18.S996: Category Theory for Scientists (David Spivak), Problem Set 3

2.6.1.3

I assume that "if x spends a lot of time thinking about y" is well-defined. This relation is not reflexive, nor symmetric, nor transitive; a love triangle neatly illustrates an example where the relation is not symmetric nor transitive; each person thinks oft of his or her beloved, but not of his or her lover. Alternatively, because "knows about" is neither symmetric nor transitive, this relation is not, either. And not everyone thinks oft of themselves, though I suspect many people do. (If "a lot of time" is sufficiently small, then the relation may be reflexive in practice, and if it is sufficiently large, than it may be symmetric, transitive, and empty, in practice.)

2.6.1.5

Yes. Here's some Coq code which proves that it's an equivalence relation. (It's based on the fact that $=$ is an equivalence relation.)

```
Require Import Setoid Utf8.

Lemma fiber_is_relation X B (f : X → B) : Equivalence (λ x y, f x = f y).
  split; repeat intro; simpl;
  repeat match goal with
           | [ H : context[f _] ⊢ _ ] ⇒ revert H
           | [ ⊢ context[f ?x] ] ⇒ generalize (f x); intro
         end;
  intros; subst; reflexivity.
Qed.
```

Let *sim* be an equivalence relation, and let $\langle x \rangle$ denote the equivalence class of x. Let $f : X \to X/\sim$ be the function defined by $f(x) = \langle x \rangle$. Then $\sim$ is equal to $X \times_f X$.

2.6.1.6

Yes, because isomorphism is an equivalence relation. Here's a Coq proof:

```
Require Import Setoid Utf8 Ensembles.

Set Implicit Arguments.

Definition compose X Y Z (f : Y → Z) (g : X → Y) := λ x, f (g x).

Arguments compose [X Y Z] f g x / .

Infix "∘" := (@compose _ _ _) (at level 70).

Record is_isomorphism X Y (f : X → Y) :=
  {
    isomorphism_inverse : Y → X;
    isomorphism_right_inverse
    : ∀ x, (f ∘ isomorphism_inverse) x = (λ x, x) x;
    isomorphism_left_inverse
    : ∀ x, (isomorphism_inverse ∘ f) x = (λ x, x) x
```

```
  }.
   Notation "i ^-1" := (isomorphism_inverse i) (at level 10).

Record isomorphic X Y :=
  {
    isomorphic_morphism :> . X → Y;
    isomorphic_is_isomorphism :> . is_isomorphism isomorphic_morphism
  }.
Hint Resolve isomorphic_is_isomorphism.

Infix "≅" := (isomorphic) (at level 70).

Notation relation A := (A → A → Type).

Definition Reflexive {A} (R : relation A) := ∀ x, R x x.
Definition Symmetric {A} (R : relation A) := ∀ x y, R x y → R y x.
Definition Transitive {A} (R : relation A) := ∀ x y z, R x y → R y z → R x z.

Record Equivalence {A : Type} (R : relation A)
  := Build_Equivalence
       { Equivalence_Reflexive : Reflexive R;
         Equivalence_Symmetric : Symmetric R;
         Equivalence_Transitive : Transitive R }.

Section isomorphic_relation.
  Local Ltac t :=
    intros; simpl;
    repeat match goal with
             | [ H : _ ≅ _ ⊢ _ ] ⇒ destruct H as [ ? [] ]; simpl in *
             | [ H : _ ⊢ _ ] ⇒ rewrite H
           end;
    reflexivity.

  Definition isomorphic_reflexive : Reflexive isomorphic.
    repeat intro.
    ∃ (λ x, x).
    ∃ (λ x, x);
      t.
  Defined.

  Definition isomorphic_symmetric : Symmetric isomorphic.
    intros x y f.
    eexists (f ^-1).
    ∃ f;
      t.
  Defined.

  Definition isomorphic_transitive : Transitive isomorphic.
    intros x y z f g.
    ∃ (g ∘ f).
    ∃ (f ^-1 ∘ g ^-1);
      t.
  Defined.
```

```
End isomorphic_relation.

Hint Resolve @isomorphic_reflexive @isomorphic_symmetric.
Hint Immediate @isomorphic_reflexive @isomorphic_symmetric @isomorphic_transitive.

Section Exercise_2_6_1_6.
  Variable I : Type.
  Variable X : I → Type.

  Example Exercise_2_6_1_6 : Equivalence (λ i j, X i ≅ X j).
  Proof.
    split; hnf; simpl;
    eauto.
  Defined.
End Exercise_2_6_1_6.
```

2.6.1.9

The equivalence relation is the entire set, i.e., $\forall x\ y,\ x \sim y$. There is one equivalence class. Symmetry gives you $x \sim y$ if $|x - y| = 1$, reflexivity gives you $x \sim x$, i.e., $x \sim y$ if $|x - y| \leq 1$, and transitivity gives you everything else.

