课后阅读：高阶函数与高阶谓词

1 高阶函数

在 Coq 中，函数的参数也可以是函数。下面定义的 `shift_left1` 是一个二元函数，它的第一个参数 `f: Z -> Z` 是一个从整数到整数的一元函数，第二个参数 `x: Z` 是一个整数，这个函数依据两个参数计算出的函数值是 `f (x + 1)` 。

```
Definition shift_left1 (f: Z -> Z) (x: Z): Z :=
  f (x + 1).
```

下面是两个 `shift_left1` 的例子。

```
Example shift_left1_square: forall x,
  shift_left1 square x = (x + 1) * (x + 1).
Proof. unfold shift_left1, square. lia. Qed.
```

```
Example shift_left1_plus_one: forall x,
  shift_left1 plus_one x = x + 2.
Proof. unfold shift_left1, plus_one. lia. Qed.
```

从这两个例子可以看出，`shift_left1` 作为一个二元函数，也可以被看作一个一元函数，这个一元函数的参数是一个整数到整数的函数，这个一元函数的计算结果也是一个整数到整数的函数。例如，`shift_left1 square` 的计算结果在数学上可以写作函数

$$f(x) = (x+1)^2。$$

更一般的，`shift_left1 f` 可以看作将函数 `f` 在坐标平面中的图像左移一个单位的结果。这样的观点也可以在 Coq 中更直白地表达出来，下面定义的 `shift_left1'` 实际上是与 `shift_left1` 完全相同的函数，但是这一定义在形式上是一个一元函数。

```
Definition shift_left1' (f: Z -> Z): Z -> Z :=
  fun x => f (x + 1).
```

Coq 表达式 1. 匿名函数与保留字 `fun` . Coq 中可以使用保留字 `fun` 表示匿名函数。例如，`fun x: Z => x + 10` 表示这样的一个匿名函数，它接收一个整数参数，如果这个参数的值为 `x` ，那么这个函数的计算结果是 `x + 10` 。当参数的类型可以自动推断得出时，上面参数的类型也可以省略。匿名函数也可以是多元函数，例如 `fun (f: Z -> Z) (x: Z) => f (x + 1)` 。Coq 中的匿名函数语法取自于著名的 lambda 表达式。

与 `shift_left1` 类似，我们还可以定义将一元函数图像向上移动一个单位的结果。

```
Definition shift_up1 (f: Z -> Z) (x: Z): Z :=
  f x + 1.
```

```
Example shift_up1_square: forall x,
  shift_up1 square x = x * x + 1.
Proof. unfold shift_up1, square. lia. Qed.
```

```
Example shift_up1_plus_one: forall x,
  shift_up1 plus_one x = x + 2.
Proof. unfold shift_up1, plus_one. lia. Qed.
```

像 `shift_left1` 和 `shift_up1` 这样以函数为参数的函数称为高阶函数。高阶函数也可以是多元函数。例如下面的 `func_plus` 与 `func_mult` 定义了函数的加法和乘法。

```
Definition func_plus (f g: Z -> Z): Z -> Z :=
  fun x => f x + g x.
```

```
Definition func_mult (f g: Z -> Z): Z -> Z :=
  fun x => f x * g x.
```

下面证明几条关于高阶函数的简单性质。首先我们证明，对于任意函数 `f` ，对它先左移再上移和先上移再左移的结果是一样的。

```
Lemma shift_up1_shift_left1_comm: forall f,
  shift_up1 (shift_left1 f) = shift_left1 (shift_up1 f).
Proof.
  intros.
  unfold shift_left1, shift_up1.
  reflexivity.
Qed.
```

在上面证明中，在展开“左移”与“上移”的定义后，需要证明的结论是：

```
(fun x : Z => f (x + 1) + 1) = (fun x : Z => f (x + 1) + 1)
```

