```
Require Import PropasTypesUtf8Notation
PropasTypesBasics_mod SwedishSetoids_mod.

Notation "p ⁻¹" := (setoidsym _ _ _ p) (at level 3,
no associativity).
Notation "p ⊙ q" := (setoidtra _ _ _ _ q p) (at level
34, right associativity).
Notation "F ⌜ p" := (setoidmapextensionality _ _ F _
_ p) (at level 100).

Record setoidfamily (A: setoid) :=
  {
    setoidfamilyobj :> A → setoid;
    setoidfamilymap :  ∀x y: A, ∀p: x ≈ y,
                         setoidfamilyobj x ⇒
setoidfamilyobj y;
    setoidfamilyref :  ∀x: A, ∀y: setoidfamilyobj x,
      setoidfamilymap x x (setoidrefl A x) y ≈ y;
    setoidfamilyirr :  ∀x y: A, ∀p q: x ≈ y, ∀z:
setoidfamilyobj x,
                         setoidfamilymap x y p z ≈
setoidfamilymap x y q z;
    setoidfamilycmp :  ∀x y z: A, ∀p: x ≈ y, ∀q: y ≈
z, ∀w: setoidfamilyobj x,
                         (setoidfamilymap y z q)
((setoidfamilymap x y p) w)
                           ≈ setoidfamilymap x z
(q⊙p) w
  }.

Notation "F • p" := (setoidfamilymap _ F _ _ p)
  (at level 9, right associativity).

Lemma setoidfamilyrefgeneral {A: setoid} (F:
setoidfamily A):
  ∀x: A, ∀p: x ≈ x, ∀y: F x, F•p y ≈ y.
```

```
Proof.
  intros x p y.
  apply setoidtra with (F●(setoidrefl A x) y);
  eauto using setoidtra, setoidfamilyirr,
setoidfamilyref, setoidrefl.
Defined.

Lemma setoidfamilycmpgeneral {A: setoid} (F:
setoidfamily A):
  ∀x y z: A, ∀p: x ≈ y, ∀q: y ≈ z, ∀r: x ≈ z, ∀w: F
x, F●q (F●p w) ≈ F●r w.
Proof.
  intros x y z p q r w;
    eauto using setoidtra, setoidfamilyirr,
setoidfamilycmp.
Defined.

Definition A₀: setoid := unit_setoid ⊕ unit_setoid ⊕
unit_setoid.

Definition F₀obj (x: A₀): setoid :=
  match x with
    | inl (inl _) => empty_setoid
    | inl (inr _) => unit_setoid
    | inr _       => nat_setoid
  end.

Definition F₀map (x y: A₀) (p: x ≈ y): F₀obj x ⇒
F₀obj y.
repeat induction x as [x | x]; induction x;
  repeat induction y as [y | y]; induction y; exact
idmap || contradiction p.
Defined.

Definition F₀: setoidfamily A₀.
apply (Build_setoidfamily A₀ F₀obj F₀map).
repeat induction x as [x | x]; induction x;
```

```
swesetoid.
intro y. apply (Identity_refl y).
repeat induction x as [x | x]; induction x;
  repeat induction y as [y | y]; induction y;
    intros; contradiction p || contradiction q ||
swesetoid.
apply (Identity_refl z).
repeat induction x as [x | x]; induction x;
  repeat induction y as [y | y]; induction y;
    repeat induction z as [z | z]; induction z;
      intros; contradiction p || contradiction q ||
swesetoid.
apply (Identity_refl w).
Defined.

(* some extra lemma for handling setoidfamilies *)
Lemma setoidfamilycmpinvert
{A: setoid} (F: setoidfamily A):
  ∀x y: A, ∀p: x ≈ y, ∀q: y ≈ x, ∀w: F x, F●q (F●p w)
≈ w.
Proof.
  intros x y p q w.
  assert  (F ● q (F ● p w) ≈,{ F x } F ● (setoidrefl
A x) w).
  apply setoidfamilycmpgeneral.
  assert (F ● (setoidrefl A x) w ≈,{ F x } w).
  apply setoidfamilyrefgeneral.
  swesetoid.
Defined.

Lemma setoidfamilycmpgeneral_3
{A: setoid} (F: setoidfamily A):
∀ x: A,  ∀ v z: A, ∀ r: x ≈ v, ∀ s: v ≈ z,
   ∀u:F z , ∀w: F x, u ≈  F●(s⊙r) w -> u ≈  F●s (F●r
w).
Proof.
  intros x v z r s u w H.
```

```
  assert (F•s (F•r w) ≈  F•(s⊙r) w).
  apply setoidfamilycmp.
  swesetoid.
Defined.

Lemma setoidfamilyirrgeneral
  {A: setoid} (F: setoidfamily A):
  ∀ x y: A,  ∀ p q: x ≈ y, ∀u v: F x,
  u  ≈ v ->  F•p u ≈ F•q v.
Proof.
  intros x y p q u v H.
  assert (F • p u ≈,{ F y } F • q u).
  apply setoidfamilyirr.
  assert (F • p u ≈,{ F y } F • p v).
  swesetoid. swesetoid.
Defined.

Lemma setoidfamilyirrrevgeneral
  {A: setoid} (F: setoidfamily A):
  ∀ x y: A,  ∀ p q: x ≈ y, ∀u v: F x,
    F•p u ≈ F•q v ->  u  ≈ v.
Proof.
  intros x y p q u v H.
  assert (F • p ⁻¹ (F • p u) ≈  u).
  apply  setoidfamilycmpinvert.
  assert (F • q ⁻¹ (F • q v) ≈ v).
  apply setoidfamilycmpinvert.
  assert (F • q ⁻¹ (F • p u) ≈ F • q ⁻¹ (F • q v)).
  apply setoidmapextensionality.
  apply H.
  assert ( F • p ⁻¹ (F • p u) ≈ F • q ⁻¹ (F • p u)).
  apply setoidfamilyirr.
  swesetoid.
Defined.


Lemma setoidfamilyirrgeneraldouble
```

```
  {A: setoid} (F: setoidfamily A):
  ∀ x y: A, ∀ z w: A,
  ∀ p: x ≈ z,  ∀ p': z ≈ y,
  ∀ q: x ≈ w,  ∀ q': w ≈ y,
  ∀u v: F x,
     u  ≈ v ->  F●p' (F●p u) ≈  F●q' (F●q v).
Proof.
  intros x y z w p p' q q' u v H.
  assert (F ● p' (F ● p u) ≈  F ● (p'⊙ p) u).
  apply setoidfamilycmp.
  assert (F ● q' (F ● q v) ≈  F ● (q'⊙ q) v).
  apply setoidfamilycmp.
  assert (F ● (p'⊙ p) u ≈  F ● (q'⊙ q) v ).
  apply setoidfamilyirrgeneral.
  exact H.
  swesetoid.
Defined.


Theorem arrsolve1 {A B: setoid} (F: setoidfamily A)
(G: setoidfamily B)
: ∀ a b: A,  ∀ c d: B,  ∀ f: F a  ⇒ G c,  ∀ g: F b ⇒
G d,
∀ p: a ≈ b, ∀ q: c ≈ d, (∀ x: F a,
  G●q (f x)  ≈ g (F●p x)) -> (∀ x: F b, g x  ≈  G●q
(f (F●(p⁻¹) x))).
Proof.
  intros a b c d f g p q P x.
  specialize (P (F●(p⁻¹) x)).
  assert (g x ≈ g (F ● p (F ● p ⁻¹ x))) as Q.
  apply setoidmapextensionality.
  apply setoidsym.
  apply setoidfamilycmpinvert.
  swesetoid.
Defined.

(* The set of stages of the construction *)
```

```
Inductive stages: Set :=
  | base   : stages
  | triple : stages → stages → stages → stages.


Definition stages_TrueFalse (n:stages): Set :=
  match n with
    | base   => ⊥
    | triple _ _ _ => ⊤
  end.

Definition stages_pred1 (n:stages):stages :=
  match n with
    | base   => base
    | triple i _ _ => i
  end.


Definition stages_pred2 (n:stages):stages :=
  match n with
    | base   => base
    | triple _ j _ => j
  end.

Definition stages_pred3 (n:stages):stages :=
  match n with
    | base   => base
    | triple _ _ k => k
  end.


