

SOFTWARE FOUNDATIONS

VOLUME 1: LOGICAL FOUNDATIONS

[TABLE OF CONTENTS](#)
[INDEX](#)
[ROADMAP](#)

REL

PROPERTIES OF RELATIONS

This short (and optional) chapter develops some basic definitions and a few theorems about binary relations in Coq. The key definitions are repeated where they are actually used (in the [Smallstep](#) chapter of *Programming Language Foundations*), so readers who are already comfortable with these ideas can safely skim or skip this chapter. However, relations are also a good source of exercises for developing facility with Coq's basic reasoning facilities, so it may be useful to look at this material just after the [IndProp](#) chapter.

```
Set Warnings "-notation-overridden,-parsing".
Require Export IndProp.
```

Relations

A binary *relation* on a set X is a family of propositions parameterized by two elements of X — i.e., a proposition about pairs of elements of X .

```
Definition relation (X: Type) := X → X → Prop.
```

Confusingly, the Coq standard library hijacks the generic term "relation" for this specific instance of the idea. To maintain consistency with the library, we will do the same. So, henceforth the Coq identifier `relation` will always refer to a binary relation between some set and itself, whereas the English word "relation" can refer either to the specific Coq concept or the more general concept of a relation between any number of possibly different sets. The context of the discussion should always make clear which is meant.

An example relation on `nat` is `le`, the less-than-or-equal-to relation, which we usually write $n_1 \leq n_2$.

```
Print le.
(* ==> Inductive le (n : nat) : nat -> Prop :=
      le_n : n <= n
```

```

      | le_S : forall m : nat, n <= m -> n <= S m *)
Check le : nat -> nat -> Prop.
Check le : relation nat.

```

(Why did we write it this way instead of starting with `Inductive le : relation nat ...`? Because we wanted to put the first `nat` to the left of the `:`, which makes Coq generate a somewhat nicer induction principle for reasoning about \leq .)

Basic Properties

As anyone knows who has taken an undergraduate discrete math course, there is a lot to be said about relations in general, including ways of classifying relations (as reflexive, transitive, etc.), theorems that can be proved generically about certain sorts of relations, constructions that build one relation from another, etc. For example...

Partial Functions

opposite of total function!
different from 1-to-1 (injective)
and being **اڤوشا**!!!

A relation R on a set X is a *partial function* if, for every x , there is at most one y such that $R\ x\ y$ — i.e., $R\ x\ y_1$ and $R\ x\ y_2$ together imply $y_1 = y_2$.

```

Definition partial_function {X: Type} (R: relation X) :=
  ∀ x y1 y2 : X, R x y1 -> R x y2 -> y1 = y2.

```

For example, the `next_nat` relation defined earlier is a partial function.

```

Print next_nat.
(* ==> Inductive next_nat (n : nat) : nat -> Prop :=
      nn : next_nat n (S n) *)
Check next_nat : relation nat.

Theorem next_nat_partial_function :
  partial_function next_nat.
+

```

However, the \leq relation on numbers is not a partial function. (Assume, for a contradiction, that \leq is a partial function. But then, since $0 \leq 0$ and $0 \leq 1$, it follows that $0 = 1$. This is nonsense, so our assumption was contradictory.)

```

Theorem le_not_a_partial_function :
  ¬ (partial_function le).
+

```

Exercise: 2 stars, optional (total relation not partial)

Show that the `total_relation` defined in earlier is not a partial function.

```

(* FILL IN HERE *)

```

□

Exercise: 2 stars, optional (empty relation partial)

Show that the `empty_relation` that we defined earlier is a partial function.

```
(* FILL IN HERE *)
```

□

Reflexive Relations

A *reflexive* relation on a set X is one for which every element of X is related to itself.

```
Definition reflexive {X: Type} (R: relation X) :=
  ∀ a : X, R a a.
```

```
Theorem le_reflexive :
  reflexive le.
```