2.7.2.2

Yes, $|B^A| = |B|^{|A|}$ so long as we define, by convention, that $0^0 = 1$.

2.7.2.6

The function ev takes in a pair consisting of a function $f : A \to B$ and an element $a \in A$, and returns the element $f(a) \in B$. The name ev can stand for *evaluation*; we are evaluating a function on its argument.

2.7.3.2

Since there is exactly one automorphism of the empty set (the identity function), $\underline{0}^{\underline{0}} \cong \underline{1}$.

2.7.4.2

In general $\mathbb{P}(A) \cong \underline{2}^A$.

(a) 1 (b) 2 (c) 2^6 (d) Because it has $2^{|A|}$ elements, perhaps, and this is an exponential, or "power" operation.

3.1.1.5

The definition $a \star b = a \cdot b$ makes $\mathbb{N}$ into a monoid under multiplication.

3.1.1.6

For any field $\mathbb{F}$, and any vectors space V over $\mathbb{F}$, let $\mathrm{Hom}_{\mathrm{Vect}}(V, V)$ denote the set of linear transformations from V to V, and for any $T, T' : V \to V$, let $T \cdot T'$ be function composition, that is, matrix multiplication. Then there is a monoid $(\mathrm{Hom}_{\mathrm{Vect}}(V, V), \mathrm{id}_V, \cdot)$.

3.1.1.7

There is a non-commutative monoid $(\{1, 0_a, 0_b\}, 1, f)$ where $f(0_a x) = 0_a$ and $f(0_b x) = 0_b$ for all x. Then $f(0_a 0_b) = 0_a$ but $f(0_b 0_a) = 0_b$. All smaller monoids are commutative.

There is a monoid $(\{\heartsuit\}, \heartsuit, (\heartsuit, \heartsuit) \mapsto \heartsuit)$.

Here's some Coq code which implements this:

```
Require Import Utf8 Setoid.

Set Implicit Arguments.

Generalizable All Variables.

Record ComputationalMonoid (M : Type) :=
  {
    monoid_obj :> . _ := M;
    id : monoid_obj;
    comp : monoid_obj → monoid_obj → monoid_obj
  }.

Infix "⋆" := (comp _) (at level 40, left associativity).

Record Monoid (M : Type) :=
  {
    cmonoid :> . ComputationalMonoid M;
    left_id : ∀ x : cmonoid, id _ ⋆ x = x;
    right_id : ∀ x : cmonoid, x ⋆ id _ = x;
    assoc : ∀ a b c : cmonoid, (a ⋆ b) ⋆ c = a ⋆ (b ⋆ c)
  }.

Definition smallest_monoid : Monoid unit.
  eexists (@Build_ComputationalMonoid _ tt (fun _ _ ⇒ tt)); repeat intro;
  repeat match goal with
           | [ H : unit ⊢ _ ] ⇒ destruct H
           | [ ⊢ context[?e] ] ⇒ destruct e; reflexivity
         end.
Defined.

Lemma no_smaller_monoid : Monoid Empty_set → False.
  intro M.
  destruct (id M).
Qed.

Inductive two := twoa | twob.

Inductive three := one | zeroa | zerob.

Definition f : three → three → three :=
  fun x y ⇒ match (x, y) with
```

```
                | (one, z) ⇒ z
                | (z, one) ⇒ z
                | (zeroa, _) ⇒ zeroa
                | (zerob, _) ⇒ zerob
              end.

Definition three_monoid : Monoid three.
  eexists (@Build_ComputationalMonoid _ one f);
  repeat intros []; reflexivity.
Defined.

Definition is_commutative M (M' : Monoid M) := ∀ a b : M', a ⋆ b = b ⋆ a.

Lemma two_monoid_is_commutative (M : Monoid two) : is_commutative M.
  repeat intro;
  pose proof (left_id M);
  pose proof (right_id M);
  set (i := id M) in *; clearbody i;
  hnf in *;
  repeat match goal with
           | [ H := _ ⊢ _ ] ⇒ subst H
           | [ H : two ⊢ _ ] ⇒ destruct H
           | [ H : _ ⊢ _ ] ⇒ rewrite H
         end;
  try reflexivity.
Qed.
```

3.1.1.13

The free monoid generated by $\{\}$ is $(\mathbb{Z}, 0, +)$, and the free monoid generated by $\emptyset$ is the trivial monoid.