Lemma stages_base_not_ind (i j k : stages) :
¬Identity base (triple i j k).
Proof.
  intro P.
  assert (stages_TrueFalse (triple i j k)) as H.
```

```
  apply tt.
  destruct P.
  apply H.
Defined.

Lemma stages_ind_inj_1 (i j k i2 j2 k2 : stages):
Identity (triple i j k) (triple i2 j2 k2) -> Identity
i i2.
Proof.
  intro P.
  apply (Identity_congr stages_pred1 (triple i j k)
(triple i2 j2 k2)).
  apply P.
Defined.

Lemma stages_ind_inj_2 (i j k i2 j2 k2 : stages):
Identity (triple i j k) (triple i2 j2 k2) -> Identity
j j2.
Proof.
  intro P.
  apply (Identity_congr stages_pred2 (triple i j k)
(triple i2 j2 k2)).
  apply P.
Defined.

Lemma stages_ind_inj_3 (i j k i2 j2 k2 : stages):
Identity (triple i j k) (triple i2 j2 k2) -> Identity
k k2.
Proof.
  intro P.
  apply (Identity_congr stages_pred3 (triple i j k)
(triple i2 j2 k2)).
  apply P.
Defined.


Lemma stages_DecId: DecId stages.
```

```
Proof.
  intro x. induction x.
  intro y. induction y.
  left. split.
  right.
  apply stages_base_not_ind.
  intro y.
  destruct y.
  right.
  intro H.
  apply (stages_base_not_ind x1 x2 x3).
  apply Identity_sym.
  apply H.
  specialize (IHx1 y1).
  specialize (IHx2 y2).
  specialize (IHx3 y3).
  elim IHx1.
  intro P1.
  elim IHx2.
  intro P2.
  elim IHx3.
  intro P3.
  left.
  destruct P1.
  destruct P2.
  destruct P3.
  apply Identity_refl.
  intro P3.
  right.
  intro H.
  apply P3.
  apply (stages_ind_inj_3 x1 x2 x3 y1 y2 y3).
  apply H.
  intro P2.
  right.
  intro H.
  apply P2.
```

```
  apply (stages_ind_inj_2 x1 x2 x3 y1 y2 y3).
  apply H.
  intro P1.
  right.
  intro H.
  apply P1.
  apply (stages_ind_inj_1 x1 x2 x3 y1 y2 y3).
  apply H.
Defined.

Definition stages_setoid: setoid.
  apply (Build_setoid stages (fun x y => Identity x
y)).
  intro x. apply Identity_refl.
  intros x y P. induction P. apply Identity_refl.
  intros x y z P Q. induction P. apply Q.
Defined.


Definition starsetoidbase
(A1: setoid) (F1: setoidfamily A1)
(A2: setoid) (F2: setoidfamily A2)
(A3: setoid) (F3: setoidfamily A3)
: Set := ∃a: A1, ∃b: A2, ∃c: A3, ∃d: A3,
   c ≈ d ∧ ((F1 a) ⇒ (F3 c))  ∧ ((F2 b)  ⇒ (F3 d)).

Definition starsetoideq
(A1: setoid) (F1: setoidfamily A1)
(A2: setoid) (F2: setoidfamily A2)
(A3: setoid) (F3: setoidfamily A3)
(x y: starsetoidbase A1 F1 A2 F2 A3 F3):Set.
  destruct x as  [a [b [c [d [q [f g ]]]]]].
  destruct y as  [a' [b' [c' [d' [q' [f' g' ]]]]]].
  apply (∃p1:  a ≈ a', ∃p2:  b ≈ b', ∃p3:  c ≈ c',
∃p4: d ≈ d',
     (F3 • p3) ∘ f ≈ f' ∘ (F1 • p1)  ∧  (F3 • p4) ∘ g
≈ g' ∘ (F2 • p2)).
```

```
Defined.

Definition starsetoid
(A1: setoid) (F1: setoidfamily A1)
(A2: setoid) (F2: setoidfamily A2)
(A3: setoid) (F3: setoidfamily A3):setoid.
  apply (Build_setoid (starsetoidbase A1 F1 A2 F2 A3
F3)
                      (starsetoideq A1 F1 A2 F2 A3
F3)).
  (* Reflexivity *)
  intro x.
  destruct x as  [a [b [c [d [q [f g ]]]]]].
  simpl.
  exists (setoidrefl _ a).
  exists (setoidrefl _ b).
  exists (setoidrefl _ c).
  exists (setoidrefl _ d).
  split.
  intro x.
  assert ( F3 • (setoidrefl A3 c) (f x) ≈ f x ) as
H1.
  apply setoidfamilyrefgeneral.
  assert ( f x ≈,{ F3 c }f (F1 • (setoidrefl A1 a) x)
) as H2.
  apply setoidmapextensionality.
  apply setoidsym.
  apply setoidfamilyrefgeneral.
  swesetoid.
  intro x.
  assert ( F3 • (setoidrefl A3 d) (g x) ≈ g x ) as
H1.
  apply setoidfamilyrefgeneral.
  assert ( g x ≈ g (F2 • (setoidrefl A2 b) x) ) as
H2.
  apply setoidmapextensionality.
  apply setoidsym.
```

```
  apply setoidfamilyrefgeneral.
  swesetoid.
  (* Symmetry *)
  intros x y.
  destruct x as  [a [b [c [d [q [f g ]]]]]].
  destruct y as  [a' [b' [c' [d' [q' [f' g' ]]]]]].
  simpl.
  intro P.
  destruct P as  [p1 [p2 [p3 [p4 [P1 P2 ]]]]].
  exists (setoidsym _ _ _ p1).
  exists (setoidsym _ _ _ p2).
  exists (setoidsym _ _ _ p3).
  exists (setoidsym _ _ _ p4).
  split.
  intro t.
  assert ( F3 • p3 ⁻¹ ( F3 • p3 (f  (F1 • p1 ⁻¹ t)))
≈
         F3 • p3 ⁻¹ (f'  (F1 • p1 (F1 • p1 ⁻¹ t))) )
as H1.
  apply setoidmapextensionality.
  apply P1.
  assert ( F3 • p3 ⁻¹ ( F3 • p3 (f  (F1 • p1 ⁻¹ t)))
≈ f  (F1 • p1 ⁻¹ t) ) as H2.
  apply setoidfamilycmpinvert.
  assert (F3 • p3 ⁻¹ (f' (F1 • p1 (F1 • p1 ⁻¹
t)))  ≈  F3 • p3 ⁻¹ (f' t)) as H3.
  apply setoidmapextensionality.
  apply setoidmapextensionality.
  apply setoidfamilycmpinvert.
  swesetoid.
  intro t.
  assert ( F3 • p4 ⁻¹ ( F3 • p4 (g  (F2 • p2 ⁻¹ t)))
≈
         F3 • p4 ⁻¹ (g'  (F2 • p2 (F2 • p2 ⁻¹ t))) )
as H1.
  apply setoidmapextensionality.
  apply P2.
```

```
  assert ( F3 ● p4 ⁻¹ ( F3 ● p4 (g  (F2 ● p2 ⁻¹ t)))
≈ g  (F2 ● p2 ⁻¹ t) ) as H2.
  apply setoidfamilycmpinvert.
  assert (F3 ● p4 ⁻¹ (g' (F2 ● p2 (F2 ● p2 ⁻¹
t)))  ≈  F3 ● p4 ⁻¹ (g' t)) as H3.
  apply setoidmapextensionality.
  apply setoidmapextensionality.
  apply setoidfamilycmpinvert.
  swesetoid.

  (* Transitivity *)
  intros x y z.
  destruct x as  [a [b [c [d [q [f g ]]]]]].
  destruct y as  [a' [b' [c' [d' [q' [f' g' ]]]]]].
  destruct z as  [a'' [b'' [c'' [d'' [q'' [f'' g''
]]]]]].
  intros P Q.
  simpl.
  destruct P as  [p1 [p2 [p3 [p4 [P1 P2 ]]]]].
  destruct Q as  [q1 [q2 [q3 [q4 [Q1 Q2 ]]]]].
  exists (q1 ⊙ p1).
  exists (q2 ⊙ p2).
  exists (q3 ⊙ p3).
  exists (q4 ⊙ p4).
  split.
  intro t.
  assert (F3 ● q3 (f'  (F1 ● p1 t)) ≈ f'' (F1 ● q1
(F1 ● p1 t))) as H1.
  simpl in Q1.
  apply Q1.
  assert (F3 ● q3 (F3 ● p3 (f t))  ≈ F3 ● q3 (f' (F1
● p1 t))) as H2.
  apply setoidmapextensionality.
  simpl in P1.
  apply P1.
  assert  (F3 ● q3 (F3 ● p3 (f t)) ≈ f'' (F1 ● q1 (F1
● p1 t))) as H3.
