rules: (Recall in Chapter 3)

The basic rules of natural deduction:

	introduction	*elimination*
$\land$	$\dfrac{\phi \quad \psi}{\phi \land \psi}\ \land\mathrm{i}$	$\dfrac{\phi \land \psi}{\phi}\ \land\mathrm{e}_1 \qquad \dfrac{\phi \land \psi}{\psi}\ \land\mathrm{e}_2$
$\lor$	$\dfrac{\phi}{\phi \lor \psi}\ \lor\mathrm{i}_1 \qquad \dfrac{\psi}{\phi \lor \psi}\ \lor\mathrm{i}_2$	$\dfrac{\phi \lor \psi \quad \boxed{\begin{array}{c}\phi\\ \vdots\\ \chi\end{array}} \quad \boxed{\begin{array}{c}\psi\\ \vdots\\ \chi\end{array}}}{\chi}\ \lor\mathrm{e}$

Introduction and elimination rules: (Recall in Chapter 3)

$\rightarrow$	$$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \rightarrow \psi}\ {\rightarrow}\mathrm{i}$$	$$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi}\ {\rightarrow}\mathrm{e}$$
$\neg$	$$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\mathrm{i}$$	$$\dfrac{\phi \quad \neg\phi}{\bot}\ \neg\mathrm{e}$$

Introduction and elimination rules: (Recall in Chapter 3)

$\perp$	(no introduction rule for $\perp$)	$\dfrac{\perp}{\phi}\ \perp\mathrm{e}$
$\neg\neg$		$\dfrac{\neg\neg\phi}{\phi}\ \neg\neg\mathrm{e}$

Introduction rule for $\to$:

$$\frac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \psi\end{array}}}{\phi \to \psi}\ \to\mathrm{i}$$

$\to$	$\dfrac{\Gamma, h : A \vdash ? : B}{\Gamma \vdash ? : A \to B}$	`intro` h

- Introduction rules: introduce new variables: h
- Backward reasoning: give a mean to prove a proposition with $\to$
  - if we can prove *simpler* propositions
- A tactic will work with a still *unresolved* goal
  - that we indicate using $?$ in place of the proof-term

Elimination rule for $\rightarrow$:

$$\frac{\phi \qquad \phi \rightarrow \psi}{\psi} \rightarrow\mathrm{e}$$

$\dfrac{\Gamma \vdash h : A \rightarrow B \qquad \Gamma \vdash ? : A}{\Gamma \vdash ? : B}$	`apply` h

- Elimination rule: manipulate the goal
- apply h: Explains how we can use a proof (h) of a proposition with $\rightarrow$

Example for assumption, intro and apply:

```
Lemma ex1: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
intro h1.
intro h2.
intro h3.
intro h4.
intro h5.
intro h6.
apply h4.
assumption.
apply h5.
assumption.
Qed.
```

Recall the example, It would be painful to apply only *atomic* rules

- Tactics usually combine in one step several introductions or elimination rules.

Tactic: intros

Introduce *all* assumptions in the context.

- Do multiple introductions and infer names when none are given

Recall the example, It would be painful to apply only *atomic* rules

```
Lemma ex2: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
intros.
apply H.
assumption.
apply H0.
assumption.
Qed.
```

Some tactics are doing proof search to help solve a goal:

contradiction	solves the goal when `False`, or A and $\neg A$ appear in the hypotheses
tauto	solves propositional tautologies
trivial	tries very simple lemmas to solve the goal
auto	searches in a database of lemmas to solve the goal
intuition	removes the propositional structure of the goal then auto
omega	solves goals in linear arithmetic

```
Lemma ex3: forall A B C:Prop,
(A -> B -> C) -> (A -> B) -> A -> C.
Proof.
auto.
Qed.
```

Keyword: Variable

Objects can be introduced using the syntax

- `Variable x : A.` `Variables x y z : A.`

Keyword: Section

A section is a block of code that can be used to group related definitions and theorems. It is defined using the syntax

