在 Coq 中表示集合

1 在 Coq 中表示集合

在 Coq 中往往使用 `X: A -> Prop` 来表示某类型 `A` 中元素构成的集合 `X` 。字面上看，这里的 `A -> Prop` 表示 `X` 是一个从 `A` 中元素到命题的映射，这也相当于说 `X` 是一个关于 `A` 中元素性质。对于每个 `A` 中元素 `a` 而言，`a` 符合该性质 `X` 等价于 `a` 对应的命题 `X a` 为真，又等价于 `a` 是集合 `X` 的元素，在 SetsClass 这一拓展库中也直接写作 `a ∈ X` 。

类似的，在 Coq 中也常常使用 `R: A -> B -> Prop` 来表示 `A` 与 `B` 中元素之间的二元关系。数学上，“`(a, b)` 是集合 `R` 中的元素”可以写作 `(a,b) ∈ R` ，SetsClass 拓展库也支持我们这么表达，当然，它在 Coq 中的实际定义就是 `R a b` 这个命题成立。

SetsClass 拓展库中提供了有关集合的一系列定义。例如：

- 空集：用 `∅` 或者一堆方括号表示，定义为 `Sets.empty` ；
- 全集：定义为 `Sets.full` （全集没有专门的表示符号）；
- 单元集：用一对方括号表示，定义为 `Sets.singleton` ；
- 补集：定义为 `Sets.complement` （补集没有专门的表示符号）；
- 并集：用 `∪` 表示，定义为 `Sets.union` ；
- 交集：用 `∩` 表示，定义为 `Sets.intersect` ；
- 集合相等：用 `==` 表示，定义为 `Sets.equiv` ；
- 元素与集合关系：用 `∈` 表示，定义为 `Sets.In` ；
- 子集关系：用 `⊆` 表示，定义为 `Sets.included` ；

在这些符号中，补集以及其他 Coq 函数的优先级最高，交集的优先级其次，并集的优先级再次，集合相等、集合包含与属于号的优先级最低。例如，下面是两个关于集合的命题，并且其中第二个命题中的括号可以省略：

```
Check forall A (X: A -> Prop), X ∪ ∅ == X.

Check forall A B (X Y: A -> B -> Prop), X ∪ (Y ∩ X) ⊆ X.
```

在 CoqIDE 中，你可以利用 CoqIDE 对于 unicode 的支持打出特殊字符：

- 首先，打出特殊字符的 latex 表示法；
- 再按 shift+ 空格键；
- latex 表示法就自动转化为了相应的特殊字符。

例如，如果你需要打出符号 `∈` ，请先在输入框中输入 `\in` ，当光标紧跟在 `n` 这个字符之后的时候，按 shift+ 空格键即可。

在 VSCode 中，你可以使用 Unicode Latex Input 拓展包从而方便地打出这些特殊字符。安装该拓展包后，只需要输入 `\in` 就会出现提示框显示 `∈` 符号。

值得一提的是，使用 SetsClass 拓展库中的集合时一定要使用双等号 `==` 而不是普通等号 `=` 表示集合相等，SetsClass 拓展库已经为其用户证明了 `==` 是一个等价关系。

2 在 Coq 中证明集合有关性质的简单方法

2.1 规约为逻辑命题

SetsClass 拓展库中的集合运算符都是基于 Coq 中的命题进行定义的。例如，当 `X Y: A -> Prop` 时，`X ∩ Y` 就可以被定义为：

```
fun a => X a /\ Y a
```

。

这与我们对“交”运算的朴素理解是一致的，即，`a ∈ X ∩ Y` 当且仅当 `a ∈ X` 并且 `a ∈ Y` 。类似的，`a ∈ X ∪ Y` 当且仅当 `a ∈ X` 或者 `a ∈ Y` 。在证明中，也可以据此将集合间的运算性质规约为集合与元素之间的逻辑命题。例如，下面在 Coq 中证明了，与另一个集合做交集运算的结果是原集合的子集。

```
Theorem Sets1_intersect_included1: forall A (X Y: A -> Prop),
  X ∩ Y ⊆ X.
Proof.
  intros.
  (** 下面一条命令_[Sets_unfold]_是SetsClass库提供的自动证明指令，它可以将有关
      集合的性质转化为有关命题的性质。*)
  Sets_unfold.
  (** 原本要证明的关于交集的性质现在就转化为了：
       _[forall a : A, a ∈ X /\ a ∈ Y -> a ∈ X]_
      这个关于逻辑的命题在Coq中是容易证明的。*)
  intros.
  tauto.
Qed.
```