```

```
  swesetoid.
  clear H1.
  clear H2.
  assert (F3 • (q3 ⊙ p3) (f t) ≈ F3 • q3 (F3 • p3 (f
t))) as H4.
  apply setoidsym.
  apply setoidfamilycmp.
  assert (f'' (F1 • q1 (F1 • p1 t)) ≈ f'' (F1 • (q1 ⊙
p1) t)) as H5.
  apply setoidmapextensionality.
  apply setoidfamilycmp.
  swesetoid.
  intro t.
  assert (F3 • q4 (g'  (F2 • p2 t)) ≈ g'' (F2 • q2
(F2 • p2 t))) as H1.
  simpl in Q2.
  apply Q2.
  assert (F3 • q4 (F3 • p4 (g t))   ≈ F3 • q4 (g' (F2
• p2 t))) as H2.
  apply setoidmapextensionality.
  simpl in P2.
  apply P2.
  assert  (F3 • q4 (F3 • p4 (g t)) ≈ g'' (F2 • q2 (F2
• p2 t))) as H3.
  swesetoid.
  clear H1.
  clear H2.
  assert (F3 • (q4 ⊙ p4) (g t) ≈ F3 • q4 (F3 • p4 (g
t))) as H4.
  apply setoidsym.
  apply setoidfamilycmp.
  assert (g'' (F2 • q2 (F2 • p2 t)) ≈ g'' (F2 • (q2 ⊙
p2) t)) as H5.
  apply setoidmapextensionality.
  apply setoidfamilycmp.
  swesetoid.
Defined.
```

```
Definition starsetoidfamilyobjbase
  (A1: setoid) (F1: setoidfamily A1)
  (A2: setoid) (F2: setoidfamily A2)
  (A3: setoid) (F3: setoidfamily A3)
  (arg: starsetoid A1 F1 A2 F2 A3 F3): Set.
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  apply ( ∃x: F1 a, ∃y: F2 b, (F3 • q (f x)) ≈ g y).
Defined.


Definition starsetoidfamilyobjeq
  (A1: setoid) (F1: setoidfamily A1)
  (A2: setoid) (F2: setoidfamily A2)
  (A3: setoid) (F3: setoidfamily A3)
  (arg: starsetoid A1 F1 A2 F2 A3 F3)
  (u v: starsetoidfamilyobjbase A1 F1 A2 F2 A3 F3
arg):Set.
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  simpl in u.
  simpl in v.
  apply ((projT1 u ≈ projT1 v) ∧ (projT1 (projT2 u) ≈
projT1 (projT2 v))).
Defined.

Definition starsetoidfamilyobj
  (A1: setoid) (F1: setoidfamily A1)
  (A2: setoid) (F2: setoidfamily A2)
  (A3: setoid) (F3: setoidfamily A3)
  (arg: starsetoid A1 F1 A2 F2 A3 F3): setoid.
  apply (Build_setoid (starsetoidfamilyobjbase A1 F1
A2 F2 A3 F3 arg)
                      (starsetoidfamilyobjeq A1 F1 A2
F2 A3 F3 arg) ).
  (* Reflexivity *)
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
```

```
  intro u.
  destruct u  as  [x [y P]].
  simpl.
  split.
  apply setoidrefl.
  apply setoidrefl.
  (* Symmetry *)
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  intro u.
  destruct u  as  [x [y P]].
  intro v.
  destruct v  as  [x' [y' P']].
  simpl.
  intro H.
  split.
  apply setoidsym.
  apply H.
  apply setoidsym.
  apply H.
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  intro u.
  destruct u  as  [x [y P]].
  intro v.
  destruct v  as  [x' [y' P']].
  intro w.
  destruct w  as  [x'' [y'' P'']].
  simpl.
  intros Q R.
  destruct Q.
  destruct R.
  split.
  swesetoid.
  swesetoid.
Defined.

Definition starsetoidfamilymapunder
  (A1: setoid) (F1: setoidfamily A1)
```

```
  (A2: setoid) (F2: setoidfamily A2)
  (A3: setoid) (F3: setoidfamily A3)
  (arg arg': starsetoid A1 F1 A2 F2 A3 F3) (p: arg ≈
arg')
  : starsetoidfamilyobj A1 F1 A2 F2 A3 F3 arg ->
    starsetoidfamilyobj A1 F1 A2 F2 A3 F3 arg'.
  intro u.
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  destruct arg' as  [a' [b' [c' [d' [q' [f' g'
]]]]]].
  simpl.
  simpl in u.
  destruct p as  [p1 [p2 [p3 [p4 [P1 P2 ]]]]].
  destruct u as [x [y P ]].
  exists (F1 ● p1 x).
  exists (F2 ● p2 y).
  specialize (P2 y).
  specialize (P1 x).
  assert (F3 ● q' (f' (F1 ● p1 x)) ≈ F3 ● q' ((F3 ●
p3 ◦ f) x)) as H.
  apply setoidmapextensionality.
  swesetoid.
  assert (F3 ● q' ((F3 ● p3 ◦ f) x) ≈ F3 ● q' ((f' ◦
F1 ● p1) x)) as H1.
  apply setoidmapextensionality.
  apply P1.
  assert (F3 ● p4 (F3 ● q (f x)) ≈ F3 ● p4 (g y)) as
H2.
  apply setoidmapextensionality.
  apply P.
  assert (F3 ● p4 (F3 ● q (f x)) ≈ F3 ● q' (F3 ● p3
(f x))) as H3.
  apply setoidfamilyirrgeneraldouble.
  apply setoidrefl.
  swesetoid.
Defined.
```

```
Definition starsetoidfamilymap
  (A1: setoid) (F1: setoidfamily A1)
  (A2: setoid) (F2: setoidfamily A2)
  (A3: setoid) (F3: setoidfamily A3)
  (arg arg': starsetoid A1 F1 A2 F2 A3 F3) (p: arg ≈
arg')
  : starsetoidfamilyobj A1 F1 A2 F2 A3 F3 arg ⇒
    starsetoidfamilyobj A1 F1 A2 F2 A3 F3 arg'.
  apply (Build_setoidmap
    (starsetoidfamilyobj A1 F1 A2 F2 A3 F3 arg)
    (starsetoidfamilyobj A1 F1 A2 F2 A3 F3 arg')
    (starsetoidfamilymapunder A1 F1 A2 F2 A3 F3 arg
arg' p)).
  intros u u' P.
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  destruct arg' as  [a' [b' [c' [d' [q' [f' g'
]]]]]].
  simpl in p.
  destruct p as  [p1 [p2 [p3 [p4 [P1 P2 ]]]]].
  destruct u as [x [y Q ]].
  destruct u' as [x' [y' Q' ]].
  simpl in P.
  simpl.
  split.
  apply setoidmapextensionality.
  apply P.
  apply setoidmapextensionality.
  apply P.
Defined.

Definition starsetoidfamily
  (A1: setoid) (F1: setoidfamily A1)
  (A2: setoid) (F2: setoidfamily A2)
  (A3: setoid) (F3: setoidfamily A3):
    setoidfamily (starsetoid A1 F1 A2 F2 A3 F3).

  apply (Build_setoidfamily
```

```
  apply (build_setoidfamily
        (starsetoid A1 F1 A2 F2 A3 F3)
        (starsetoidfamilyobj A1 F1 A2 F2 A3 F3)
        (starsetoidfamilymap A1 F1 A2 F2 A3 F3)).
  intros arg u.
  destruct arg as  [a [b [c [d [q [f g ]]]]]].
  destruct u as [x [y P ]].
  simpl.
  split.
  apply setoidfamilyrefgeneral.
  apply setoidfamilyrefgeneral.