- `Section S.` `End S.`

```
Section SectionExample.
Variables A B C: Prop.
Lemma ex4 : (A -> B -> C) -> (A -> B) -> A -> C.
Proof.
intros. apply H. assumption. apply H0. assumption.
Qed.
End SectionExample.
```

It is convenient to *postpone* a proof but it is also potentially *dangerous*

Keyword: Admitted

- introducing the original goal as an axiom.
- Safety is *only guaranteed* if there are no axioms left in the proof.

```
Variables A B C: Prop.
Lemma falselemma: (A -> B) -> C.
Proof.
Admitted.
```

```
Lemma falseconclusion: B -> C.
Proof.
intros.
apply falselemma.
intros. exact H.
Qed.
```

Logical rules and tactics:

- Implication rules: $\rightarrow$
- Conjunction rules: $\wedge$
- Disjunction rules: $\vee$
- Negation rules: $\neg$
- Contradiction rules: $\perp$
- Double negation rules: $\neg\neg$
- Universal quantification rules: $\forall$
- Existential quantification rules: $\exists$
- Equality rules: $=$

Current Coq tactics: axiom intro apply $\rightarrow$ $\wedge$ $\vee$ $\neg$ $\perp$ $\neg\neg$ $\forall$ $\exists$ $=$

Rules in Classical Logics

$\rightarrow$	$$\dfrac{\boxed{\begin{array}{c}\phi \\ \vdots \\ \psi\end{array}}}{\phi \rightarrow \psi}\ {\rightarrow}\mathrm{i}$$	$$\dfrac{\phi \quad \phi \rightarrow \psi}{\psi}\ {\rightarrow}\mathrm{e}$$

Rules in Coq

$\rightarrow$	$$\dfrac{\Gamma, h{:}A \vdash ?{:}B}{\Gamma \vdash ?{:}A \rightarrow B}$$	`intro` h	$$\dfrac{\Gamma \vdash h{:}A \rightarrow B \qquad \Gamma \vdash ?{:}A}{\Gamma \vdash ?{:}B}$$	`apply` h

See intro apply

2.2 Logical rules and tactics – $\wedge$

Rules in Classical Logics

$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi}\wedge\mathrm{i}$	$\dfrac{\phi \wedge \psi}{\phi}\wedge\mathrm{e}_1$	$\dfrac{\phi \wedge \psi}{\psi}\wedge\mathrm{e}_2$

Rules in Coq

$\wedge$	$\dfrac{\Gamma \vdash ?:A \quad \Gamma \vdash ?:B}{\Gamma \vdash ?:A \wedge B}$	split	$\dfrac{\Gamma \vdash h:A \wedge B \quad \Gamma, l:A, m:B \vdash ?:C}{\Gamma \vdash ?:C}$	destruct h as (l, m)

```
Variables A B : Prop
Lemma and_intro : A -> B -> A /\ B.
Proof.
    intros HA HB.
    split.
    exact HA.
    exact HB.
Qed.
```

Rules in Classical Logics

$\wedge$	$\dfrac{\phi \quad \psi}{\phi \wedge \psi}\wedge\mathrm{i}$	$\dfrac{\phi \wedge \psi}{\phi}\wedge\mathrm{e}_1$	$\dfrac{\phi \wedge \psi}{\psi}\wedge\mathrm{e}_2$

Rules in Coq

$\wedge$	$\dfrac{\Gamma \vdash ?{:}A \quad \Gamma \vdash ?{:}B}{\Gamma \vdash ?{:}A \wedge B}$	`split`	$\dfrac{\Gamma \vdash h{:}A \wedge B \quad \Gamma, l{:}A, m{:}B \vdash ?{:}C}{\Gamma \vdash ?{:}C}$	`destruct` h `as` (l, m)

```
Variables A B : Prop.
Lemma and_elim_ascii : A /\ B -> A.
Proof.
  intros H.
  destruct H as [HA HB].
  exact HA.
Qed.
```