+

Transitive Relations

A relation R is *transitive* if $R a c$ holds whenever $R a b$ and $R b c$ do.

```
Definition transitive {X: Type} (R: relation X) :=
  ∀ a b c : X, (R a b) → (R b c) → (R a c).
```

```
Theorem le_trans :
  transitive le.
```

+

```
Theorem lt_trans:
  transitive lt.
```

+

Exercise: 2 stars, optional (le_trans hard way)

We can also prove `lt_trans` more **laboriously** by induction, without using `le_trans`. Do this.

```
Theorem lt_trans' :
  transitive lt.
Proof.
  (* Prove this by induction on evidence that m is less than o. *)
  unfold lt. unfold transitive.
  intros n m o Hnm Hmo.
  induction Hmo as [| m' Hm'o].
  (* FILL IN HERE *) Admitted.
```

□

Exercise: 2 stars, optional (lt_trans")

Prove the same thing again by induction on o .

```
Theorem lt_trans'' :
  transitive lt.
```

+

□

The transitivity of le , in turn, can be used to prove some facts that will be useful later (e.g., for the proof of antisymmetry below)...

Theorem `le_Sn_le` : $\forall n\ m, S\ n \leq m \rightarrow n \leq m$.

+

Exercise: 1 star, optional (le S n)

Theorem `le_S_n` : $\forall n\ m, (S\ n \leq S\ m) \rightarrow (n \leq m)$.

man that took me a long time :(

Proof.

(* FILL IN HERE *) **Admitted.**

□

Exercise: 2 stars, optional (le Sn n inf)

maybe later!!!

Provide an informal proof of the following theorem:

Theorem: For every n , $\neg (S\ n \leq n)$

A formal proof of this is an optional exercise below, but try writing an informal proof without doing the formal proof first.

Proof:

(* FILL IN HERE *)

□

Exercise: 1 star, optional (le Sn n)

Theorem `le_Sn_n` : $\forall n, \neg (S\ n \leq n)$.

Proof.

(* FILL IN HERE *) **Admitted.**

□

Reflexivity and transitivity are the main concepts we'll need for later chapters, but, for a bit of additional practice working with relations in Coq, let's look at a few other common ones...

Symmetric and Antisymmetric Relations

A relation R is *symmetric* if $R\ a\ b$ implies $R\ b\ a$.

Definition `symmetric` {X: Type} (R: relation X) :=
 $\forall a\ b : X, (R\ a\ b) \rightarrow (R\ b\ a)$.

Exercise: 2 stars, optional (le not symmetric)

Theorem `le_not_symmetric` :
 $\neg (\text{symmetric } \text{le})$.

Proof.

(* FILL IN HERE *) **Admitted.**

□

A relation R is *antisymmetric* if $R\ a\ b$ and $R\ b\ a$ together imply $a = b$ — that is, if the only "cycles" in R are trivial ones.

```
Definition antisymmetric {X: Type} (R: relation X) :=
  ∀ a b : X, (R a b) → (R b a) → a = b.
```

Exercise: 2 stars, optional (le antisymmetric)

found interesting proofs on the internet
definitely discuss!!!

```
Theorem le_antisymmetric :
  antisymmetric le.
Proof.
  (* FILL IN HERE *) Admitted.
```

□

Exercise: 2 stars, optional (le step)

```
Theorem le_step : ∀ n m p,
  n < m →
  m ≤ S p →
  n ≤ p.
Proof.
  (* FILL IN HERE *) Admitted.
```

□

Equivalence Relations

A relation is an *equivalence* if it's reflexive, symmetric, and transitive.

```
Definition equivalence {X:Type} (R: relation X) :=
  (reflexive R) ∧ (symmetric R) ∧ (transitive R).
```