  intros arg arg' p p' z.
  destruct arg as   [a [b [c [d [q [f g ]]]]]].
  destruct arg' as  [a' [b' [c' [d' [q' [f' g'
]]]]]].
  simpl in p.
  simpl in p'.
  simpl in z.
  destruct z as [x [y P ]].
  destruct p as  [p1 [p2 [p3 [p4 [P1 P2 ]]]]].
  destruct p' as  [p1' [p2' [p3' [p4' [P1' P2' ]]]]].
  simpl.
  split.
  apply setoidfamilyirr.
  apply setoidfamilyirr.

  intros arg arg' arg'' p p' z.
  destruct arg as   [a [b [c [d [q [f g ]]]]]].
  destruct arg' as  [a' [b' [c' [d' [q' [f' g'
]]]]]].
  destruct arg'' as  [a'' [b'' [c'' [d'' [q'' [f''
g'' ]]]]]].

  destruct p as  [p1 [p2 [p3 [p4 [P1 P2 ]]]]].
  destruct p' as  [p1' [p2' [p3' [p4' [P1' P2' ]]]]].
  destruct z as [x [y P ]].
  simpl.
```

```
  split.
  apply setoidfamilycmp.
  apply setoidfamilycmp.
Defined.


Fixpoint ifm (i: stages): ∃A: setoid, setoidfamily A
  := match i with
      | base   => existT _ A₀ F₀
      | triple i j k =>
                existT _ (starsetoid _ (projT2 (ifm
i))
                                       _ (projT2 (ifm
j))
                                       _ (projT2 (ifm
k)))
                         (starsetoidfamily  _ (projT2
(ifm i))
                                            _ (projT2
(ifm j))
                                            _ (projT2
(ifm k)))
    end.



Definition omegalimsetoidbase (fam: stages → ∃A,
setoidfamily A): Set := ∃n, projT1 (fam n).



Definition transport (S:Set)(A:S -> Set)(s t: S)
(p:Identity s t): A s -> A t.
  induction p. intro x. exact x.
Defined.
```

```
Definition omegalimsetoideq  (fam: stages → ∃A,
setoidfamily A)
  : (omegalimsetoidbase fam) → (omegalimsetoidbase
fam) → Set.
  intros u v.
  exact  (∃p: Identity (projT1 u) (projT1 v),
         (transport stages (fun k => projT1 (fam k))
(projT1 u) (projT1 v) p
         (projT2 u)) ≈  (projT2 v)).
Defined.

Definition cross_transport (fam: stages → ∃A,
setoidfamily A)(s s':stages)
(p: Identity s s')(a: projT1 (fam s))(x: projT2 (fam
s) a):
projT2 (fam s') (transport stages (fun s=>  projT1
(fam s)) s s' p a).
induction p. exact x.
Defined.


Definition omegalimsetoid (fam: stages → ∃A,
setoidfamily A): setoid.
  apply (Build_setoid (∃n, projT1 (fam n))
(omegalimsetoideq fam)).
  intros [n x]. simpl. exists (Identity_refl n).
apply setoidrefl.
  intros [m x] [n y] [i e]. simpl in i. destruct i.
  exists (Identity_refl m). simpl. simpl in e. apply
setoidsym. assumption.
  intros [m x] [n y] [o z] [i e] [j f]. simpl in i.
destruct i. simpl in e.
  simpl in j. destruct j. simpl in f. exists
(Identity_refl m). exact (f ⊙ e).
Defined.

Definition omegalimsetoidfamilyobj (fam: stages → ∃A,
```

```
Definition omegalimsetoidfamilyobj (fam: stages → ∃A,
setoidfamily A)
  : (omegalimsetoid fam) -> setoid.
  intros [m x]. exact (projT2 (fam m) x).
Defined.

Definition omegalimsetoidfamilymaphelper (fam: stages
→ ∃A, setoidfamily A)
  (u v: omegalimsetoid fam)
  : u ≈ v → omegalimsetoidfamilyobj fam u ->
omegalimsetoidfamilyobj fam v.
  destruct u as [s a]. destruct v  as [s' a']. intros
[p q]. simpl in p. simpl in q.
  intro x.
  exact ((projT2 (fam s')) • q (cross_transport fam s
s' p a x)).
Defined.

Definition omegalimsetoidfamilymap (fam: stages → ∃A,
setoidfamily A)
  (x y: omegalimsetoid fam)(p: x ≈ y):
  omegalimsetoidfamilyobj fam x ⇒
omegalimsetoidfamilyobj fam y.
  apply (Build_setoidmap (omegalimsetoidfamilyobj fam
x)
                         (omegalimsetoidfamilyobj fam
y)
          (omegalimsetoidfamilymaphelper fam x y p)).
  destruct x. destruct y. destruct p as [i e]. simpl
in i. destruct i. simpl in e |- *. intros. apply
setoidmapextensionality. assumption.
Defined.

Definition omegalimsetoidfamily (fam: stages → ∃A,
setoidfamily A)
  : setoidfamily (omegalimsetoid fam).
  apply (Build_setoidfamily (omegalimsetoid fam)
                            (omegalimsetoidfamilyobj
```

```
                                  (omegaLimSetoidFamilyObj
fam)
                                  (omegalimsetoidfamilymap
fam)).
  intros [m x] y. simpl in *. apply setoidfamilyref.
  intros [m x] [n y] [p e] [q f] z. simpl in z.
  simpl omegalimsetoidfamilyobj.
  assert (Identity p q) as H.
  apply hedberg. apply stages_DecId.
  destruct H. simpl in p. destruct p. simpl. apply
setoidfamilyirr.
  intros [m x] [n y] [o z] [i e] [j f] w. simpl in i.
destruct i. simpl in j. destruct j. simpl.
  apply setoidfamilycmp.
Defined.


(* Now use this limit construction to define the
categorical universe *)

Definition U: setoid := omegalimsetoid ifm.
Definition T: setoidfamily U := omegalimsetoidfamily
ifm.



(* Some setoid stuff *)


Definition Jointly_monic_setoidmaps (A B C:setoid)
(h:A ⇒ B)(k:A ⇒ C):=
( ∀x y: A, h x ≈ h y -> k x ≈ k y -> x ≈ y).



(*

   Categories with object equality
```

```
   Categories with object equality
   built from a graded universe

*)

(*
Require Import PropasTypesUtf8Notation
PropasTypesBasics SwedishSetoids.
Require Import ExtensionalUniverse3. *)

Record cat: Type :=
  {
    catobj  :> setoid;
    catarr  : setoid;
    catcms  : setoid;
    catid   : catobj ⇒ catarr;
    catdom  : catarr ⇒ catobj;
    catcod  : catarr ⇒ catobj;
    catfst  : catcms ⇒ catarr;
    catsnd  : catcms ⇒ catarr;
    catcmp  : catcms ⇒ catarr;
    catK1 : ∀x: catobj, catdom (catid x) ≈ x;
    catK2 : ∀x: catobj, catcod (catid x) ≈ x;
    catK3 : ∀u: catcms, catdom (catcmp u) ≈
                    catdom (catfst u);
    catK4 : ∀u: catcms, catcod (catcmp u) ≈
                    catcod (catsnd u);
    catK5 : ∀u v: catcms, catfst u ≈ catfst v ->
                 catsnd u ≈ catsnd v -> u ≈ v;
    catK6 : ∀f g: catarr, catdom f ≈ catcod g ->
     ∃u : catcms, catsnd u ≈ f ∧ catfst u ≈ g;
    catK7 : ∀u: catcms, ∀y: catobj, catfst u ≈ catid
y ->
              catcmp u ≈ catsnd u;
    catK8 : ∀u: catcms, ∀x: catobj, catsnd u ≈ catid
x ->
              catcmp u ≈ catfst u;
```

```
                catcmp u ≈ catfst u,

    catK9 : ∀u v:catcms, ∀ w z: catcms,
       catfst w ≈ catfst v -> catsnd v ≈ catfst u ->
       catsnd u ≈ catsnd z -> catsnd w ≈ catcmp u ->
       catcmp v ≈ catfst z -> catcmp w ≈ catcmp z
  }.

(* Define a compostion predicate
   Comp(f,g,h) - f o g = h *)

Definition Comp {C:cat}(f g h: catarr C): Set :=
  ∃u : catcms C, catfst C u ≈ g ∧ catsnd C u ≈ f
                ∧ catcmp C u ≈ h.

