Coq 中的集合与关系

1 二元关系与二元关系上的运算

数学上，如果称 R 是一个从 A 集合到 B 集合的一个二元关系，那么它表示 $R \subseteq A \times B$。如果 A 与 B 是同一个集合，即 $R \subseteq A \times A$，那么我们也称 R 是一个 A 集合上的二元关系。因此，一个二元关系 R 是一些有序对的集合。一般而言，二元关系会被用于表示这些有序对的共同性质。例如：

$$\{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x < y\} = \{(-1,0),(-1,1),(0,5),(10,100),...\}$$

$$\{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid \exists z.\ x * z = y\} = \{(1,2),(2,6),(3,15),(-1,1),...\}$$

这两个集合分别表示整数上的“小于”关系和整数上的“整除”关系。

由于二元关系是集合，因此二元关系之间可以做交集、并集等集合运算。除此之外，二元关系之间还可以做一种特有的运算：二元关系的连接（亦可称为二元关系的复合）。数学上，二元关系的连接可以用下面式子定义：

$(x,z) \in R_1 \circ R_2$ 当且仅当存在 y 使得 $(x,y) \in R_1$ 并且 $(y,z) \in R_2$。

例如，当 S_1、S_2、$S_\leqslant$ 与 $S_<$ 表示整数上的“加 1”关系、“加 2”关系、“小于等于”关系和“小于”关系的时候，即：

$$S_1 = \{(x, x+1) \mid x \in \mathbb{Z}\}$$
$$S_2 = \{(x, x+2) \mid x \in \mathbb{Z}\}$$
$$S_< = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x < y\}$$
$$S_\leqslant = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x \leqslant y\}$$

我们就知道：

$$S_1 \circ S_1 = S_2$$
$$S_1 \circ S_\leqslant = S_\leqslant \circ S_1 = S_<$$
$$S_1 \circ S_< = S_< \circ S_1 = \{(x,y) \in \mathbb{Z} \times \mathbb{Z} \mid x + 1 < y\} \subseteq S_<$$

对于任意集合 A，空集是集合 A 上的二元关系，$A \times A$ 本身（某种意义上的全集）也是集合 A 上的二元关系。除此之外，我们还经常需要用到下面两个特殊的二元关系，“相等关系”与“测试”。

$$I_A = \{(a,a) \mid a \in A\}$$

如果 $X \subseteq A$，那么 $\mathrm{test}(X) = \{(a,a) \mid a \in X\}$

因此，对于空集和“全集”而言，$\mathrm{test}(\varnothing) = \varnothing$ 并且 $\mathrm{test}(A) = I_A$。

2 在 Coq 中表示集合与二元关系

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。

SetsClass 拓展库中提供了这些关于二元关系的定义：

- 二元关系的连接：用 `∘` 表示，定义为 `Rels.concat` ；
- 相等关系：定义为 `Rels.id` （没有专门的表示符号）；
- 测试：定义为 `Rels.test` （没有专门的表示符号）。

基于此，我们可以将一些二元关系运算的例子写作 Coq 命题，下面就是一个这样的例子。

```
Fact plus_1_concat_plus_1:
  forall S1 S2: Z -> Z -> Prop,
    (forall n m, (n, m) ∈ S1 <-> m = n + 1) ->
    (forall n m, (n, m) ∈ S2 <-> m = n + 2) ->
    S1 ○ S1 == S2.
Proof.
  intros S1 S2 H_S1 H_S2.
  Sets_unfold.
  intros x z.
  (** _[Sets_unfold]_指令将_[○]_的定义展开，现在需要证明：
        - exists y, (x, y) ∈ S1 /\ (y, z) ∈ S1
      当且仅当
        - (x, z) ∈ S2]_。*)
  rewrite H_S2.
  setoid_rewrite H_S1.
  (** 根据_[S1]_与_[S2]_的定义，就只需要证明：
        - (exists y, y = x + 1 /\ z = y + 1) <-> z = x + 2 *)
  split.
  + intros [y [? ?]].
    lia.
  + intros.
    exists (x + 1).
    lia.
Qed.
```