Rules in Classical Logics

$\vee$	$\dfrac{\phi}{\phi \vee \psi}\,\vee i_1$ $\quad$ $\dfrac{\psi}{\phi \vee \psi}\,\vee i_2$ $\quad$ $\dfrac{\phi \vee \psi \quad \boxed{\begin{matrix}\phi \\ \vdots \\ \chi\end{matrix}} \quad \boxed{\begin{matrix}\psi \\ \vdots \\ \chi\end{matrix}}}{\chi}\,\vee e$

Rules in Coq

$\vee$	$\dfrac{\Gamma \vdash ?:A}{\Gamma \vdash ?:A \vee B}$	left	$\dfrac{\Gamma \vdash h:A \vee B \quad \Gamma, l:A \vdash ?:C \quad \Gamma, l:B \vdash ?:C}{\Gamma \vdash ?:C}$	destruct h as $[l\|l]$
	$\dfrac{\Gamma \vdash ?:B}{\Gamma \vdash ?:A \vee B}$	right		

```
Variables A B : Prop.
Lemma or_intro_left_example : A -> A \/ B.
Proof.
    intros HA.
    left.
    exact HA.
Qed.
```

```
Variables A B C : Prop.
Lemma or_elim_example : A \/ B -> (A -> C) -> (B -> C) -> C.
Proof.
  intros H_or HA_to_C HB_to_C.
  destruct H_or as [HA | HB].
  - apply HA_to_C. exact HA.
  - apply HB_to_C. exact HB.
Qed.
```

Rules in Classical Logics

¬	$$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg i$$	$$\dfrac{\phi \quad \neg\phi}{\bot}\ \neg e$$

Rules in Coq

¬	$\dfrac{\Gamma, h{:}A \vdash \texttt{False}}{\Gamma \vdash ?{:}\neg A}$	`intro` h	$\dfrac{\Gamma \vdash h{:}\neg A \qquad \Gamma \vdash ?{:}A}{\Gamma \vdash ?{:}C}$	`destruct` h

```
Variables A : Prop.
Lemma not_intro_example : (A -> False) -> ~A.
Proof.
  intros H.
  exact H.
Qed.
```

Rules in Classical Logics

¬	$\dfrac{\boxed{\begin{array}{c}\phi\\ \vdots\\ \bot\end{array}}}{\neg\phi}\ \neg\mathrm{i}$	$\dfrac{\phi \quad \neg\phi}{\bot}\ \neg\mathrm{e}$

Rules in Coq

¬	$\dfrac{\Gamma, h : A \vdash \texttt{False}}{\Gamma \vdash ? : \neg A}$	`intro` h	$\dfrac{\Gamma \vdash h : \neg A \qquad \Gamma \vdash ? : A}{\Gamma \vdash ? : C}$	`destruct` h

```
Variable A : Prop.
Lemma not_elim_with_destruct : A -> ~A -> False.
Proof.
  intros HA HNA.
  destruct HNA.
  exact HA.
Qed.
```

Rules in Classical Logics

$\bot$	(no introduction rule for $\bot$)	$\dfrac{\bot}{\phi}\ \bot\mathrm{e}$

Rules in Coq

$\bot$			$\dfrac{\Gamma \vdash ?{:}\texttt{False}}{\Gamma \vdash ?{:}C}$	exfalso

```
Variable A : Prop.
Lemma false_elim_with_exfalso : A -> ~A -> forall B : Prop, B.
Proof.
  intros HA HNA B.
  exfalso.
  apply HNA. exact HA.
Qed.
```

Rules in Classical Logics

$$\neg\neg \qquad \frac{\neg\neg\phi}{\phi}\ \neg\neg e$$

Rules in Coq

Does not exist!!! Why?