(* Some properties of the Comp predicate *)

Lemma cat_cong_comp (C:cat)(f f' g g' h h': catarr
C):
  f ≈ f' ->  g ≈ g' ->  h ≈ h' ->
      Comp f g h ->  Comp f' g' h'.
Proof.
  intros H1 H2 H3 HC.
  destruct HC as [x p].
  exists x.
  destruct p as  [p1 p2].
  destruct p2 as  [p21 p22].
  split. swesetoid. split. swesetoid. swesetoid.
Defined.

Lemma cat_unique_comp (C:cat)(f g h k :catarr C):
Comp f g h -> Comp f g k -> h ≈ k.
Proof.
  intros H1 H2.
  destruct H1 as [x1 p1]. destruct H2 as [x2 p2].
  destruct p1 as [p11 p12]. destruct p12 as [p121
p122].
  destruct p2 as [p21 p22]. destruct p22 as [p221
p222].
```

```
p222].
  assert (x1 ≈ x2).
  apply (catK5 C).
  swesetoid. swesetoid. swesetoid.
Defined.

Lemma cat_comp_exists (C:cat)(f g:catarr C):
  catdom C f  ≈ catcod C g -> ∃h:catarr C, Comp f g
h.
Proof.
  intro H.
  assert (∃u, catsnd C u ≈ f ∧ catfst C u ≈ g) as H1.
  apply catK6.
  assumption.
  destruct H1 as  [x p].
  exists (catcmp C x).
  destruct p as  [p1 p2].
  exists x.
  split.
  assumption.
  split.
  swesetoid. swesetoid.
Defined.

Lemma cat_associativity (C:cat)(f g h k l m n: catarr
C):
  Comp f g k -> Comp g h l -> Comp k h m -> Comp f l
n
    -> m ≈ n.
Proof.
  intros H1 H2 H3 H4.
  destruct H4 as [x4 p4]. destruct H3 as [x3 p3].
  destruct H2 as [x2 p2]. destruct H1 as [x1 p1].
  destruct p1 as [p11 p12]. destruct p12 as [p121
p122].
  destruct p2 as [p21 p22]. destruct p22 as [p221
p222].
  destruct p3 as [p31 p32]. destruct p32 as [p321
```

```
  destruct p3 as [p31 p32]. destruct p32 as [p321
p322].
  destruct p4 as [p41 p42]. destruct p42 as [p421
p422].
  assert (catcmp C x3  ≈ catcmp C x4).
  apply (catK9 C) with x1 x2.
  swesetoid. swesetoid.
  swesetoid. swesetoid.
  swesetoid. swesetoid.
Defined.


Lemma cat_id_right (C:cat)(f: catarr C):
  Comp f (catid C (catdom C f)) f.
Proof.
  assert (∃u, catsnd C u ≈ f ∧ catfst C u ≈  (catid C
(catdom C f))).
  apply (catK6 C).
  apply setoidsym.
  apply (catK2 C).
  destruct H as [x p].
  destruct p as [p1 p2].
  exists x. split.
  assumption. split. assumption.
  assert ((catcmp C x) ≈ (catsnd C x)).
  apply (catK7 C x (catdom C f)).
  assumption. swesetoid.
Defined.

Lemma cat_id_left (C:cat)(f: catarr C):
  Comp (catid C (catcod C f)) f f.
Proof.
  assert (∃u, catsnd C u ≈  (catid C (catcod C f)) ∧
catfst C u ≈ f).
  apply (catK6 C).
  apply (catK1 C).
  destruct H as [x p]. destruct p as [p1 p2].
  exists x. split.
```

```
  exists x. split.
  assumption. split. assumption.
  assert ((catcmp C x) ≈ (catfst C x)).
  apply (catK8 C x (catcod C f)).
  assumption. swesetoid.
Defined.

(* Predicate for being a commutative square *)

Definition Square {C:cat}(f g h k: catarr C):Set :=
  ∃p: catarr C, Comp f h p  ∧  Comp g k p.


Definition Square' {C:cat}(f g h k: catarr C):Set :=
  ∃u : catcms C, ∃v : catcms C,
  catfst C u ≈ h ∧ catsnd C u ≈ f ∧
  catfst C v ≈ k ∧ catsnd C v ≈ g ∧ catcmp C u ≈
catcmp C v.

Theorem Square_to_Square' {C:cat}(f g h k: catarr C):
Square f g h k -> Square' f g h k.
Proof.
intro P.
destruct P as [p  [P1 P2]].
destruct P1 as [u  [P11  [P12  P13]]].
destruct P2 as [v  [P21  [P22  P23]]].
exists u.
exists v.
split.
apply P11.
split.
apply P12.
split.
apply P21.
split.
apply P22.
swesetoid.
Defined.
```

```
Defined.


Theorem Square'_to_Square {C:cat}(f g h k: catarr C):
Square' f g h k -> Square f g h k.
Proof.
intro P.
destruct P as [u  [v  [P1  [P2  [P3  [P4  P5]]]]]].
exists ((catcmp C) u).
split.
exists u.
split. apply P1. split. apply P2. apply setoidrefl.
exists v.
split. apply P3. split. apply P4. swesetoid.
Defined.


(* Define a compostion predicate
   Comp(f,g,h) - f o g = h

Definition Comp {C:cat}(f g h: catarr C): Set :=
  ∃u : catcms C, catfst C u ≈ g ∧ catsnd C u ≈ f
                ∧ catcmp C u ≈ h. *)

Definition Jointly_monic {C:cat}(h k: catarr C):Set
:=
   catdom C h ≈ catdom C k  ∧
   ∀s t:catarr C, catdom C s ≈ catdom C t  ∧ catcod C
s ≈ catcod C t  ->
   ∀r r':catarr C,
   Comp h s r ∧ Comp h t r ∧ Comp k s r' ∧ Comp k t
r'
     -> s ≈ t.


(* Predicate for being a pullback *)
```

```
Definition Pullback {C:cat}(f g h k: catarr C):Set :=
  Square' f g h k  ∧
  Jointly_monic h k  ∧
  ∀h' k':catarr C, Square' f g h' k' → 
    ∃p:catarr C, Comp h p h' ∧ Comp k p k'.



(* Predicate for being an arrow between specified
objects *)


Definition Mor {C:cat} (a b:catobj C)(f: catarr
C):Set :=
catdom C f ≈ a ∧ catcod C f ≈ b.

(* Predicate for being a terminal object *)

Definition Terminal {C:cat}(t: catobj C): Set :=
  ∀a:catobj C, ∃f:catarr C, Mor a t f  ∧
  ∀f':catarr C, Mor a t f'  → f ≈ f'.


(*

       The construction of  a category from a
       proof-irrelevant family of setoids.

*)




(* Build the set or arrows from a family *)
```

```
Definition arr_from_family {A:setoid} (F:
setoidfamily A): setoid.
  apply
  (Build_setoid (∃a: A, (∃b: A, F a ⇒ F b))
   (λ p q, match (p, q) with
    (existT a (existT b f), existT a' (existT b' f'))
=>
          ∃e: a ≈ a',  ∃d: b ≈ b',  ∀x:(F a),
          F●d (f x)  ≈ f' (F●e x)
               end)).
  intro x. destruct x as [a s]. destruct s as [b f].
  exists (setoidrefl _ _).
  exists (setoidrefl _ _).
  intro x.
  assert ( f x ≈,{ F b }f (F ● (setoidrefl A a) x)).
  apply setoidmapextensionality.
  apply setoidsym.
  apply setoidfamilyrefgeneral.
  assert (F ● (setoidrefl A b) (f x) ≈,{ F b } (f
x)).
  apply setoidfamilyrefgeneral.
  swesetoid.

