Quantifiers, equality, . . .

Pierre Castéran

Suzhou, August 2011

In this lecture, we shall see how to write (and hopefully prove) formulas containing predicates, quantifiers and the equality symbol.

```
~(exists x: nat, S x = 0) -> forall y:nat, S y <> 0.

forall (f:nat -> nat)
  (forall(x:nat), f (f x) = f x) ->
   exists y:nat, f y = y.

forall P Q : Prop, P -> ~P -> Q.

forall (A:Type)(P: A ->Prop)(Q : Prop),
    (forall x:A, P x -> Q) <->
    ((exists x:A, P x) -> Q).
```

Formulas of First-Order Logic : Terms

We add new construction rules for building propositions.

First, we can build terms according to the declarations of constants and variables, using *Coq*'s typing rules.

```
exp : Z -> Z -> Z.
reverse : list Z -> list Z.
Variable f : Z -> Z.
```

Formulas of First-Order Logic : Terms

We add new construction rules for building propositions.

First, we can build terms according to the declarations of constants and variables, using *Coq*'s typing rules.

```
exp : Z -> Z -> Z.
reverse : list Z -> list Z.
Variable f : Z -> Z.

Check f (exp 2 10).
```

f (expt 2 10) :Z

```
Check reverse (reverse (1::2::3::nil)).
```

reverse (reverse (1::2::3::nil)) : list Z

Predicates

A predicate is just any function whose result type is Prop

```
sorted : list Z -> Prop.
positive : Z -> Prop.
permutation : list Z -> list Z -> Prop.
```

```
Check sorted (1::2::6::4::nil).
```

sorted (1::2::6::4::nil) : Prop

```
Check positive (3 * 3).
```

*positive (3 * 3) : Prop*

```
Check permutation (1::nil) (2::nil). ...
```

It is always possible to declare or define new predicates :

```
Parameter P : nat -> nat -> Prop.
```

```
Definition negative (z:Z) := z <= 0.
```

```
Check fun n : Z => n * n < n + n.
```

*fun n : Z => n * n < n + n : Z -> Prop.*

```
Check fun n: nat => P n n.
```

fun n: nat => P n n : nat -> Prop

Equality

If t_1 and t_2 are terms *of the same type*, then $t_1 = t_2$ is a proposition.

```
Check reverse(reverse(1::2::3::nil) = 1::2::3::nil.
```

reverse(reverse(1::2::3::nil) = 1::2::3::nil
: Prop

```
Check true = 3.
          ^
```

Error: The term "3" has type "nat" while it is expected to have type "bool".

Equality

If t_1 and t_2 are terms *of the same type*, then $t_1 = t_2$ is a proposition.

```
Check reverse(reverse(1::2::3::nil) = 1::2::3::nil.
```

reverse(reverse(1::2::3::nil) = 1::2::3::nil
: Prop

```
Check true = 3.
          ^
```

Error: The term "3" has type "nat" while it is expected to have type "bool".

Check true <> 3. (~ true = 3 *)*
^

Error: The term "3" has type "nat" while it is expected to have type "bool".

Quantifiers

Let F be a proposition and x be a variable, then $\forall$ `x:A,F` and $\exists x : A, F$ are propositions. x is said to be **bound** in F.

ASCII notation : The symbol $\forall$ is typed forall and $\exists$ is typed exists.

Examples

```
Parameter A : Type.
Parameter R : A → A → Prop.
Parameter f : A → A.
Parameter a : A.

Check f (f a).
```

(f (f a)) : A

```
Check R a (f (f a)).
```

R a (f (f a)): Prop

```
Check forall x :A, R a x → R a (f (f (f x))).
```

forall x :A, R a x → R a (f (f (f x))) : Prop.

Introduction tactic for the universal quantifier

This tactic applies to a goal of the form :

```
...
=============
forall x:A, F
```

The tactic intro x transforms this goal into :

```
...
x : A
============
F
```

Note that the variable x must not appear freely in the context.
One can always use intro with a fresh variable.

It is very usual to use intros on nested universal quantifications and implications :

...
====================
forall x :A, P x → forall y: A, R x y → R x (f (f (f y))).

```
intros x Hx y Hy.
```

...
x: A
Hx: P x
y: A
Hy: R x y
=======================
R x (f (f (f y)))

Elimination tactic for the universal quantifier

The tactic apply H solves goals of the form :

```
...
H : forall x:A, P x.
=================
P t   (assuming t:A)
```

Example :

```
H : forall x:Z, 0 <= x * x
==========================
0 <= 3 * 3
apply H.
```

The tactic apply is generalized to the case of nested implications and universal quantifications, like, for instance :

```
H : ∀ x:A, P x → ∀ y:A, R x y → R x (f y)
```

On a goal like R a (f (f a)), the tactic apply H will generate two subgoals : P a and R a (f a).

