课后阅读：Coq 中的归纳类型

下面 Coq 代码定义了节点上有整数标号的二叉树。

```
Inductive tree: Type :=
| Leaf: tree
| Node (l: tree) (v: Z) (r: tree): tree.
```

这个定义说的是，一棵二叉树要么是一棵空树 `Leaf` ，要么有一棵左子树、有一棵右子树外加有一个根节点整数标号（`Node` 的情况）。我们可以在 Coq 中写出一些具体的二叉树的例子。

```
Definition tree_example0: tree :=
  Node Leaf 1 Leaf.
```

```
Definition tree_example1: tree :=
  Node (Node Leaf 0 Leaf) 2 Leaf.
```

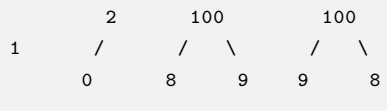
```
Definition tree_example2a: tree :=
  Node (Node Leaf 8 Leaf) 100 (Node Leaf 9 Leaf).
```

```
Definition tree_example2b: tree :=
  Node (Node Leaf 9 Leaf) 100 (Node Leaf 8 Leaf).
```

```
Definition tree_example3a: tree :=
  Node (Node Leaf 3 Leaf) 5 tree_example2a.
```

```
Definition tree_example3b: tree :=
  Node tree_example2b 5 (Node Leaf 3 Leaf).
```

它们分别表示下面这些树结构：



Coq 中，我们往往可以使用递归函数定义归纳类型元素的性质。Coq 中定义递归函数时使用的关键字是 `Fixpoint` 。下面两个定义通过递归定义了二叉树的高度和节点个数。

```
Fixpoint tree_height (t: tree): Z :=
  match t with
  | Leaf => 0
  | Node l v r => Z.max (tree_height l) (tree_height r) + 1
  end.
```

```
Fixpoint tree_size (t: tree): Z :=
  match t with
  | Leaf => 0
  | Node l v r => tree_size l + tree_size r + 1
  end.
```

Coq 系统“知道”每一棵特定树的高度和节点个数是多少。下面是一些 Coq 代码的例子。

```
Example Leaf_height:
  tree_height Leaf = 0.
Proof. reflexivity. Qed.
```

```
Example tree_example2a_height:
  tree_height tree_example2a = 2.
Proof. reflexivity. Qed.
```

```
Example treeexample3b_size:
  tree_size tree_example3b = 5.
Proof. reflexivity. Qed.
```

Coq 中也可以定义树到树的函数。下面的 `tree_reverse` 函数把二叉树进行了左右翻转。

```
Fixpoint tree_reverse (t: tree): tree :=
  match t with
  | Leaf => Leaf
  | Node l v r => Node (tree_reverse r) v (tree_reverse l)
  end.
```

下面是三个二叉树左右翻转的例子：

```
Example Leaf_tree_reverse:
  tree_reverse Leaf = Leaf.
Proof. reflexivity. Qed.
```

```
Example tree_example0_tree_reverse:
  tree_reverse tree_example0 = tree_example0.
Proof. reflexivity. Qed.
```

```
Example tree_example3_tree_reverse:
  tree_reverse tree_example3a = tree_example3b.
Proof. reflexivity. Qed.
```

归纳类型有几条基本性质。(1) 归纳定义规定了一种分类方法，以 `tree` 类型为例，一棵二叉树 `t` 要么是 `Leaf` ，要么具有形式 `Node l v r` ；(2) 以上的分类之间是互斥的，即无论 `l` 、 `v` 与 `r` 取什么值，`Leaf` 与 `Node l v r` 都不会相等；(3) `Node` 这样的构造子是函数也是单射。这三条性质对应了 Coq 中的三条证明指令： `destruct` 、 `discriminate` 与 `injection` 。利用它们就可以证明几条最简单的性质：

```
Lemma Node_inj_left: forall l1 v1 r1 l2 v2 r2,
  Node l1 v1 r1 = Node l2 v2 r2 ->
  l1 = l2.
Proof.
  intros.
  (** 现在，Coq证明目标中的前提条件是两个_[Node]_型的二叉树相等。*)
  injection H as H_l H_v H_r.
  (** 上面的_[injection]_指令使用了_[Node]_是单射这一性质。它将原先的前提_[H]_拆分成
      为了三条前提：
      - H_l: l1 = l2
      - H_v: v1 = v2
      - H_r: r1 = r2 *)
  rewrite H_l.
  reflexivity.
Qed.
```