  intro x. destruct x as [a s]. destruct s as [b f].
  intro y. destruct y as [a' s']. destruct s' as [b'
f'].
  intro H.
  destruct H as [e s]. destruct s as [d H2].
  exists e⁻¹.
  exists d⁻¹.
  intro x.
  apply setoidsym.
  assert (F ● d (f (F ● e ⁻¹ x)) ≈,{ F b' }f'
                              (F ● e (F ● e ⁻¹ x))).
  apply H2.
  assert (F ● d⁻¹ (F ● d (f (F ● e ⁻¹ x))) ≈,{ F b }
                       F ● d⁻¹ (f' (F ● e (F ● e ⁻¹
```

```
x)))).
  apply setoidmapextensionality.
  assumption.
  assert (F • d ⁻¹ (F • d (f (F • e ⁻¹ x))) ≈,{ F b
}  f (F • e ⁻¹ x)).
  assert (F • d ⁻¹ (F • d (f (F • e ⁻¹ x))) ≈,{ F b
}  F • (setoidrefl A b) (f (F • e ⁻¹ x))).
  apply setoidfamilycmpgeneral.
  assert (F • (setoidrefl A b) (f (F • e ⁻¹ x)) ≈,{ F
b } f (F • e ⁻¹ x)).
  apply setoidfamilyref.
  swesetoid.
  assert  (f (F • e ⁻¹ x) ≈,{ F b } F • d ⁻¹ (f' (F •
e (F • e ⁻¹ x)))).
  swesetoid.
  assert (F • d ⁻¹ (f' (F • e (F • e ⁻¹ x))) ≈,{ F b
}F • d ⁻¹ (f' x)).
  apply setoidmapextensionality.
  apply setoidmapextensionality.
  assert (F • e (F • e ⁻¹ x) ≈,{ F a' }  F •
(setoidrefl A a') x).
  apply setoidfamilycmpgeneral.
  assert (F • (setoidrefl A a') x  ≈,{ F a' } x).
  apply setoidfamilyref.
  swesetoid. swesetoid.


  intro x. destruct x as [a s]. destruct s as [b f].
  intro y. destruct y as [a' s']. destruct s' as [b'
f'].
  intro z. destruct z as [a'' s'']. destruct s'' as
[b'' f''].

  intro H.
  destruct H as [e s].
  destruct s as [d H].
```

```
  intro H'.
  destruct H' as [e' s'].
  destruct s' as [d' H'].
  exists (e'⊙e).
  exists (d'⊙d).
  intro x.
  apply setoidtra with  (F • d' ((F • d) (f x))).
  apply setoidsym.
  apply setoidfamilycmp.
  apply setoidtra with (f'' (F • e' ((F • e) x))).
  swesetoid.
  apply setoidmapextensionality.
  apply setoidfamilycmp.
Defined.

(* The id-map on a setoid *)

Definition idmapp (A: setoid): setoidmap A A.
  apply (Build_setoidmap A A (λ x: A, x)); swesetoid.
Defined.

(* The id-map on a setoid as an arrow *)

Definition catid_from_family_helper {A:setoid} (F:
setoidfamily A)(a: A) : arr_from_family F.
  exists a.
  exists a.
  apply (idmapp (F a)).
Defined.

(* The id creator in the category *)

Definition catid_from_family
  {A:setoid} (F: setoidfamily A):
           A  ⇒ arr_from_family F.
  apply (Build_setoidmap A (arr_from_family F)
  (λ a:A, catid_from_family_helper F a)).
```

```
  intros x y H.
  exists H. exists H.
  intro t.
swesetoid.
Defined.

(* The domain map in the category *)

Definition catdom_from_family_helper
{A:setoid}(F: setoidfamily A)(f:arr_from_family F):
A.
  destruct f as [a s].
  exact a.
Defined.

Definition catdom_from_family
  {A:setoid} (F: setoidfamily A): arr_from_family
F  ⇒ A.
  apply (Build_setoidmap  (arr_from_family F) A
        (λ a, catdom_from_family_helper F a)).
  intros x y H.
  destruct x as [a s].
  destruct s.
  destruct y as [b t].
  destruct t.
  destruct H.
  swesetoid.
Defined.

(* The codomain map in the category *)

Definition catcod_from_family_helper
  {A:setoid} (F: setoidfamily A)(f:arr_from_family
F): A.
  destruct f as [a s].
  destruct s as [b t].
  exact b.
```

```
Defined.


Definition catcod_from_family
  {A:setoid} (F: setoidfamily A): arr_from_family
F  ⇒ A.
  apply (Build_setoidmap  (arr_from_family F) A
        (λ a, catcod_from_family_helper F a)).
  intros x y H.
  destruct x as [a s]. destruct s.
  destruct y as [b t]. destruct t.
  destruct H as [p H]. destruct H as [q H].
  swesetoid.
Defined.


(* Construction of the setoid of composable arrows
   These are pairs

   (g f) : s.t. cod g =  dom f

*)

Definition cms_from_family
  {A:setoid} (F: setoidfamily A): setoid.
  apply (Build_setoid (∃g: arr_from_family F,
                       ∃f: arr_from_family F,
                       catcod_from_family F g ≈
                       catdom_from_family F f)
      (λ p q, match (p, q) with
            (existT g (existT f p),
             existT g' (existT f' p'))
               => g ≈ g' ∧ f ≈ f'
             end)).
  intro H. destruct H as  [g H]. destruct H as  [f
H].
  split.
  apply setoidrefl.
```

```
  apply setoidrefl.

  intro H. destruct H as  [g H]. destruct H as  [f
H].
  intro H1. destruct H1 as  [g1 H1]. destruct H1
as  [f1 H1].
  intro H2.
  destruct H2 as [H21 H22].
  split.
  apply setoidsym. assumption.
  apply setoidsym. assumption.


  intro H1. destruct H1 as  [g1 H1]. destruct H1
as  [f1 H1].
  intro H2. destruct H2 as  [g2 H2]. destruct H2
as  [f2 H2].
  intro H3. destruct H3 as  [g3 H3]. destruct H3
as  [f3 H3].
  intro H4. destruct H4 as  [H41 H42].
  intro H5. destruct H5 as  [H51 H52].
  split.
  apply setoidtra with g2. assumption. assumption.
  apply setoidtra with f2. assumption. assumption.
Defined.

(* Map picking out the first map of a composable pair
*)

Definition catfst_from_family_helper
  {A:setoid} (F: setoidfamily A)(u: cms_from_family
F):
    arr_from_family F.
  destruct u as  [g u]. destruct u as  [f u].
  exact g.
Defined.
```

```
Definition catfst_from_family
  {A:setoid} (F: setoidfamily A):
    cms_from_family F  ⇒  arr_from_family F.
  apply (Build_setoidmap
        (cms_from_family F)
        (arr_from_family F)
        (λ u,  catfst_from_family_helper F u)).
  intros u v H.
  destruct u as  [g u]. destruct u as  [f u].
  destruct v as  [g' v]. destruct v as  [f' v].
  destruct H as [H1 H2].
  exact H1.
Defined.

(* Map picking out the second map of a composable
pair *)

Definition catsnd_from_family_helper
  {A:setoid} (F: setoidfamily A)(u: cms_from_family
F):
     arr_from_family F.
  destruct u as  [g u]. destruct u as  [f u].
  exact f.
Defined.

Definition catsnd_from_family
  {A:setoid} (F: setoidfamily A):
    cms_from_family F  ⇒  arr_from_family F.
  apply (Build_setoidmap
        (cms_from_family F)
        (arr_from_family F)
        (λ u,  catsnd_from_family_helper F u)).
  intros u v H.
  destruct u as  [g u]. destruct u as  [f u].
  destruct v as  [g' v]. destruct v as  [f' v].
  destruct H as [H1 H2].
  exact H2
```

```
  exact H2.
Defined.

(* The composition map in the category *)

Definition catcmp_from_family_helper
  {A:setoid} (F: setoidfamily A) (u: cms_from_family
F):
    arr_from_family F.
  destruct u as  [g u]. destruct u as  [f p].
  destruct g as  [a s]. destruct s as  [b g].
  destruct f as  [c s]. destruct s as  [d f].
  exists a.
  exists d.
  apply (comp _ f (comp _ (F•p) g)).
Defined.