In fact, the comparison between the goal R a (f (f a)) and the conclusion R x (f y) returns a substitution that maps x to a and y to f a.

A Small Example

```
Hypothesis Hf : forall x y:A, R x y → R x (f y).
Hypothesis R_refl : forall x:A, R x x.

Lemma Lf : forall x :A, R x (f (f (f x))).
Proof.
 intro  x;apply Hf.
```

1 subgoal

Hf : forall x y : A, R x y → R x (f y)
R_refl : forall x : A, R x x
x : A
============================
R x (f (f x))

```
repeat apply Hf.
```

1 subgoal

A : Set
f : A → A
Hf : forall x y : A, R x y → R x (f y)
R_refl : forall x : A, R x x
x : A
============================
R x x

```
apply R_refl.
Qed.
```

Helping apply

Let us use the following theorems from the library `Arith` :

lt_n_Sn : forall n : nat, $n < S\ n$
lt_trans : forall n m p : nat, $n < m \rightarrow m < p \rightarrow n < p$

```
Lemma lt_n_SSn : forall i:nat, i < S (S i).
Proof.
 intro i;apply lt_trans.
```

Error: Unable to find an instance for the variable m.

Helping apply

Let us use the following theorems from the library `Arith` :

lt_n_Sn : forall n : nat, $n < S\ n$
lt_trans : forall n m p : nat, $n < m \rightarrow m < p \rightarrow n < p$

```
Lemma lt_n_SSn : forall i:nat, i < S (S i).
Proof.
 intro i;apply lt_trans.
```

Error: Unable to find an instance for the variable m.

```
 intro i;apply lt_trans with (S i);apply lt_n_Sn.
```

Another possibility : use eapply (see the documentation).
See also the pattern tactic.

Introduction rule for the existential quantifier

$$\frac{\Gamma \vdash F\{x/t\} \quad t : A}{\Gamma \vdash \exists x : A, F}\ \exists_i$$

The associated tactic is exists t.

=========

forall n:nat, exists p:nat, $n < p$.

```
intro n; exists (S n).
```

$n : nat$

================

$n < S\ n$

Elimination rule for the existential quantifier

$$\dfrac{\dfrac{\dots}{\Gamma, x : A, Hx : F \vdash G} \quad \Gamma \vdash \exists x : A,\ F}{\Gamma \vdash G}\ x \text{ not bound in } \Gamma$$

The associated tactic is destruct H as [x Hx], where H $:\exists\, x : A,\ F$ w.r.t. Γ.

H : exists n:nat, forall p: nat, p < n
====================
False

```
destruct H as [n Hn].
```

n : nat
Hn : forall p : nat, p < n
==============================
False

Rules and tactics for the equality

Introduction rule.

$$\frac{a : A}{a = a} \; \mathit{refl\_equal}$$

Associated tactics : reflexivity, trivial, auto.

```
Lemma L36 : 9 * 4 = 3 * 12.
Proof.
 reflexivity.
Qed.
```

The tactic `rewrite`

Let $H : a = b$. the tactic rewrite -> H replaces every occurrence of a by b in the conclusion of the current goal.

The tactic rewrite <- H replaces every occurrence of b by a in the conclusion of the current goal.

Note

The tactic `rewrite` has a quite more complex behaviour, when `H` contains universal quantifiers. Look at the documentation.
See also : tactics symmetry, transitivity, replace, etc.

Example

```
Lemma eq_trans_on_A :
 forall x y z:A,  x = y → y = z → x = z.
Proof.
 intros x y z e.
```

...

$e : x = y$

==============================

$y = z \rightarrow x = z$

```
 rewrite → e.
```

...