3 集合与二元关系 Coq 证明

3.1 集合命题的基本证明方法

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。

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.
```

3.2 交集与并集性质的 Coq 证明

相应的，要证明两个集合相等，就只需要证明它们相互包含。在 Coq 中只需要 `apply` 下面引理来实现这个证明步骤。

```
  Sets_equiv_Sets_included:
    forall x y, x == y <-> x ⊆ y /\ y ⊆ x
```

要证明某两个集合的交集包含第三个集合，或者证明某两个集合的交集被第三个集合包含，又可以采取以下方法。

`x ⊆ y ∩ z` 可以被规约为 `x ⊆ y` 与 `x ⊆ z`；
`x ∩ y ⊆ z` 可以被规约为 `x ⊆ z`；
`x ∩ y ⊆ z` 也可以被规约为 `y ⊆ z`。

在 Coq 中，前一种证明可以通过 `apply` 下面引理实现。

```
Sets_included_intersect:
  forall x y z, x ⊆ y -> x ⊆ z -> x ⊆ y ∩ z
```

而后两种证明可以通过 `rewrite` 下面引理实现。

```
Sets_intersect_included1:
  forall x y, x ∩ y ⊆ x
Sets_intersect_included2:
  forall x y, x ∩ y ⊆ y
```

例如，我们可以如下证明集合交具有交换律和结合律。

```
Theorem Sets1_intersect_comm:
  forall {A: Type} (x y: A -> Prop),
    x ∩ y == y ∩ x.
Proof.
  intros.
  (** 首先，要证明两个集合相等只需要证明它们互为子集。*)
  apply Sets_equiv_Sets_included; split.
  + (** 第一个分支需要证明_[x ∩ y ⊆ y ∩ x]_，右边是两个集合的交集，因此这两个集合都包
        含左边集合即可。*)
    apply Sets_included_intersect.
    - (** 现在需要证明_[x ∩ y ⊆ y]_，形式上，是要证明左侧两个集合的交集是右侧集合的子
          集，这只需要证明左侧的第二个集合是右侧集合的子集就够了。 *)
      rewrite Sets_intersect_included2.
      reflexivity.
    - (** 类似的，这个子分支需要证明_[x ∩ y ⊆ x]_，我们可以选择将其归结为证明左边的一
          个集合是右边集合的子集。。 *)
      rewrite Sets_intersect_included1.
      reflexivity.
  + (** 第二个分支的证明是类似的。*)
    (** ... 证明详见Coq源代码 ... *)
Qed.
```

```
Theorem Sets1_intersect_assoc:
  forall {A: Type} (x y z: A -> Prop),
    (x ∩ y) ∩ z == x ∩ (y ∩ z).
Proof.
  intros.
  (** 与证明交集交换律的时候类似，我们将两个集合相等的证明归于为证明它们互为子集。*)
  apply Sets_equiv_Sets_included; split.
  + (** 第一个分支需要证明_[(x ∩ y) ∩ z ⊆ x ∩ (y ∩ z)]_。要证明左侧是右侧三个集合交集
        的子集，就需要证明左侧是右侧每一个集合的子集。*)
    apply Sets_included_intersect; [| apply Sets_included_intersect].
    (** 现在三个证明目标分别是：
        - (x ∩ y) ∩ z ⊆ x
        - (x ∩ y) ∩ z ⊆ y
        - (x ∩ y) ∩ z ⊆ z
        证明时只需要指明左边三个集合中哪一个是右边的子集即可。*)
    (** ... 证明详见Coq源代码 ... *)
  + (** 第二个分支的证明是类似的。*)
    apply Sets_included_intersect; [apply Sets_included_intersect |].
    (** ... 证明详见Coq源代码 ... *)
Qed.
```

对于并集运算而言，要证明某两个集合的并集包含第三个集合，或者证明某两个集合的并集被第三个集合包含，就类似于要证明形如 `P -> Q \/ R` 或 `P \/ Q -> R` 的命题。具体地，

`x ⊆ y ∪ z` 可以被规约为 `x ⊆ y`；
`x ⊆ y ∪ z` 也可以被规约为 `x ⊆ z`；
`x ∪ y ⊆ z` 可以被规约为 `x ⊆ z` 与 `y ⊆ z`。

在 Coq 中，前两种证明可以通过从右向左 `rewrite` 下面引理实现。