Rules in Classical Logics

$$\frac{\boxed{\begin{array}{l} x_0 \\ \quad\vdots \\ \phi[x_0/x] \end{array}}}{\forall x\, \phi}\ \forall x\,\text{i.} \qquad\qquad \frac{\forall x\, \phi}{\phi[t/x]}\ \forall x\,\text{e.}$$

Rules in Coq

∀	$\dfrac{\Gamma, y{:}A \vdash ?{:}B[x \leftarrow y]}{\Gamma \vdash ?{:}\forall x{:}A, B}$	`intro` y	$\dfrac{\Gamma \vdash h{:}\forall x{:}A, B \qquad \Gamma \vdash t{:}A}{\Gamma \vdash ?{:}B[x \leftarrow t]}$	`apply` y `with` $(x:=t)$

```
Variable A : nat -> Prop.
Variable H : forall n : nat, A n.
Lemma forall_intro_example2 : forall n : nat, A n.
Proof.
  intros n.
  apply H.
Qed.
```

Rules in Classical Logics

$$\dfrac{\boxed{\begin{array}{l} x_0 \\ \quad\quad \vdots \\ \quad \phi[x_0/x] \end{array}}}{\forall x\, \phi}\ \forall x\,\mathrm{i.} \qquad\qquad \dfrac{\forall x\, \phi}{\phi[t/x]}\ \forall x\,\mathrm{e.}$$

Rules in Coq

$\forall$	$\dfrac{\Gamma, y{:}A \vdash ?{:}B[x \leftarrow y]}{\Gamma \vdash ?{:}\forall x{:}A, B}$	`intro` y	$\dfrac{\Gamma \vdash h{:}\forall x{:}A, B \qquad \Gamma \vdash t{:}A}{\Gamma \vdash ?{:}B[x \leftarrow t]}$	`apply` y `with` $(x{:=}t)$

```
Variable A : nat -> Prop.
Lemma forall_elim_example : (forall n : nat, A n) -> A 0.
Proof.
  intros H.
  apply H.
Qed.
```

Rules in Classical Logics

$$\frac{\phi[t/x]}{\exists x \phi}\ \exists x\,\mathrm{i.} \qquad \frac{\exists x\,\phi \quad \boxed{\begin{array}{c} x_0\ \phi[x_0/x] \\ \vdots \\ \chi \end{array}}}{\chi}\ \exists\mathrm{e.}$$

Rules in Coq

∃	$\dfrac{\Gamma \vdash t:A \quad \Gamma \vdash ?:B[x \leftarrow t]}{\Gamma \vdash ?:\exists x:A,B}$	`exists` t	$\dfrac{\Gamma \vdash h:\exists x:A,B \quad \Gamma,x:A,l:B \vdash ?:C}{\Gamma \vdash ?:C}$	`destruct` h `as` (x,l)

```
Variable A : nat -> Prop.
Lemma exists_intro_example : A 0 -> exists n : nat, A n.
Proof.
  intros H.
  exists 0.
  exact H.
Qed.
```

Rules in Classical Logics

$$\frac{\phi[t/x]}{\exists x \phi}\ \exists x\,\mathrm{i}. \qquad \frac{\exists x\ \phi \quad \boxed{\begin{array}{c} x_0\ \phi[x_0/x] \\ \vdots \\ \chi \end{array}}}{\chi}\ \exists\mathrm{e}.$$

Rules in Coq

∃	$\frac{\Gamma \vdash t:A \quad \Gamma \vdash ?:B[x \leftarrow t]}{\Gamma \vdash ?:\exists x:A,B}$	`exists` t	$\frac{\Gamma \vdash h:\exists x:A,B \quad \Gamma,x:A,l:B \vdash ?:C}{\Gamma \vdash ?:C}$	`destruct` h `as` (x,l)

```
Variable A : nat -> Prop.
Lemma exists_elim_example : (exists n : nat, A n) -> forall P :
    Prop, (forall x, A x -> P) -> P.
Proof.
  intros Hexists P Hforall. destruct Hexists as [n HAn].
  apply Hforall with (x := n). exact HAn.
Qed.
```