$e : x = y$

==============================

$y = z \rightarrow y = z$

Other tactics for equality

- symmetry transforms any goal $t_1 = t_2$ into $t_2 = t_1$
- transitivity t_3 transforms aany goal $t_1 = t_3$ into two subgoals $t_2 = t_3$ and $t_3 = t_2$

See also replace, subst, etc.

rewriting some occurences

```
Lemma L1 : forall x y : nat,
  x = S (S y) -> 2 <= x * x .
intros x y e. rewrite e.
(*
 x : nat
 y : nat
 e : x = S (S y)
 ============================
  2 <= S (S y) * S (S y)
*)
```

```
Undo.
pattern x at 1; rewrite e.
```

1 subgoal

$x : nat$
$y : nat$
$e : x = S\ (S\ y)$
============================
$2 <= S\ (S\ y) * x$

Using function application

Let us consider rewrite again :

```
Variable f : nat -> nat -> nat.
Hypothesis f_comm : forall x y, f x y = f y x.

Lemma L : forall x y z, f (f x y) z = f z (f y x).
intros x y z; rewrite f_comm.
```

1 subgoal

x : nat
y : nat
z : nat
============================
$f\ z\ (f\ x\ y) = f\ z\ (f\ y\ x)$

Using function application

Let us consider rewrite again :

```
Variable f : nat -> nat -> nat.
Hypothesis f_comm : forall x y, f x y = f y x.

Lemma L : forall x y z, f (f x y) z = f z (f y x).
intros x y z; rewrite f_comm.
```

1 subgoal

$x : nat$
$y : nat$
$z : nat$
$============================$
$f\ z\ (f\ x\ y) = f\ z\ (f\ y\ x)$

```
rewrite (f_comm x y); reflexivity.
```

```
Require Import Omega.
Lemma L : forall n:nat, n < 2 -> n = 0 \/ n = 1.
Proof.
 intros;omega.
Qed.

Lemma L2 : forall i:nat, i < 2 -> i*i = i.
Proof.
 intros i H; destruct (L _ H); subst i; trivial.
Qed.
```

Higher Order Predicate Logic

It is possible to quantify over types, functions, predicates ...

```
Lemma or_comm : forall P Q:Prop, P \/ Q -> Q \/ P.
Proof.
 intros P Q H; destruct H; [right | left];assumption.
Qed.

Lemma not_ex_all_not : forall (A:Type)(P:A->Prop),
  (~exists a:A, P a) -> forall a, ~ P a.
Proof.
 intros A P H a Ha; destruct H;exists a;assumption.
Qed.
```

```
Lemma L: exists P:nat->Prop,
   P 0 /\ ~ P 1.
Proof.
```

```
Lemma L: exists P:nat->Prop,
   P 0 /\ ~ P 1.
Proof.
 exists (fun n => n = 0).
```

1 subgoal

============================

0 = 0 /\ 1 <> 0

```
split;[reflexivity|discriminate].
Qed.
```

```
Lemma exf :exists f:nat->nat,
           forall n p, 0 < p -> p <= n  ->
             exists q, f n = q * p.
Proof.
```

```
Lemma exf :exists f:nat->nat,
           forall n p, 0 < p -> p <= n  ->
             exists q, f n = q * p.
Proof.
 exists fact.
```

1 subgoal

============================

forall n p : nat, 0 < p -> p <= n ->
*exists q : nat, fact n = q * p*

...

Qed.

```
Section HO.
 Variable A : Type.
 Variable f : A -> A.
 Hypothesis f_idem : forall a, f (f a) = a.

 Lemma f_onto : forall b, exists a, b= f a.
 Proof.
 intro b; exists (f b); rewrite f_idem; reflexivity.
 Qed.
End HO.

Check f_onto.
```

f_onto
: forall (A : Type) (f : A -> A),
(forall a : A, f (f a) = a) ->
forall b : A, exists a : A, b = f a

A useful tactic

The tactics f_equal breaks a goal of the form
f a b ... x = f a' b' ...x' into the subgoals
a = a', b = b', ..., x = x'

```
Require Import ZArith.
Require Import Ring.
Open Scope Z_scope.
Parameter f : Z -> Z -> Z -> Z.

Goal forall x y z:Z ,
  f (x+y) z 0 = f(y+x+0) (z*(1+0)) (x-x).
intros x y z; f_equal; ring.
Qed.
```

Rewriting a logical equivalence

The tactic rewrite H and its derivates can be used even if H is a logical equivalence.

Note that in some old versions of *Coq*, you have to require a module by Require Import Setoid.

```
Variables (A:Type)(P Q : A -> Prop).
Hypothesis H : forall a:A, P a <-> ~ Q a.

Goal (exists a, P a) -> ~(forall x, Q x).
intros [a Ha] H0.
```

H : forall a : A, P a <-> ~ Q a
a : A
Ha : P a
H0 : forall x : A, Q x
============
False

```
rewrite H in Ha.
```

```
(*
  H : forall a : A, P a <-> ~ Q a
  a : A
  Ha : ~ Q a
  H0 : forall x : A, Q x
  ============================
   False
*)
destruct Ha;apply H0.
Qed.
```