```
  Sets_included_union1:
    forall x y, x ⊆ x ∪ y
  Sets_included_union2:
    forall x y, y ⊆ x ∪ y
```

而后一种证明则可以通过 `apply` 下面引理实现。

```
  Sets_union_included:
    forall x y z, x ⊆ z -> y ⊆ z -> x ∪ y ⊆ z;
```

有时，包含号 `⊆` 左侧的集合不是一个并集，需要先使用交集对于并集的分配律才能使用 `Sets_union_included`。

习题 1. 请证明下面关于集合的性质。

```
Fact sets_fact_ex: forall (A: Type) (X Y: A -> Prop),
  X ⊆ Y ->
  X ∩ Y == X.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 2. 请证明下面集合运算的性质。

```
Example Sets1_intersect_absorb_union:
  forall {A: Type} (x y: A -> Prop),
    x ∩ (x ∪ y) == x.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 3. 请证明下面集合运算的性质。

```
Example Sets1_union_absorb_intersect:
  forall {A: Type} (x y: A -> Prop),
    x ∪ (x ∩ y) == x.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

总而言之，以下这些 SetsClass 拓展库中的引理，构成了供我们手动证明集合运算性质的基本方法。

```
  Sets_equiv_Sets_included:
    forall x y, x == y <-> x ⊆ y /\ y ⊆ x
  Sets_intersect_included1:
    forall x y, x ∩ y ⊆ x
  Sets_intersect_included2:
    forall x y, x ∩ y ⊆ y
  Sets_included_intersect:
    forall x y z, x ⊆ y -> x ⊆ z -> x ⊆ y ∩ z
  Sets_included_union1:
    forall x y, x ⊆ x ∪ y
  Sets_included_union2:
    forall x y, y ⊆ x ∪ y
  Sets_union_included:
    forall x y z, x ⊆ z -> y ⊆ z -> x ∪ y ⊆ z
  Sets_intersect_union_distr_r:
    forall x y z, (x ∪ y) ∩ z == x ∩ z ∪ y ∩ z
  Sets_intersect_union_distr_l:
    forall x y z, x ∩ (y ∪ z) == x ∩ y ∪ x ∩ z
```

基于这些引理,我们前面已经证明集合交的交换律与结合律。这些我们演示过的证明都已经包含在 SetsClass 拓展库中了，除此之外，SetsClass 拓展库还提供了集合并的交换律与结合律以及集合并对集合交左右分配律。SetsClass 拓展库中的证明也不限于形如 `A -> Prop` 类型的 Coq 集合，而一并考虑了 `A -> B -> Prop` 、`A -> B -> C -> Prop` 等所有可能的情形。

```
  Sets_intersect_comm:
    forall x y, x ∩ y == y ∩ x
  Sets_intersect_assoc:
    forall x y z, (x ∩ y) ∩ z == x ∩ (y ∩ z)
  Sets_union_comm:
    forall x y, x ∪ y == y ∪ x
  Sets_union_assoc:
    forall x y z, (x ∪ y) ∪ z == x ∪ (y ∪ z)
  Sets_union_intersect_distr_l:
    forall x y z, x ∪ (y ∩ z) == (x ∪ y) ∩ (x ∪ z)
  Sets_union_intersect_distr_r:
    forall x y z, (x ∩ y) ∪ z == (x ∪ z) ∩ (y ∪ z)
```

