Chapter 1

Library clodomir_tactics

Capítulo 5 -

Tática Apply

```
Theorem silly1 : ∀ (n m o p : nat),
     n = m →
     [n; o] = [n; p] →
     [n; o] = [m; p].
Theorem silly1' : ∀ (n m o p : nat),
     n = m →
     [n; o] = [n; p] →
     [n; o] = [m; p].
Theorem silly2 : ∀ (n m o p : nat),
     n = m →
     (∀ (q r : nat), q = r → [q; o] = [r; p]) →
     [n; o] = [m; p].
Theorem silly2a : ∀ (n m : nat),
     (n, n) = (m, m) →
     (∀ (q r : nat), (q, q) = (r, r) → [q] = [r]) →
     [n] = [m].
```

Exercise: 2 stars, optional (silly_ex)

```
Theorem silly_ex : (∀ n, evenb n = true → oddb (S n) = true) →
     oddb 3 = true →
     evenb 4 = true.
Theorem silly3_firsttry : ∀ (n : nat),
     true = eqb n 5 →
     eqb (S (S n)) 7 = true.
```

Exercise: 3 stars (apply_exercise1)

```
Theorem rev_exercise1 : ∀ (l l' : list nat),
     l = rev l' →
     l' = rev l.
```

```
Example trans_eq_example : ∀ (a b c d e f : nat),
     [a; b] = [c; d] →
     [c; d] = [e; f] →
     [a; b] = [e; f].
```

```
Theorem trans_eq : ∀ (X : Type) (n m o : X),
  n = m → m = o → n = o.
```

```
Example trans_eq_example' : ∀ (a b c d e f : nat),
     [a; b] = [c; d] →
     [c; d] = [e; f] →
     [a; b] = [e; f].
```

Exercise: 3 stars, optional (apply_with_exercise)

```
Example trans_eq_exercise : ∀ (n m o p : nat),
     m = (minustwo o) →
     (n + p) = m →
     (n + p) = (minustwo o).
```

```
Theorem S_injective : ∀ (n m : nat),
  S n = S m →
  n = m.
```

```
Theorem S_injective' : ∀ (n m : nat),
  S n = S m →
  n = m.
```

```
Theorem S_injective'' : ∀ (n m : nat),
  S n = S m →
  n = m.
```

```
Theorem injection_ex1 : ∀ (n m o : nat),
  [n; m] = [o; o] →
  [n] = [m].
```

```
Theorem injection_ex1' : ∀ (n m o : nat),
  [n; m] = [o; o] →
  [n] = [m].
```

```
Theorem injection_ex2 : ∀ (n m : nat),
  [n] = [m] →
  n = m.
```

Exercise: 1 star (inversion_ex3)

```
Example inversion_ex3 : ∀ (X : Type) (x y z w : X) (l j : list X),
  x :: y :: l = w :: z :: j →
  x :: l = z :: j →
  x = y.
```

```
Theorem eqb_0_l : ∀ n,
   eqb 0 n = true → n = 0.
```

```
Theorem eqb_0_l' : ∀ n,
   eqb 0 n = true → n = 0.
```

```
Theorem discriminate_ex1 : ∀ (n : nat),
  S n = O →
  2 + 2 = 5.
```

```
Theorem discriminate_ex1' : ∀ (n : nat),
  S n = O →
  2 + 2 = 5.
```

```
Theorem discriminate_ex2 : ∀ (n m : nat),
  false = true →
  [n] = [m].
```

```
Theorem discriminate_ex2' : ∀ (n m : nat),
  false = true →
  [n] = [m].
```