有时，手动为 `injection` 生成的命题进行命名显得很啰嗦，Coq 允许用户使用问号占位，从而让 Coq 进行自动命名。

```
Lemma Node_inj_right: forall l1 v1 r1 l2 v2 r2,
  Node l1 v1 r1 = Node l2 v2 r2 ->
  r1 = r2.
Proof.
  intros.
  injection H as ? ? ?.
  (** 这里，Coq自动命名的结果是使用了_[H]_、_[H0]_与_[H1]_。下面也使用_[apply]_
      指令取代_[rewrite]_简化后续证明。*)
  apply H1.
Qed.
```

如果不需要用到 `injection` 生成的第一与第三个命题，可以将不需要用到的部分用下划线占位。

```
Lemma Node_inj_value: forall l1 v1 r1 l2 v2 r2,
  Node l1 v1 r1 = Node l2 v2 r2 ->
  v1 = v2.
Proof.
  intros.
  injection H as _ ? _.
  apply H.
Qed.
```

下面引理说：若 `Leaf` 与 `Node l v r` 相等，那么 `1 = 2` 。换言之，`Leaf` 与 `Node l v r` 始终不相等，否则就形成了一个矛盾的前提。

```
Lemma Leaf_Node_conflict: forall l v r,
  Leaf = Node l v r -> 1 = 2.
Proof.
  intros.
  (** 下面指令直接从前提中发现矛盾并完成证明。*)
  discriminate H.
Qed.
```

下面这个简单性质与 `tree_reverse` 有关。

```
Lemma reverse_result_Leaf: forall t,
  tree_reverse t = Leaf ->
  t = Leaf.
Proof.
  intros.
  (** 下面的_[destruct]_指令根据_[t]_是否为空树进行分类讨论。*)
  destruct t.
  (** 执行这一条指令之后，Coq中待证明的证明目标由一条变成了两条，对应两种情况。
      为了增加Coq证明的可读性，我们推荐大家使用bullet记号把各个子证明过程分割开
      来，就像一个一个抽屉或者一个一个文件夹一样。Coq中可以使用的bullet标记有：
      _[+ - * ++ -- **]_等等*)
  + reflexivity.
    (** 第一种情况是_[t]_是空树的情况。这时，待证明的结论是显然的。*)
  + discriminate H.
    (** 第二种情况下，其实前提_[H]_就可以推出矛盾。可以看出，_[discriminate]_指
        令也会先根据定义化简，再试图推出矛盾。*)
Qed.
```

Coq 证明脚本 1. discriminate 指令. 如果前提 `H` 是一个关于某归纳类型 `T` 的等式，`discriminate H` 这一指令将先依据定义对等式 `H` 化简，后利用归纳类型定义中各个分支之间的互斥性推出矛盾。具体而言，如果化简后，等式左边是由构造子 `C1` 定义的类型为 `T` 的项，而等式右边是由另一个不同的构造子 `C2` 定义的项，那么 `discriminate H` 将直接由 `H` 推出矛盾。如果 `H` 化简后，等式的至少一侧不是直接由构造子定义的项，那么 `discriminate H` 指令就无法推出矛盾。如果 `H` 化简后，等式两边是由同样的构造子 `C` 定义的，那么该指令将首先运用构造子 `C` 是单射这一性质，将 `H` 拆解成若干个更细粒度的等式，再逐一尝试递归使用 `discriminate` 推出矛盾。