3.3 空集、补集、全集、无穷交与无穷并性质的 Coq 证明

SetsClass 拓展库对于空集的支持主要是通过以下性质：空集是一切集合的子集。

```
  Sets_empty_included: forall x, ∅ ⊆ x
```

相对应的，一切集合都是全集的子集。

```
  Sets_included_full: forall x, x ⊆ Sets.full
```

基于这两条性质，可以证明许多有用的导出性质。SetsClass 提供的导出性质有：

```
Sets_union_empty_l: forall x, ∅ ∪ x == x
Sets_union_empty_r: forall x, x ∪ ∅ == x
Sets_intersect_empty_l: forall x, ∅ ∩ x == ∅
Sets_intersect_empty_r: forall x, x ∩ ∅ == ∅
Sets_union_full_l: forall x, Sets.full ∪ x == Sets.full
Sets_union_full_r: forall x, x ∪ Sets.full == Sets.full
Sets_intersect_full_l: forall x, Sets.full ∩ x == x
Sets_intersect_full_r: forall x, x ∩ Sets.full == x
Sets_equiv_empty_fact: forall x, x ⊆ ∅ <-> x == ∅
Sets_equiv_full_fact: forall x, Sets.full ⊆ x <-> x == Sets.full
```

习题 4. 前面已经提到，SetsClass 拓展库已经证明了 `Sets_intersect_empty_l` 。请你只使用 `Sets_empty_included` 以及交集的性质证明它。

```
Lemma Sets1_intersect_empty_l:
  forall (A: Type) (x: A -> Prop), ∅ ∩ x == ∅.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

SetsClass 库中提供的关于补集的性质有下面四组。(1) 一个集合与自己的补集求交或求并会得到空集或全集。

```
Sets_intersect_complement_self
  forall x, x ∩ Sets.complement x == ∅
Sets_complement_self_intersect
  forall x, Sets.complement x ∩ x == ∅
Sets_union_complement_self:
  forall x, x ∪ Sets.complement x == Sets.full
Sets_complement_self_union
  forall x, Sets.complement x ∪ x == Sets.full
```

(2) 补集的补集是原集合。

```
Sets_complement_complement
  forall x, Sets.complement (Sets.complement x) == x
```

(3) 交集或并集的补集满足德摩根律。

```
Sets_complement_union
  forall x y,
    Sets.complement (x ∪ y) ==
    Sets.complement x ∩ Sets.complement y
Sets_complement_intersect
  forall x y,
    Sets.complement (x ∩ y) ==
    Sets.complement x ∪ Sets.complement y
```

(4) 补集与包含关系之间满足类似逆否命题之间的性质。

```
Sets_contrapositive_PP:
  forall x y, x ⊆ y -> Sets.complement y ⊆ Sets.complement x
Sets_contrapositive_CC:
  forall x y, Sets.complement y ⊆ Sets.complement x -> x ⊆ y
Sets_contrapositive_PC:
  forall x y, y ⊆ Sets.complement x -> x ⊆ Sets.complement y
Sets_contrapositive_CC:
  forall x y, Sets.complement x ⊆ y -> Sets.complement y ⊆ x
```

如果 $X_0, X_1, X_2, \ldots$ 是一列无穷长的集合序列，那么数学上就可以将它们的并集和它们的交集就可以定义为：

$$\bigcup_{n\in\mathbb{N}} X_n \triangleq \{x \mid \exists n \in \mathbb{N}.\ x \in X_n\}$$

$$\bigcap_{n\in\mathbb{N}} X_n \triangleq \{x \mid \forall n \in \mathbb{N}.\ x \in X_n\}$$

在 Coq 中，一般会用 `X: nat -> A -> Prop` （如果 $X_n \subseteq A$）或 `X: nat -> A -> B -> Prop` （如果 $X_n \subseteq A \times B$）等形式表示 X 是一个无穷长的集合序列。SetsClass 拓展库允许用户用 `⋃ X` 和 `⋂ X` 表示它们的无穷并和无穷交。

从数学上看，上面定义中自然数集 $\mathbb{N}$ 起到了指标集的作用。如果把它推广到一般的指标集，那么上面的数学公式又可以改写为：

$$\bigcup_{i\in I} X_i \triangleq \{x \mid \exists i \in I.\ x \in X_i\}$$

$$\bigcap_{i\in I} X_i \triangleq \{x \mid \forall i \in I.\ x \in X_i\}$$

值得一提的是，基于一般指标集 I 的集合交不一定是一个合法的数学定义。主要是当指标集 I 是空集的时候，$X_{i\in I}$ 的集合交是“全集”，但是公理集合论中并不存在“包含所有数学对象的全集”。不过，如果在 Coq 中约定 `X: I -> A -> Prop` （即 $X_i \subseteq A$），那么这些集合的交其实是可以良定义的，只需在 I 是空集的时候规定 $X_{i\in I}$ 的交集是“全集” A 就可以了。SetsClass 拓展库中，基于指标集的集合并与集合交是由这两个 Coq 定义支持的：