这个等式两边的表达式字面上完全相同，因此该结论显然成立。证明指令 `reflexivity` 表示利用“自反性”完成证明，所谓等式的自反性就是“任意一个数学对象都等于它自身”。下面我们还可以证明，将 `f` 与 `g` 两个函数先相加再图像左移，与先图像左移再函数相加得到的结果是一样的。

```
Lemma shift_left1_func_plus: forall f g,
  shift_left1 (func_plus f g) =
  func_plus (shift_left1 f) (shift_left1 g).
Proof.
  intros.
  unfold shift_left1, func_plus.
  reflexivity.
Qed.
```

习题 1. 请证明下面命题。

```
Fact shift_up1_eq: forall f,
  shift_up1 f = func_plus f (fun x => 1).
Proof.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

2 高阶谓词

类似于高阶函数，如果一个谓词的参数中有函数（或谓词），那它就是一个高阶谓词。其实，许多常见数学概念都是高阶谓词。例如，函数的单调性就是一个高阶谓词。

```
Definition mono (f: Z -> Z): Prop :=
  forall n m, n <= m -> f n <= f m.
```

有许多函数都是单调的。前面我们定义的 `plus_one` 函数就是个单调函数。

```
Example plus_one_mono: mono plus_one.
Proof.
  unfold mono, plus_one.
  intros.
  lia.
Qed.
```

我们还可以定义整数函数的复合，然后证明复合函数能保持单调性。

```
Definition Zcomp (f g: Z -> Z): Z -> Z :=
  fun x => f (g x).
```

```
Lemma mono_compose: forall f g,
  mono f ->
  mono g ->
  mono (Zcomp f g).
Proof.
  unfold mono, Zcomp.
  intros f g Hf Hg n m Hnm.
  pose proof Hg n m Hnm as Hgnm.
  pose proof Hf (g n) (g m) Hgnm.
  lia.
Qed.
```

习题 2. 请证明常值函数都是单调的。

```
Lemma const_mono: forall a: Z,
  mono (fun x => a).
Proof.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 3. 请证明立方函数是单调的。

```
Example cube_mono: mono (fun x => x * x * x).
Proof.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 4. 请证明函数加法能保持单调性。

```
Lemma mono_func_plus: forall f g,
  mono f ->
  mono g ->
  mono (func_plus f g).
Proof.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

“结合律”也是一个常用的数学概念。如下面定义所示，一个二元函数 `f` 具有结合律，当且仅当它满足 `f x (f y z) = f (f x y) z` 。

```
Definition assoc (f: Z -> Z -> Z): Prop :=
  forall x y z,
    f x (f y z) = f (f x y) z.
```

我们熟知整数的加法于乘法都满足结合律。下面是加法结合律与乘法结合律的 Coq 证明。

```
Lemma plus_assoc: assoc (fun x y => x + y).
Proof. unfold assoc. lia. Qed.
```

```
Lemma mult_assoc: assoc (fun x y => x * y).
Proof. unfold assoc. nia. Qed.
```

习题 5. 请证明，我们先前定义的 `smul` 函数也符合结合律。

```
Lemma smul_assoc: assoc smul.
Proof.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

上面例子中的两个高阶谓词都是以函数为参数的高阶谓词，下面两个例子都是以谓词为参数的高阶谓词，甚至它们的参数本身也是高阶谓词。它们分别说的是，函数的性质 `P` 能被图像上移变换（`shift_up1` ）保持以及能被图像左移变换（`shift_left1` ）保持。

```
Definition preserved_by_shifting_up (P: (Z -> Z) -> Prop): Prop :=
  forall f, P f -> P (shift_up1 f).
```

```
Definition preserved_by_shifting_left (P: (Z -> Z) -> Prop): Prop :=
  forall f, P f -> P (shift_left1 f).
```

不难发现，单调性就能被这两种图像平移变换保持。

```
Lemma mono_pu: preserved_by_shifting_up mono.
(* 证明详见Coq源代码。 *)
```

```
Lemma mono_pl: preserved_by_shifting_left mono.
(* 证明详见Coq源代码。 *)
```

习题 6. 请证明“恒为非负”这一性质（见下面 Coq 定义）也被图像上移与图像左移变换保持。