Definition catcmp_from_family
  {A:setoid} (F: setoidfamily A):
    cms_from_family F  ⇒  arr_from_family F.
  apply (Build_setoidmap
        (cms_from_family F)
        (arr_from_family F)
        (λ u,  catcmp_from_family_helper F u)).
  intros u u'.
  destruct u as  [g u]. destruct u as  [f p].
  destruct u' as  [g' u']. destruct u' as  [f' p'].
  destruct f as  [a f]. destruct f as  [b f].
  destruct f' as  [a' f']. destruct f' as  [b' f'].
  destruct g as  [c g]. destruct g as  [d g].
  destruct g' as  [c' g']. destruct g' as  [d' g'].
  simpl.
  intro H.
  destruct H as [H1 H2].
  destruct H1 as [q H1]. destruct H1 as [r H1].
  destruct H2 as [s H2]. destruct H2 as [t H2].
  exists q. exists t.
  intro x.
```

```
  assert (F • t (f (F • p (g x))) ≈,{ F b' }
         f' (F • s (F • p ( g x )))).
  apply H2.
  assert  (f' (F • p' ( F • r (g x)))  ≈,{ F b' }
         f' (F • p' (g' (F • q x)))).
  apply setoidmapextensionality.
  apply setoidmapextensionality.
  apply H1.
  assert (f' (F • s (F • p (g x)))  ≈,{ F b' }
          f' (F • p' (F • r (g x))) ).
  apply setoidmapextensionality.
  assert ((F • s (F • p (g x))) ≈,{ F a' } F • (s⊙p)
(g x)).
  apply setoidfamilycmp.
  assert ((F • p' (F • r (g x))) ≈,{ F a' } F •
(p'⊙r) (g x)).
  apply setoidfamilycmp.
  assert (F • (p'⊙r) (g x) ≈,{ F a' } F • (s⊙p) (g
x)).
  apply setoidfamilyirr.
  swesetoid.
  swesetoid.
Defined.

Definition cms_from_family_builder
  {A:setoid} (F: setoidfamily A)
  (g f : arr_from_family F)
  (p: catcod_from_family F g ≈ catdom_from_family F
f):
  cms_from_family F.
  exists g.
  exists f.
  assumption.
Defined.

(*
```

```
  Now combine all the above components

  to build the category and prove that they
  satisfies the laws.

  *)

Definition cat_from_family
  {A:setoid} (F: setoidfamily A): cat.
  apply (Build_cat A (arr_from_family F)
                     (cms_from_family F)
                     (catid_from_family F)
                     (catdom_from_family F)
                     (catcod_from_family F)
                     (catfst_from_family F)
                     (catsnd_from_family F)
                     (catcmp_from_family F)
        ).

  (* K1 *)
  intro x. apply setoidrefl.
  (* K2 *)
  intro x. apply setoidrefl.
  (* K3 *)
  intro u. destruct u as [g u]. destruct u as [f p].
  destruct g as [c g]. destruct g as [d g].
  destruct f as [a f]. destruct f as [b f].
  simpl.
  apply setoidrefl.
  (* K4 *)
  intro u. destruct u as [g u]. destruct u as [f p].
  destruct g as [c g]. destruct g as [d g].
  destruct f as [a f]. destruct f as [b f].
  simpl.
  apply setoidrefl.
  (* K5 *)
  intros u u'.
  intros H1 H2.
```

```
  destruct u as  [g  u]. destruct u as [f p].
  destruct u' as [g' u']. destruct u' as [f' p'].
  swesetoid.
  (* K6 *)
  intros f g.
  intro H.
  assert ((catcod_from_family F) g ≈,{ A }
(catdom_from_family F) f) as H1.
  apply setoidsym.
  apply H.
  exists (cms_from_family_builder F g f H1).
  split. apply setoidrefl. apply setoidrefl.
  (* K7 *)
  intro u.
  intro y.
  intro H.
  destruct u as  [g u]. destruct u as [f p].
  destruct g as  [c g].  destruct g as [d g].
  destruct f as  [a f].  destruct f as [b f].
  (* p: d = a *)
  destruct H as [e H]. destruct H as [k H].
  simpl.
  assert ( c ≈,{ A }a) as H2.
  swesetoid.
  exists H2.
  assert (b  ≈,{ A } b) as H3.
  apply setoidrefl.
  exists H3.
  intro x.
  assert (∀x : F c, F ● k (g x) ≈,{ F y } F ● e x) as
H5.
  apply H.
  clear H.
  assert  (F ● H3 (f (F ● p (g x)))  ≈,{ F b }
  f (F ● p (g x))) as H6.
  apply setoidfamilyrefgeneral.
  assert (f (F ● p (a x))  ≈,{ F b } f (F ● H2 x)) as
```

```
H7.

  apply setoidmapextensionality.
  assert (y ≈,{ A } a) as q.
  swesetoid.
  assert (F • q ( F • k (g x)) ≈,{ F a } F • q (F • e
x)) as H8.
  apply setoidmapextensionality.
  apply H5.
  clear H5.
  assert (F • q (F • k (g x)) ≈,{ F a }  F • p (g
x)).
  apply setoidfamilycmpgeneral.
  assert (F • q (F • e x) ≈,{ F a } F • H2 x).
  apply setoidfamilycmpgeneral.
  swesetoid.
  swesetoid.
  (* K8 *)
  intro u.
  intro y.
  intro H.
  destruct u as  [g u]. destruct u as [f p].
  destruct g as  [c g].  destruct g as [d g].
  destruct f as  [a f].  destruct f as [b f].
  (* p: d = a *)
  destruct H as [e H].
  destruct H as [k H].
  simpl.
  assert ( c ≈,{ A }c) as H2.
  apply setoidrefl.
  exists H2.
  assert (b  ≈,{ A } d) as H3.
  apply setoidtra with y.
  assumption.
  apply setoidsym.
  swesetoid.
  exists H3.
  intro x.
```

```
  assert (g (F • H2 x) ≈,{ F d } g x) as H0.

  apply setoidmapextensionality.
  apply setoidfamilyrefgeneral.
  assert (F • H3 (f (F • p (g x))) ≈,{ F d } g x) as
H4.
  assert (F • k (f (F • p (g x))) ≈,{ F y }(F • e (F
• p (g x)))) as H5.
  apply H.
  clear H.
  assert (y ≈,{ A } d) as H6.
  swesetoid.
  assert ( F • H3 (f (F • p (g x))) ≈,{ F d }
   F • H6 (F • k (f (F • p (g x))))) as H7.
  apply setoidsym.
  apply setoidfamilycmpgeneral.
  assert (F • H6 (F • k (f (F • p (g x)))) ≈,{ F d }
   (F • H6 (F • e (F • p (g x))))) as H8.
  apply setoidmapextensionality.
  assumption.
  assert ( F • H6 (F • e (F • p (g x))) ≈,{ F d } g x
) as H9.
  assert ( F • H6 (F • e (F • p (g x)))  ≈,{ F d }
   F • (H6  ⊙ e) (F • p (g x))) as HA.
  apply setoidfamilycmpgeneral.
  assert (F • (H6 ⊙ e) (F • p (g x)) ≈,{ F d }g x).
  apply setoidfamilycmpinvert.
  swesetoid.
  swesetoid.
  swesetoid.
  (* K9 *)
  intros u v w z.
  destruct u as  [u1 u]. destruct u as [u2 up].
  destruct u1 as  [u1d u1]. destruct u1 as [u1c u1].
  destruct u2 as  [u2d u2]. destruct u2 as [u2c u2].
  destruct v as  [v1 v]. destruct v as [v2 vp].
  destruct v1 as  [v1d v1]. destruct v1 as [v1c v1].
  destruct v2 as  [v2d v2]. destruct v2 as [v2c v2].
```

```
  destruct w as  [w1 w]. destruct w as [w2 wp].

  destruct w1 as  [w1d w1]. destruct w1 as [w1c w1].
  destruct w2 as  [w2d w2]. destruct w2 as [w2c w2].
  destruct z as  [z1 z]. destruct z as [z2 zp].
  destruct z1 as  [z1d z1]. destruct z1 as [z1c z1].
  destruct z2 as  [z2d z2]. destruct z2 as [z2c z2].
  simpl.
  intros H1 H2 H3 H4 H5.
  destruct H1 as [p1 H1]. destruct H1 as [q1 H1].
  destruct H2 as [p2 H2]. destruct H2 as [q2 H2].
  destruct H3 as [p3 H3]. destruct H3 as [q3 H3].
  destruct H4 as [p4 H4]. destruct H4 as [q4 H4].
  destruct H5 as [p5 H5]. destruct H5 as [q5 H5].
  assert (w1d ≈,{ A }z1d) as p6.
  swesetoid.
  exists p6.
  assert (w2c ≈,{ A }z2c) as q6.
  swesetoid.
  exists q6.
  intro x.