Partial Orders and Preorders

A relation is a *partial order* when it's reflexive, *anti*-symmetric, and transitive. In the Coq standard library it's called just "order" for short.

```
Definition order {X:Type} (R: relation X) :=
  (reflexive R) ∧ (antisymmetric R) ∧ (transitive R).
```

A preorder is almost like a partial order, but doesn't have to be antisymmetric.

```
Definition preorder {X:Type} (R: relation X) :=
  (reflexive R) ∧ (transitive R).
```

```
Theorem le_order :
  order le.
```

+

Reflexive, Transitive Closure

The *reflexive, transitive closure* of a relation R is the smallest relation that contains R and that is both reflexive and transitive. Formally, it is defined like this in the Relations module of the Coq standard library:

```

Inductive clos_refl_trans {A: Type} (R: relation A) : relation A
:=
| rt_step :  $\forall x y, R x y \rightarrow \text{clos\_refl\_trans } R x y$ 
| rt_refl :  $\forall x, \text{clos\_refl\_trans } R x x$ 
| rt_trans :  $\forall x y z,$ 
     $\text{clos\_refl\_trans } R x y \rightarrow$ 
     $\text{clos\_refl\_trans } R y z \rightarrow$ 
     $\text{clos\_refl\_trans } R x z.$ 

```

For example, the reflexive and transitive closure of the `next_nat` relation coincides with the `le` relation.

HARD TO GRASP!!
not sure if I completely got the whole concept!

```

Theorem next_nat_closure_is_le :  $\forall n m,$ 
   $(n \leq m) \leftrightarrow ((\text{clos\_refl\_trans next\_nat}) n m).$ 
+

```

The above definition of reflexive, transitive closure is natural: it says, explicitly, that the reflexive and transitive closure of R is the least relation that includes R and that is closed under rules of reflexivity and transitivity. But it turns out that this definition is not very convenient for doing proofs, since the "nondeterminism" of the `rt_trans` rule can sometimes lead to tricky inductions. Here is a more useful definition:

```

Inductive clos_refl_trans_1n {A : Type}
  (R : relation A) (x : A)
  : A  $\rightarrow$  Prop :=
| rtl_n_refl :  $\text{clos\_refl\_trans\_1n } R x x$ 
| rtl_n_trans (y z : A) :
   $R x y \rightarrow \text{clos\_refl\_trans\_1n } R y z \rightarrow$ 
   $\text{clos\_refl\_trans\_1n } R x z.$ 

```

Our new definition of reflexive, transitive closure "bundles" the `rt_step` and `rt_trans` rules into the single rule `step`. The left-hand premise of this step is a single use of R , leading to a much simpler induction principle.

Before we go on, we should check that the two definitions do indeed define the same relation...

First, we prove two lemmas showing that `clos_refl_trans_1n` mimics the behavior of the two "missing" `clos_refl_trans` constructors.

```

Lemma rsc_R :  $\forall (X:\text{Type}) (R:\text{relation } X) (x y : X),$ 
   $R x y \rightarrow \text{clos\_refl\_trans\_1n } R x y.$ 
+

```

Exercise: 2 stars, optional (rsc_trans)

```

Lemma rsc_trans :
   $\forall (X:\text{Type}) (R: \text{relation } X) (x y z : X),$ 
     $\text{clos\_refl\_trans\_1n } R x y \rightarrow$ 
     $\text{clos\_refl\_trans\_1n } R y z \rightarrow$ 
     $\text{clos\_refl\_trans\_1n } R x z.$ 
Proof.
  (* FILL IN HERE *) Admitted.

```

□

Then we use these facts to prove that the two definitions of reflexive, transitive closure do indeed define the same relation.

Exercise: 3 stars, optional (rtc_rsc_coincide)

```
Theorem rtc_rsc_coincide :  
  ∀ (X:Type) (R: relation X) (x y : X),  
    clos_refl_trans R x y ↔ clos_refl_trans_ln R x y.  
Proof.  
  (* FILL IN HERE *) Admitted.
```

□