Exercise: 1 star, standard (discriminate_ex3) `Example discriminate_ex3 : ∀ (X : Type) (x y z : X) (l j : list X),`

```
    x :: y :: l = [] →
    x = z.
```

```
Theorem f_equal : ∀ (A B : Type) (f : A → B ) (x y: A),
x = y → f x = f y.
```

```
Theorem S_inj : ∀ (n m : nat) (b : bool),
     eqb (S n) (S m) = b →
     eqb n m = b.
```

```
Theorem silly3' : ∀ (n : nat),
  (eqb n 5 = true → eqb (S (S n)) 7 = true) →
  true = eqb n 5 →
  true = eqb (S (S n)) 7.
```

Exercise: 3 stars, recommended (plus_n_n_injective) `Theorem plus_n_n_injective : ∀ n m,`

```
     n + n = m + m →
     n = m.
```

```
Theorem double_injective : ∀ n m,
     double n = double m →
     n = m.
```

Exercise: 2 stars (beq_nat_true) `Theorem eqb_true : ∀ n m,`

```
    eqb n m = true → n = m.
```

```
Theorem double_injective_take2 : ∀ n m,
     double n = double m →
     n = m.
```

```
Theorem eqb_id_true : ∀ x y,
  eqb_id x y = true → x = y.
```

Exercise: 3 stars, recommended (gen_dep_practice)

Theorem nth_error_after_last: forall (n : nat) (X : Type) (l : list X), length l = n -> xth_error l n = None. Proof. intros n X l H. generalize dependent n. induction l as |*m' l' IHl.*

- reflexivity.

- intros n H. destruct n as |*n'.* + inversion H. + simpl. inversion H. apply IHl. reflexivity.

Qed.

```
Definition square n := n × n.
```

```
Lemma square_mult : ∀ n m, square (n × m) = square n × square m.
```

```
Definition foo (x: nat) := 5.
```

```
Fact silly_fact_1 : ∀ m, foo m + 1 = foo (m + 1) + 1.
```

```
Definition bar x :=
  match x with
  | O ⇒ 5
  | S _ ⇒ 5
  end.
```

```
Fact silly_fact_2 : ∀ m, bar m + 1 = bar (m + 1) + 1.
```

```
Fact silly_fact_2' : ∀ m, bar m + 1 = bar (m + 1) + 1.
```

```
Definition sillyfun (n : nat) : bool :=
  if eqb n 3 then false
  else if eqb n 5 then false
  else false.
```

```
Theorem sillyfun_false : ∀ (n : nat),
  sillyfun n = false.
```

```
Fixpoint split {X Y : Type} (l : list (X×Y))
               : (list X) × (list Y) :=
  match l with
  | [] ⇒ ([], [])
  | (x, y) :: t ⇒
      match split t with
      | (lx, ly) ⇒ (x :: lx, y :: ly)
      end
  end.
```

```
Theorem combine_split : ∀ X Y (l : list (X × Y)) l1 l2,
  split l = (l1, l2) →
  combine l1 l2 = l.
```

```
Definition sillyfun1 (n : nat) : bool :=
  if eqb n 3 then true
  else if eqb n 5 then true
  else false.
```

```
Theorem sillyfun1_odd : ∀ (n : nat),
     sillyfun1 n = true →
     oddb n = true.
```

Exercise: 2 stars (destruct_eqn_practice)

```
Theorem bool_fn_applied_thrice :
  ∀ (f : bool → bool) (b : bool),
  f (f (f b)) = f b.
```

1.1 Review

We've now seen many of Coq's most fundamental tactics. We'll introduce a few more in the coming chapters, and later on we'll see some more powerful *automation* tactics that make Coq help us with low-level details. But basically we've got what we need to get work done.