```
Sets.indexed_union: forall {I: Type}, (I -> T) -> T
Sets.indexed_intersect: forall {I: Type}, (I -> T) -> T
```

其中，`T` 是具有 `A -> Prop` 、 `A -> B -> Prop` 等形式的 Coq 类型。

除此之外，集合论上，集族 U 的广义并指的是 $\{x \mid \exists X \in U.\ x \in X\}$。在 Coq 中，如果 `U: (A -> Prop) -> Prop` ，那么既可以说 `U` 是 `A` 集合的子集的一元谓词，也可以说 `U` 是一些 `A` 集合的子集构成的集合。SetsClass 拓展库允许我们用 `⋃ U` 表示 `U` 的广义并。如果 `U` 具有

`(A -> B -> Prop) -> Prop` 或者 `(A -> B -> C -> Prop) -> Prop`

等类型，也可以同样表示 `U` 的广义并。类似的，SetsClass 拓展库也定义了这些 `U` 的广义交，如果 `U: (A -> Prop) -> Prop` 本身是空集，那么它的广义交被定义为“全集” `A` 。

总的来说，SetsClass 拓展库提供了两种支持无穷交集和无穷并集的定义。

- 基于指标集的集合并： `⋃ X` ， `Sets.indexed_union X`

$$\{x \mid \exists i \in I.\ X_i\}$$

- 基于指标集的集合交： `⋂ X` ， `Sets.indexed_intersect X`

$$\{x \mid \forall i \in I.\ X_i\}$$

- 广义并： `⋃ U` ， `Sets.general_union U`

$$\{x \mid \exists X \in U.\ x \in X\}$$

- 广义交：`⋂ U`，`Sets.general_intersect U`

$$\{x \mid \forall X \in U.\ x \in X\}$$

它们相关性质的证明方式与普通并集与交集的证明方式是类似的。下面是一个简单的例子。

```
Example Sets1_union_indexed_intersect_fact:
  forall {A: Type} (x: nat -> A -> Prop) (y: A -> Prop),
    (⋂ x) ∪ y ⊆ ⋂ (fun n => x n ∪ y).
Proof.
  intros.
  (** 要证明左边集合是右边这无穷多个集合交集的子集，就需要证明左边集合是右边每一个集合
      的子集。 *)
  apply Sets_included_indexed_intersect.
  intros n.
  (** 现在只需要证明_[⋂ x ∪ y ⊆ x n ∪ y]_。 *)
  rewrite (Sets_indexed_intersect_included n).
  reflexivity.
Qed.
```

下面是 SetsClass 库中已经证明的关于无穷交集与无穷并集（基于指标集）的性质。

