课后阅读：Coq 标准库中的自然数

在 Coq 中，许多数学上的集合可以用归纳类型定义。例如，Coq 中自然数的定义就是最简单的归纳类型之一。下面 Coq 代码可以用于查看 `nat` 在 Coq 中的定义。

```
Print nat.
```

查询结果如下：

```
Inductive nat := O : nat | S: nat -> nat.
```

可以看到，自然数集合的归纳定义可以看做 `tree` 退化成为 `list` ，再从 `list` 进一步退化的结果。下面我们将以自然数的加法为代表，介绍 Coq 标准库自然数相关函数的定义方式，我们还会试着证明一条加法的基本性质：加法交换律。

先定义自然数 `nat` 。

```
Inductive nat :=
| O: nat
| S (n: nat): nat.
```

这里 `O` 表示零，`S n` 表示自然数 `n` 的后继，即 `n + 1` 。下面再定义自然数加法运算。

```
Fixpoint add (n m: nat): nat :=
  match n with
  | O => m
  | S n' => S (add n' m)
  end.
```

下面开始试着证明加法交换律。

```
Theorem add_comm:
  forall n m, add n m = add m n.
Proof.
  intros.
  induction n; simpl.
  (** 奠基步骤需要证明_[add O m = add m O]_。根据_[add]_的定义，_[add O m = m]_，因
      此我们需要先证明_[add m O = m]_这条性质。*)
Abort.
```

下面是引理的证明

```
Lemma add_0_r: forall n, add n 0 = n.
Proof.
  intros.
  induction n; simpl.
  + reflexivity.
  + rewrite IHn.
    reflexivity.
Qed.
```

引理的证明结束后，可以继续加法交换律的证明。

```
Theorem add_comm: forall n m,
  add n m = add m n.
Proof.
  intros.
  induction n; simpl.
  + rewrite add_0_r.
    reflexivity.
  + (** 归纳步骤需要证明关于_[add (S n) m = add m (S n)]_，而根据_[add]_的定义，
        _[add (S n) m = S (add n m)]_。因此，只需要先证明引理
        _[add m (S n) = S (add m n)]_，就可以由归纳假设_[add n m = add m n]_推出要
        证明的结论了。*)
Abort.
```

下面是引理的证明。

```
Lemma add_succ_r: forall n m,
  add n (S m) = S (add n m).
Proof.
  intros.
  induction n; simpl.
  + reflexivity.
  + rewrite IHn.
    reflexivity.
Qed.
```

到这里，我们可以在 Coq 中完成加法交换律的证明了。

```
Theorem add_comm: forall n m,
  add n m = add m n.
Proof.
  intros.
  induction n; simpl.
  + rewrite add_0_r.
    reflexivity.
  + rewrite add_succ_r.
    rewrite IHn.
    reflexivity.
Qed.
```

由于自然数范围内，数学意义上的减法是一个部分函数，因此，相关定义在 Coq 中并不常用。相对而言，自然数的加法与乘法在 Coq 中更常用。

```
Fixpoint mul (n m: nat): nat :=
  match n with
  | O => O
  | S p => add m (mul p m)
  end.
```

下面列举加法与乘法的其它重要性质。

```
Theorem add_assoc:
  forall n m p, add n (add m p) = add (add n m) p.
```

```
Theorem add_cancel_l:
  forall n m p, add p n = add p m <-> n = m.
```

```
Theorem add_cancel_r:
  forall n m p, add n p = add m p <-> n = m.
```

```
Lemma mul_0_r: forall n, mul n O = O.
```

```
Lemma mul_succ_r:
  forall n m, mul n (S m) = add (mul n m) n.
```

```
Theorem mul_comm:
  forall n m, mul n m = mul m n.
```

```
Theorem mul_add_distr_r:
  forall n m p, mul (add n m) p = add (mul n p) (mul m p).
```

```
Theorem mul_add_distr_l:
  forall n m p, mul n (add m p) = add (mul n m) (mul n p).
```

```
Theorem mul_assoc:
  forall n m p, mul n (mul m p) = mul (mul n m) p.
```

```
Theorem mul_1_l: forall n, mul (S O) n = n.
```

```
Theorem mul_1_r: forall n, mul n (S O) = n.
```

前面已经提到，Coq 在自然数集合上不便于表达减法等运算，因此，Coq 用户有些时候可以选用 `Z` 而非 `nat` 。然而，由于其便于表示计数概念以及表述数学归纳法，`nat` 依然有许多用途。例如，Coq 标准库中的 `Nat.iter` 就表示函数多次迭代，具体而言，`Nat.iter n f` 表示将函数 `f` 迭代 `n` 次的结果。其 Coq 定义如下：

```
Fixpoint iter {A: Type} (n: nat) (f: A -> A) (x: A): A :=
  match n with
  | O => x
  | S n' => f (iter n' f x)
  end.
```

它符合许多重要性质，例如：

```
Theorem iter_S:
  forall {A: Type} (n: nat) (f: A -> A) (x: A),
    Nat.iter n f (f x) = Nat.iter (S n) f x.
```

注意，哪怕是如此简单的性质，我们还是需要在 Coq 中使用归纳法证明。

```
Proof.
  intros.
  induction n; simpl.
  + reflexivity.
  + rewrite IHn; simpl.
    reflexivity.
Qed.
```

习题 1. 请证明下面关于 `Nat.iter` 的性质。

```
Theorem iter_add:
  forall {A: Type} (n m: nat) (f: A -> A) (x: A),
    Nat.iter (n + m) f x = Nat.iter n f (Nat.iter m f x).
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 2. 请证明下面关于 `Nat.iter` 的性质。

```
Theorem iter_mul:
  forall {A: Type} (n m: nat) (f: A -> A) (x: A),
    Nat.iter (n * m) f x = Nat.iter n (Nat.iter m f) x.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```