Rules in Classical Logics

$$\frac{}{t = t}=\text{i} \qquad \frac{t_1 = t_2 \quad \phi[t_1/x]}{\phi[t_2/x]}=\text{e}.$$

Rules in Coq

$=$	$\dfrac{t \equiv u}{\Gamma \vdash ?{:}t = u}$	`reflexivity`	$\dfrac{\Gamma \vdash h{:}t = u \qquad \Gamma \vdash ?{:}C[x \leftarrow u]}{\Gamma \vdash ?{:}C[x \leftarrow t]}$	`rewrite` h

- Two terms u and v are *convertible* (written $t \equiv u$) when they represent the same value *after computation*.
- The *elimination* rule allows to *replace* a term by an equal in any context.

Rules in Coq

$=$	$\dfrac{t \equiv u}{\Gamma \vdash ? : t = u}$	`reflexivity`	$\dfrac{\Gamma \vdash h : t = u \qquad \Gamma \vdash ? : C[x \leftarrow u]}{\Gamma \vdash ? : C[x \leftarrow t]}$	`rewrite` h

$\dfrac{\Gamma \vdash ? : u = t}{\Gamma \vdash ? : t = u}$	`symmetry`
$\dfrac{\Gamma \vdash ? : t = v \qquad \Gamma \vdash ? : v = u}{\Gamma \vdash ? : t = u}$	`transitivity` v
$\dfrac{\Gamma \vdash ? : f = g \qquad \Gamma \vdash ? : t_1 = u_1 \ldots \Gamma \vdash ? : t_n = u_n}{\Gamma \vdash ? : f\ t_1 \ldots t_n = g\ u_1 \ldots u_n}$	`f_equal`

Recall Backward reasoning.

Forward reasoning

- A tactic transforms a goal into a set of subgoals
  - solving these subgoals is sufficient to solve the original goal

$\dfrac{\Gamma \vdash ? : B \qquad \Gamma, h : B \vdash ? : A}{\Gamma \vdash ? : A}$	`assert` $(h : B)$

```
Variables A B C : Prop.
Lemma assert_example : (A -> B) -> (B -> C) -> A -> C.
Proof.
  intros H1 H2 HA.
  assert (HB : B).
  - apply H1. exact HA.
  - apply H2. exact HB.
Qed.
```

Coq implements an intuitionistic logic, which differs from classical logic in the following ways:

- *Law of Excluded Middle*: $P \lor \neg P$ is not generally valid in Coq.
- *Double Negation Elimination*: In Coq, $\neg\neg P \to P$ is *not valid* unless P is constructively proven.
  - $P \lor \neg P \vdash \neg\neg P \to P$ in Coq (见作业 5)
- *Constructive Proofs*: Intuitionistic logic requires constructive proofs, meaning that to prove $\exists x, P(x)$, one must explicitly construct a witness x such that $P(x)$ holds.
- *Proof as Programs*: Intuitionistic logic aligns with the *Curry-Howard correspondence*.

It is also possible to use *classical* versions of logical connectives

- a library Classical introduces *the excluded middle* as an axiom

```
Require Import Classical.
```

实验小作业: 使用 Coq 证明如下命题（不允许使用搜索策略，不允许使用 Classical 库），附上代码和文档（文档中列出每个证明步骤的输出截图）

```
Section Homework.
Variables A B : Prop.
Variable T : Type.
Variable P : T -> Prop .

Lemma homework1: forall A, ~~~ A -> ~ A.
Lemma homework2: A \/ B -> ~ (~ A /\ ~ B).
Lemma homework3: (~ exists x, P x) -> forall x, ~ P x.
Lemma homework4: A -> ~~A.
Lemma homework5: (A \/ ~A) -> (~~A -> A).
End Homework.
```