```
Definition univ_nonneg {A: Type} (f: A -> Z): Prop :=
  forall a: A, f a >= 0.
```

```
Lemma univ_nonneg_pu: preserved_by_shifting_up univ_nonneg.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

```
Lemma univ_nonneg_pl: preserved_by_shifting_left univ_nonneg.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

在数学证明中，我们往往会先证明一些具有一般性的定理或引理，并后续的证明中使用这些定理或引理。站在编写 Coq 证明脚本的角度看，使用先前已经证明的定理也可以看作对定理本身证明代码的复用。反过来，如果一系列数学对象具有相似的性质，这些性质的证明方式上也是相似的，那么我们在 Coq 编码时就可以利用函数与谓词（特别是高阶函数和高阶谓词）概括性地描述这些数学对象之间的关系，并给出统一的证明。例如，当我们要证明

$$f(x) = x^3 - 3x^2 + 3x$$

具备单调性时，可以基于下面代数变换并多次使用复合函数保持单调性这一引理完成证明。

$$f(x) = x^3 - 3x^2 + 3x = (x-1)^3 + 1$$

```
Example mono_ex1: mono (fun x => x * x * x - 3 * x * x + 3 * x).
```

具体而言，下面在 Coq 中证明这一单调性性质的时候，将分两步进行。第一步先证明 `fun x => x - 1` 、`fun x => x + 1` 与 `fun x => x * x * x` 这三个简单函数都是单调的；第二步则利用复合函数的单调性性质将这三个结论组合起来完成证明。

```
Proof.
  assert (mono (fun x => x - 1)) as H_minus.
  (** 这里的_[assert]_指令表示：声明_[mono (fun x => x - 1)]_这一命题成立。逻辑上
      看，一个合理的逻辑系统不应当允许我们凭空声明一条性质。因此，我们在完成声明后需要
      首先证明“为什么这一命题成立”，再基于此进行后续证明。在执行完这一条指令后，Coq系
      统显示目前有两个证明目标需要证明，它们分别对应这里所说的“为什么这一命题成立”与后
      续证明两部分。*)
  { unfold mono. lia. }
  (** 在有多个证明目标需要证明的时候，证明脚本中的左大括号表示进入其中的第一个证明目标
      的证明。这里的第一个证明目标就是要证明_[fun x => x - 1]_是一个单调函数。该证明
      结束后，证明脚本中的右大括号表示返回其余证明目标。返回后可以看到，证明目标中增加
      了一条前提：_[H_minus: fun x => x - 1]_。这一前提的名称_[H_minus]_是先前的
      _[assert]_指令选定的。*)
  assert (mono (fun x => x + 1)) as H_plus.
  { unfold mono. lia. }
  (** 类似的，这里可以使用声明+证明的方式证明_[fun x => x + 1]_也是一个单调函数。*)
  pose proof cube_mono as H_cube.
  (** 先前已经证明过，立方函数是单调的，这里直接_[pose proof]_这一结论。至此，我们已
      经完成了三个子命题的证明，下面的将通过复合函数的形式把它们组合起来。*)
  pose proof mono_compose _ _ H_plus (mono_compose _ _ H_cube H_minus).
  unfold mono.
  intros n m Hnm.
  unfold mono, Zcomp, plus_one in H.
  pose proof H n m Hnm.
  nia.
Qed.
```

Coq 证明脚本 1. assert 指令. 如果 `P` 是一个 Coq 命题，那么 `assert(P)` 指令可以将当前证明目标规约为两个目标：其一是用当前的前提推导 `P` ；其二是在使用当前的前提与 `P` 共同推导当前的结论。如果要对新增的前提 `P` 手动命名，可以采用形如 `assert(P) as H99` 的指令；如果交由 Coq 系统自动命名，它的名称将是 `H` 、`H0` 、`H1` ... 中第一个可以使用的名字。

习题 7. 请证明下面 Coq 命题。

```
Example mono_ex2: mono (fun x => x * x * x + 3 * x * x + 3 * x).
Proof.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```