```
  Sets_included_indexed_intersect:
    forall xs y, (forall n, y ⊆ xs n) -> y ⊆ ⋂ xs
  Sets_included_indexed_union:
    forall n xs x, xs n ⊆ ⋃ xs
  Sets_indexed_union_included:
    forall xs y, (forall n, xs n ⊆ y) -> ⋃ xs ⊆ y
  Sets_indexed_intersect_included:
    forall n xs, ⋂ xs ⊆ xs n
  Sets_intersect_indexed_union_distr_r:
   forall xs y, ⋃ xs ∩ y == ⋃ (fun n => xs n ∩ y)
  Sets_intersect_indexed_union_distr_l:
   forall x ys, x ∩ ⋃ ys == ⋃ (fun n => x ∩ ys n)
```

下面是广义交与广义并的性质。

```
  Sets_general_intersect_included:
    forall U x, U x -> ⋂ U ⊆ x
  Sets_general_union_included:
    forall U y, (forall x, U x -> x ⊆ y) -> ⋃ U ⊆ y
  Sets_included_general_union:
    forall U x, U x -> x ⊆ ⋃ U
  Sets_included_general_intersect:
    forall U y, (forall x, U x -> y ⊆ x) -> y ⊆ ⋂ U
```

3.4 二元关系运算性质的 Coq 证明

二元关系除了满足普通集合的运算性质之外，还有几条额外的重要运算性质。

- 结合律：`(x ∘ y) ∘ z == x ∘ (y ∘ z)`
- 左单位元：`Rels.id ∘ x == x`
- 右单位元：`x ∘ Rels.id == x`
- 左分配律：`x ∘ (y ∪ z) == x ∘ y ∪ x ∘ z`
- 右分配律：`(x ∪ y) ∘ z == x ∘ z ∪ y ∘ z`

另外，二元关系对并集的分配律对于无穷并集也成立。这些性质对应了 SetsClass 库中的下面这些定理。

```
  Rels_concat_assoc:
    forall x y z, (x ○ y) ○ z == x ○ y ○ z
  Rels_concat_id_l:
    forall x, Rels.id ○ x == x
  Rels_concat_id_r:
    forall x, x ○ Rels.id == x
  Rels_concat_union_distr_l:
    forall x y1 y2, x ○ (y1 ∪ y2) == x ○ y1 ∪ x ○ y2
  Rels_concat_union_distr_r:
    forall x1 x2 y, (x1 ∪ x2) ○ y == x1 ○ y ∪ x2 ○ y
  Rels_concat_indexed_union_distr_l:
    forall x ys, x ○ ⋃ ys == ⋃ (fun n => x ○ ys n)
  Rels_concat_indexed_union_distr_r:
    forall xs y, ⋃ xs ○ y == ⋃ (fun n => xs n ○ y)
```

测试关系 `Rels.test` 有下面几条重要性质。

```
  Rels_test_full: Rels.test Sets.full == Rels.id
  Rels_test_empty: Rels.test ∅ == ∅
  Rels_test_is_id:
    forall x, x == Sets.full -> Rels.test x == Rels.id
  Rels_test_is_empty:
    forall x, x == ∅ -> Rels.test x == ∅
```

习题 5. 请根据二元关系连接的定义证明下面性质。

```
Lemma plus_n_plus_m:
  forall (plus_rel: Z -> Z -> Z -> Prop),
    (forall n m1 m2, (m1, m2) ∈ plus_rel n <-> m1 + n = m2) ->
    (forall n m, plus_rel n ○ plus_rel m == plus_rel (n + m)).
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

习题 6. 请根据二元关系连接的定义证明下面性质。

```
Lemma Rels22_concat_assoc:
  forall {A: Type} (x y z: A -> A -> Prop),
    (x ○ y) ○ z == x ○ (y ○ z).
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

```
Lemma Rels22_concat_id_l:
  forall {A: Type} (x: A -> A -> Prop),
    Rels.id ○ x == x.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```

```
Lemma Rels22_concat_union_distr_r:
  forall {A: Type} (x y z: A -> A -> Prop),
    (x ∪ y) ○ z == x ○ z ∪ y ○ z.
(* 请在此处填入你的证明，以_[Qed]_结束。 *)
```