Here are the ones we've seen:

- `intros`: move hypotheses/variables from goal to context
- `reflexivity`: finish the proof (when the goal looks like $e = e$)
- `apply`: prove goal using a hypothesis, lemma, or constructor
- `apply`... `in` H: apply a hypothesis, lemma, or constructor to a hypothesis in the context (forward reasoning)
- `apply`... `with`...: explicitly specify values for variables that cannot be determined by pattern matching
- `simpl`: simplify computations in the goal
- `simpl in` H: ... or a hypothesis
- `rewrite`: use an equality hypothesis (or lemma) to rewrite the goal
- `rewrite` ... `in` H: ... or a hypothesis

- `symmetry`: changes a goal of the form $t{=}u$ into $u{=}t$
- `symmetry in` H: changes a hypothesis of the form $t{=}u$ into $u{=}t$
- `unfold`: replace a defined constant by its right-hand side in the goal
- `unfold... in` H: ... or a hypothesis
- `destruct... as...`: case analysis on values of inductively defined types
- `destruct...` *eqn*:...: specify the name of an equation to be added to the context, recording the result of the case analysis
- `induction... as...`: induction on values of inductively defined types
- `inversion`: reason by injectivity and distinctness of constructors
- `assert` (H: e) (or `assert` (e) `as` H): introduce a "local lemma" e and call it H
- `generalize dependent` x: move the variable x (and anything else that depends on it) from the context back to an explicit hypothesis in the goal formula

Exercise: 3 stars (eqb_sym)

```
Theorem eqb_sym : ∀ (n m : nat),
  eqb n m = eqb m n.
```

Exercise: 3 stars, optional (beq_nat_trans)

```
Theorem eqb_trans : ∀ n m p,
  eqb n m = true →
  eqb m p = true →
  eqb n p = true.
```

Exercise: 3 stars, advanced (split_combine)

```
Check @list (nat×nat).

Definition split_combine_statement : Prop :=
  ∀ (X Y : Type) (l1 : list X) (l2 : list Y),
    length l1 = length l2 →
    split (combine l1 l2) = (l1, l2).

Theorem split_combine : split_combine_statement.
```

Exercise: 3 stars, advanced (filter_exercise)

```
Theorem filter_exercise : ∀ (X : Type) (test
: X → bool)
                             (x : X) (l lf : list X),
     filter test l = x :: lf →
     test x = true.

Fixpoint forallb {X : Type} (f : X → bool) (l : list X) :=
  match l with
  | nil ⇒ true
  | h :: t ⇒ andb (f h) (forallb f t)
  end.
Fixpoint existsb {X : Type} (f : X → bool) (l : list X) :=
  match l with
  | nil ⇒ false
  | h :: t ⇒ orb (f h) (existsb f t)
  end.
Definition existsb' {X : Type} (f : X → bool) (l : list X) : bool :=
  negb (forallb (fun x ⇒ negb (f x)) l).
Example forallb1 : forallb oddb [1; 3; 5; 7; 9] = true.
Example forallb2 : forallb negb [false; false] = true.
Example forallb3 : forallb evenb [0; 2; 4; 5] = false.
Example forallb4 : forallb (eqb 5) [] = true.
Example existsb1 : existsb (eqb 5) [0; 2; 3; 6] = false.
Example existsb2 : existsb (andb true) [true; true; false] = true.
Example existsb3 : existsb oddb [1; 0; 0; 0; 0; 3] = true.
Example existsb4 : existsb evenb [] = false.
Example existsb'1 : existsb' (eqb 5) [0; 2; 3; 6] = false.
Example existsb'2 : existsb' (andb true) [true; true; false] = true.
Example existsb'3 : existsb' oddb [1; 0; 0; 0; 0; 3] = true.
Example existsb'4 : existsb' evenb [] = false.
```

Theorem existsb_existsb' : forall (X : Type) (f : X -> bool) (l : list X), existsb f l = existsb' f l. Proof. intros X f l. induction l as |*n l' IHl*.

- reflexivity.
- destruct (f n) eqn:Hfn1. + unfold existsb'. simpl. rewrite -> Hfn1. reflexivity. + assert (H : existsb' f (n :: l') = f n \/ existsb' f l'). { rewrite -> Hfn1. unfold existsb'. simpl. rewrite -> Hfn1. reflexivity. } rewrite -> H. rewrite <- IHl. reflexivity.

Qed.