下面是一条关于并集运算的性质。

```
Lemma Sets1_included_union1: forall A (X Y: A -> Prop),
  X ⊆ X ∪ Y.
Proof.
  intros.
  Sets_unfold.
  (** 经过转化，要证明的结论是：_[forall a : A, a ∈ X -> a ∈ X \/ a ∈ Y]_。*)
  intros.
  tauto.
Qed.
```

我们也可以利用集合运算相关的前提进行证明。

```
Example Sets2_proof_sample1: forall A B (X Y Z: A -> B -> Prop),
  X ∪ Y ⊆ Z ->
  Y ⊆ Z.
Proof.
  intros.
  Sets_unfold in H.
  Sets_unfold.
  intros a b.
  specialize (H a b).
  tauto.
Qed.
```

当所需证明性质较为简单的时候，将集合相关的 Coq 命题展开为逻辑相关的命题是一种有效的自动证明方法。然而，需要证明的结论有时也会比较复杂，例如，哪怕下面这条性质“交集运算对有穷多个集合的并具有分配律”的证明也需要我们对有穷长的集合序列做归纳证明。

```
A ∩ (B1 ∪ B2 ∪ ... ∪ Bn) == (A ∩ B1) ∪ ... ∪ (A ∩ Bn)
```

这时，在集合层面基于集合运算的基本性质表述证明就变得更为直观简便了。从下一节开始，我们将介绍这样的证明方法。

2.2 使用 Rewrite 指令

我们熟知，集合相等是一个等价关系，集合包含具有自反性和传递性。在 Coq 中，这些性质即是说：

```
Equivalence Sets.equiv
Reflexive Sets.included
Transitive Sets.included
```

SetsClass 拓展库已经证明了这些定理，因此我们就可以把 `rewrite` 、`reflexivity` 等证明指令用在集合相关的证明中。下面就是两个简单的例子。

```
Example Sets1_proof_sample2: forall (A: Type) (X Y Z: A -> Prop),
  X == Y -> X == Z -> Y == Z.
Proof.
  intros.
  rewrite <- H, <- H0.
  reflexivity.
Qed.
```

```
Example Sets1_proof_sample3: forall (A: Type) (F: (A -> Prop) -> (A -> Prop)),
  (forall X: A -> Prop, X ⊆ F X) ->
  (forall X: A -> Prop, X ⊆ F (F X)).
Proof.
  intros.
  rewrite <- H, <- H.
  reflexivity.
Qed.
```

另外，集合间的交集、并集和补集运算会保持“包含”与“被包含”关系，也会保持集合相等关系。在 SetsClass 拓展库中，已经证明了：

```
Sets_union_mono:
  Proper (Sets.included ==> Sets.included ==> Sets.included) Sets.union
Sets_intersect_mono:
  Proper (Sets.included ==> Sets.included ==> Sets.included) Sets.intersect
Sets_union_congr:
  Proper (Sets.equiv ==> Sets.equiv ==> Sets.equiv) Sets.union
Sets_intersect_mono:
  Proper (Sets.equiv ==> Sets.equiv ==> Sets.equiv) Sets.intersect
Sets_complement_congr:
  Proper (Sets.equiv ==> Sets.equiv) Sets.complement
Sets_complement_mono:
  Proper (Sets.included --> Sets.included) Sets.complement
```

当然，`Sets.equiv` 与 `Sets.included` 也满足一些基于 `Proper` 描述的性质。

```
Proper (Sets.included --> Sets.included ==> Basics.impl) Sets.included
Proper (Sets.equiv ==> Sets.equiv ==> iff) Sets.equiv
Proper (Sets.equiv ==> Sets.equiv ==> iff) Sets.included
```

上面这三条性质中，前两条是由 `Sets.included` 与 `Sets.equiv` 的传递性自动推导得到的，而第三条性质是 SetsClass 拓展库额外证明并提供给用户的。这些性质结合在一起，使得我们在许多时候都可以用 Coq 的

rewrite 指令较为方便地完成证明。下面是一个简单的例子。

```
Example Sets1_proof_sample4: forall (A: Type) (X1 X2 Y1 Y2: A -> Prop),
  X1 == X2 -> Y1 ⊆ Y2 -> X1 ∪ Y1 ⊆ X2 ∪ Y2.
Proof.
  intros.
  rewrite H, H0.
  reflexivity.
Qed.
```