Coq 标准库中的自然数

1 用归纳法证明自然数性质

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、乘法与带余除法在 Coq 中更常用。自然数乘法可以基于加法定义。

```
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。 *)
```

2 补充内容：Coq 中的跨步归纳法

如果要定义自然数中的“除以二取整”这个运算，则很自然可以在 Coq 中写出如下定义：

```
Fixpoint div2 (n: nat): nat :=
  match n with
  | O => O
  | S n' => match n' with
            | O => O
            | S n'' => S (div2 n'')
            end
  end.
```

该定义就是 Coq 标准库中的 `Nat.div2` 。可以看出，`Nat.div2` 是一个结构递归函数，不过，与其他简单递归函数不同，该定义并不是将 `Nat.div2 (S n)` 的定义规约为 `Nat.div2 n` ，而是将 `Nat.div2 (S (S n))` 的定义跨两步规约为 `Nat.div2 n` 。Coq 也是允许这样的结构递归定义的。下面我们证明几条 `Nat.div2` 的基本性

质。我们首先证明 `Nat.div2` 的“跨两步递归等式”：`div2_succ_succ` 。这一性质可以直接基于定义用 `simpl` 指令证明。

```
Lemma div2_succ_succ: forall n, Nat.div2 (S (S n)) = S (Nat.div2 n).
Proof. intros. simpl. reflexivity. Qed.
```

其次，`2*n` 除以二取整会得到 `n` 。很自然，该性质可以通过对 `n` 归纳完成证明。

```
Theorem div2_double: forall n, Nat.div2 (2 * n) = n.
Proof.
  intros.
  induction n.
  + (** 奠基步骤需要我们证明_[Nat.div2 (2 * 0) = 0]_，这是显然的。*)
    simpl.
    reflexivity.
  + (** 归纳步骤需要我们证明_[Nat.div2 (2 * S n) = S n]_，我们可以作下面代数变换：
            2 * (S n) = 2 * n + 2 = 2 + 2 * n = S (S (2 * n))
        从而便于我们进一步化简_[Nat.div2]_的计算结果，并使用归纳假设。*)
    rewrite Nat.mul_succ_r.
    rewrite Nat.add_comm.
    unfold Nat.add.
    (** 现在，我们只需证明_[Nat.div2 (S (S (2 * n))) = S n]_了*)
    rewrite div2_succ_succ.
    rewrite IHn.
    reflexivity.
Qed.
```

类似的，`S (2 * n)` 除以二取整也会得到 `n` 本身。

```
Theorem div2_succ_double: forall n, Nat.div2 (S (2 * n)) = n.
(* 证明详见Coq源代码。 *)
```

上面我们证明了，先乘二再除以二取整能得到原数，先乘二加一再除以二取整也能得到原数。下面我们证明一个反方向的结论，将一个自然数先除以二取整，再乘二或者再乘二加一，同样能得到原数。

```
Theorem double_div2:
  forall n,
    2 * Nat.div2 n = n \/
    S (2 * Nat.div2 n) = n.
```

与前面的其他性质相比，该性质并不容易证明。假如试图套用归纳法作证明，那么在归纳步骤中就需要从以下归纳假设

```
        2 * Nat.div2 n = n \/
        S (2 * Nat.div2 n) = n
```

推出以下结论

```
        2 * Nat.div2 (S n) = n \/
        S (2 * Nat.div2 (S n)) = n
```

然而，根据 `Nat.div2` 的定义，我们并不能对 `Nat.div2 (S n)` 化简，进而使用归纳假设。此处的问题在于，`Nat.div2` 的定义是跨两步递归的，因此，我们事实上需要一种跨两步的归纳证明方法。在 Coq 中，一般通过如下方法实现跨两步的归纳证明。

如果我们需要跨两步归纳证明的性质是 `forall n, P n` ，那么我们就在 Coq 中归纳证明

```
P n /\ P (S n)
```

这样一来，名义上的奠基步骤就是要证明

```
P 0 /\ P 1
```

而名义上的归纳步骤就是要证明

```
P n /\ P (S n) -> P (S n) /\ P (S (S n))
```

由于其中的前提与结论都包含 `P (S n)` ，所以，这就等价于要证明

```
P n /\ P (S n) -> P (S (S n))
```

下面我们就用这种跨两步归纳法来完成 `double_div2` 的证明。

```
Proof.
  intros.
  (** 首先改为证明加强后的命题，这可以通过_[assert]_指令完成。
      加强后的命题显然可以推出原命题，这可以通过_[tauto]_指令完成。*)
  assert ((2 * Nat.div2 n = n \/
           S (2 * Nat.div2 n) = n) /\
          (2 * Nat.div2 (S n) = S n \/
           S (2 * Nat.div2 (S n)) = S n)); [| tauto].
  induction n.
  + (** 这是名义上的奠基步骤，它需要我们证明原命题对0和1成立。*)
    split; simpl.
    - left.
      reflexivity.
    - right.
      reflexivity.
  + (** 这是名义上的归纳步骤，其结论的左半边是名义归纳假设的右半边，直接_[tauto]_求解。*)
    split; [tauto |].
    (** 由于我们只需要由_[n]_推_[S (S n)]_，所以我们只保留名义归纳假设_[IHn]_的左
        半边，供后续证明使用。*)
    destruct IHn as [IHn _].
    (** 至此，我们只需要由归纳假设
            2 * Nat.div2 n = n \/
            S (2 * Nat.div2 n) = n
        证明下面结论即可：
            2 * Nat.div2 (S (S n)) = S (S n) \/
            S (2 * Nat.div2 (S (S n))) = S (S n)。
        这利用下面代数变化不难完成：
            2 * Nat.div2 (S (S n)) =
            2 * (S (Nat.div2 n)) =
            S (S (2 * (Nat.div2 n)) *)
    simpl Nat.div2.
    rewrite Nat.mul_succ_r, Nat.add_comm.
    unfold Nat.add.
```

```
    (** 我们依计划做代数变化后，现在只需证明
            S (S (2 * Nat.div2 n)) = S (S n) \/
            S (S (S (2 * Nat.div2 n))) = S (S n) *)
    destruct IHn as [IHn | IHn].
    - left.
      (** 如果归纳假设的第一个子句成立，那么结论中也是第一个子句成立，现在只需由
            IHn: 2 * Nat.div2 n = n
          推出
            S (S (2 * Nat.div2 n)) = S (S n) *)
      rewrite IHn.
      reflexivity.
    - right.
      (** 如果归纳假设的第二个子句成立，那么结论中也是第二个子句成立，现在只需由
            IHn: S (2 * Nat.div2 n) = n
          推出
            S (S (S (2 * Nat.div2 n))) = S (S n) *)
      rewrite IHn.
      reflexivity.
Qed.
```

习题 3. 请证明以下关于 `Nat.div2` 的结论。

```
Theorem double_div2':
  forall n, Nat.div2 n + Nat.div2 (S n) = n.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```