Coq 证明脚本 2. injection...as 指令. 如果前提 `H` 是一个关于某归纳类型 `T` 的等式，`injection H as ...` 这一指令将先依据定义对等式 `H` 化简，后利用归纳类型定义中各个构造子都为单射这一性质得到更多的等式。具体而言，如果 `H` 化简后，等式两边是由同样的构造子 `C` 定义的，那么该指令将运用构造子 `C` 是单射这一性质，将 `H` 拆解成若干个更细粒度的等式，再逐一尝试递归使用 `injection` 处理这些等式。如果构造子 `C` 的某些参数在 `H` 的左右两边本就对应相等，那么 `injection` 指令不会生成这些多余的形如 `x = x` 的等式。`injection H as ...` 指令中，可以手动为新生成的等式命名，也可以交由 Coq 系统命名（用问号 `?`），还可以选择从其中丢弃一些无用的等式（用下划线 `_`）。`injection H as ...` 指令后的等式名数量（含问号与下划线）不得超过实际将产生的等式数量，如果数量不足的，Coq 将使用问号补齐。

下面例子可以让我们看出，复杂情况下也可以直接使用 `discriminate` 指令推出矛盾。

```
Fact tree_fact_ex1: forall t1 t2 t3 x y,
  Node t1 x (Node t2 y t3) = Node t1 x Leaf -> 1 = 2.
Proof.
  intros.
  discriminate H.
Qed.
```

值得一提的是，并非所有关于“归纳类型”的矛盾都可以由 `discriminate` 推出。例如，下面两个例子都无法直接使用 `discriminate` 推出矛盾，如果试图在这两处使用 `discriminate` 推出矛盾，Coq 会反馈报错信息：Not a discriminable equality.

```
Fact tree_fact_ex2: forall t1 v t2,
  Node t1 v t2 = t1 -> 1 = 2.
Proof.
  intros.
  Fail discriminate H.
  (** *)
Abort.
```

```
Fact tree_fact_ex3: forall t t1 t2 x y,
  Node t x t = Node Leaf x (Node t1 y t2) -> 1 = 2.
Proof.
  intros.
  Fail discriminate H.
Abort.
```

下面是在复杂情况下使用 `injection` 指令的例子。

```
Fact tree_fact_ex4: forall t1 t2 t3 x y z,
  Node t1 x (Node t2 y t3) = Node t1 y (Node t3 z t2) -> x = z.
Proof.
  intros.
  injection H as Hxy ? Hyz ?.
  (** 这里，_[injection H]_指令会生成4个等式：
          - Hxy: x = y
          - H: t2 = t3
          - Hyz: y = z
          - H0: t3 = t2
      我们将其中有用的两个命名为了_[Hxy]_与_[Hyz]_。*)
  rewrite Hxy, Hyz.
  reflexivity.
Qed.
```

值得一提的是，这里的 `injection` 指令并不会生成等式 `t1 = t1` 。另外，如果我们使用 `injection ... as` 指令时不为其生成的任何一个等式命名，也不使用问号或者下划线占位，那么 Coq 就会默认为全部由 Coq 系统命名（相当于全部写问号占位）。

```
Fact tree_fact_ex4_alter: forall t1 t2 t3 x y z,
  Node t1 x (Node t2 y t3) = Node t1 y (Node t3 z t2) -> x = z.
Proof.
  intros.
  injection H as.
  (** 此时生成的4个等式全部由Coq系统自动命名
          - H: x = y
          - H0: t2 = t3
          - H1: y = z
          - H2: t3 = t2
      我们将其中有用的两个命名为了_[Hxy]_与_[Hyz]_。*)
  rewrite H, H1.
  reflexivity.
Qed.
```

如果像下面这个例子这样，`injection` 无法将指定的条件分成若干个等式，Coq 系统会报错：Nothing to inject.

```
Fact tree_fact_ex5: forall l1 v1 r1 l2 v2 r2,
  tree_reverse (tree_reverse l1) = Node r1 v1 (Node l2 v2 r2) ->
  l1 = l2.
Proof.
  intros.
  Fail injection H.
Abort.
```

前面提到的 `tree_fact_ex3` 虽然不能直接用 `discriminate` 完成证明，但是也可以先用 `injection` 指令，再后续导出矛盾完成证明。

```
Fact tree_fact_ex3: forall t t1 t2 x y,
  Node t x t = Node Leaf x (Node t1 y t2) -> 1 = 2.
Proof.
  intros.
  injection H as H H0.
  rewrite H in H0.
  discriminate H0.
Qed.
```