  assert  (∀x, F ● q5 (v2 (F ● vp (v1  (F ● p5 ⁻¹ x)
))) ≈,{ F z1c }z1 x) as H5'.
  intro x0.
  (* Fin *)
  assert (F ● q5 (v2 (F ● vp (v1 (F ● p5 ⁻¹ x0)))) ≈,
{ F z1c } z1 (F ● p5 (F ● p5 ⁻¹ x0))) as H5''.
  apply H5.
  assert (z1 (F ● p5 (F ● p5 ⁻¹ x0))  ≈,{ F z1c }z1
x0).
  apply setoidmapextensionality.
  apply setoidfamilycmpinvert.
  swesetoid.
  clear H5.

  assert ( ∀x : F w2d, w2 x ≈
                  F ● q4 ⁻¹ ( u2 (F ● up (u1 (F ● p4
```

```
x)))) ) as H4'.
  intro x0.
  assert ( ∀x : F w2d, F • q4 ⁻¹ (F • q4 (w2 x)) ≈
     F • q4 ⁻¹ ( u2 (F • up (u1 (F • p4 x)))) ).
  intro x1.
  apply setoidmapextensionality.
  apply H4.
  assert (w2 x0 ≈,{ F w2c } F • q4 ⁻¹ (F • q4 (w2
x0))).
  apply setoidsym.
  apply setoidfamilycmpinvert.
  swesetoid.
  clear H4.

  assert (∀x, F • q3 (u2 (F • p3 ⁻¹ x)) ≈,{ F z2c }z2
x  ) as H3'.

  assert (∀x, F • q3 (u2 (F • p3 ⁻¹ x)) ≈,{ F z2c
}z2  (F • p3 (F • p3 ⁻¹ x))).
  intro x0.
  apply H3.
  intro x0.
  assert ( z2 (F • p3 (F • p3 ⁻¹ x0)) ≈ z2 x0).
  apply setoidmapextensionality.
  apply setoidfamilycmpinvert.
  swesetoid.
  clear H3.
  assert (∀x : F w1d, w1 x ≈ F • q1 ⁻¹ (v1 (F • p1
x))) as H1'.
  intro x0.
  assert (w1 x0 ≈,{ F w1c }F • q1 ⁻¹ (F • q1 ( w1
x0))).
  apply setoidsym.
  apply setoidfamilycmpinvert.
  swesetoid.
  clear H1.
  assert (F • q6 (w2 (F • wp (w1 x))) ≈ { F z2c }
```

```
  assert (F • q6 (w2 (F • wp (w1 x))) ≈,{ F z2c }
F • q6 (F • q4 ⁻¹ (u2 (F • up (u1 (F • p4 (F • wp (w1
x)))))))) as H6.
  apply setoidmapextensionality.
  apply H4'.
  clear H4'.
  assert (z2 (F • zp (z1 (F • p6 x)))
≈,{ F z2c }  F • q3 (u2 (F • p3 ⁻¹ (F • zp (z1 (F •
p6 x))))))
as H7.
  apply setoidsym.
  apply H3'.
  clear H3'.
  assert ((F • up (u1 (F • p4 (F • wp (w1 x)))))
≈,{ F u2d }  (F • p3 ⁻¹ (F • zp (z1 (F • p6 x))))) as
H8.

  assert (
F • p3 ⁻¹ (F • zp (F • q5 (v2 (F • vp (v1 (F • p5
⁻¹  (F • p6 x)))))))
≈,{ F u2d }
   F • p3 ⁻¹ (F • zp (z1 (F • p6 x)))) as H9.
  apply setoidmapextensionality.
  apply setoidmapextensionality.
  simpl.
  apply H5'.

  assert ( F • up (u1 (F • p4 (F • wp (w1 x)))) ≈,{ F
u2d }
F • p3 ⁻¹ (F • zp (F • q5 (v2 (F • vp (v1 (F • p5 ⁻¹
(F • p6 x)))))))).
  clear H9.
  assert (
F • up (u1 (F • p4 (F • wp (w1 x)))) ≈,{ F u2d }
F • up (u1 (F • p4 (F • wp (F • q1 ⁻¹ (v1 (F • p1
x)))))) as
HA.
  apply setoidmapextensionality
```

```
  apply setoidmapextensionality.
  apply setoidmapextensionality.

  apply setoidmapextensionality.
  apply setoidmapextensionality.
  swesetoid.
  assert (
F • up (u1 (F • p4 (F • wp (F • q1 ⁻¹ (v1 (F • p1
x))))))
≈,{ F u2d }
   F • p3 ⁻¹ (F • zp (F • q5 (v2 (F • vp (v1 (F • p5
⁻¹ (F • p6 x)))))))) as HB.
  assert (  ∀x : F v2d,  v2 x ≈  F • q2 ⁻¹( u1 (F •
p2 x)) ) as H2'.
  intro x0.
  assert (v2 x0 ≈ F • q2 ⁻¹(F • q2 (v2 x0))).
  apply setoidsym.
  apply setoidfamilycmpinvert.
  assert (
F • q2 ⁻¹ (F • q2 (v2 x0))
≈,{ F v2c }F • q2 ⁻¹ (u1 (F • p2 x0))).
  apply setoidmapextensionality.
  apply H2.
  swesetoid.
  assert ( F • up (u1 (F • p4 (F • wp (F • q1 ⁻¹ (v1
(F • p1 x)))))) ≈,{
   F u2d }
F • p3 ⁻¹ (F • zp (F • q5 (F • q2 ⁻¹ (u1 (F • p2 (F •
vp (v1 (F • p5 ⁻¹ (F • p6 x)))))))))).
  apply setoidfamilycmpgeneral_3.
  apply setoidfamilycmpgeneral_3.
  apply setoidfamilycmpgeneral_3.
  apply setoidfamilyirrgeneral.
  apply setoidmapextensionality.
  apply setoidfamilycmpgeneral_3.
  apply setoidsym.
  apply setoidfamilycmpgeneral_3.
  apply setoidfamilycmpgeneral_3.
  apply setoidfamilyirrgeneral
```

```
  apply setoidfamilyirrgeneral.
  apply setoidmapextensionality.

  apply setoidfamilycmpgeneral.
  swesetoid.
  swesetoid.
  swesetoid.

  assert (F • q6 (F • q4 ⁻¹ (u2 (F • up (u1 (F • p4
(F • wp (w1 x)))))))) ≈
F • q3 (u2 (F • p3 ⁻¹ (F • zp (z1 (F • p6 x)))))).
  apply setoidsym.
  apply setoidfamilycmpgeneral_3.
  apply setoidfamilyirrgeneral.
  apply setoidmapextensionality.
  swesetoid.
  swesetoid.
Defined.

Definition mapof_general
(A:setoid)(F:setoidfamily A)(f: catarr
(cat_from_family F)):
F (catdom (cat_from_family F) f) ⇒  F (catcod
(cat_from_family F) f).
destruct f as  [a  [b f]].
exact f.
Defined.

Theorem Jointly_monic_special (A:setoid)
(F:setoidfamily A)
(h k: catarr (cat_from_family F))
(p: (catdom (cat_from_family F) h) ≈ (catdom
(cat_from_family F) k)):
  Jointly_monic_setoidmaps
     (F (catdom (cat_from_family F) h))
     (F (catcod (cat_from_family F) h))
     (F (catcod (cat_from_family F) k))
     (mapof_general A F h) ((mapof_general A F k) ∘
```

```
(h ● p))
-> Jointly_monic h k.
Proof.
intro H.
unfold Jointly_monic.
split.
apply p.
intros s t  [Q1 Q2].
intros r1 r2  [R1  [R2 [R3 R4]]].
destruct R1 as  [u1  [R11 [R12 R13]]].
destruct R2 as  [u2  [R21 [R22 R23]]].
destruct R3 as  [u3  [R31 [R32 R33]]].
destruct R4 as  [u4  [R41 [R42 R43]]].
assert ((catcmp (cat_from_family F)) u1 ≈ (catcmp
(cat_from_family F)) u2) as R1323.
apply ((R23 ⁻¹) ⊙ R13).
clear R13.
clear R23.
clear r1.
assert ((catcmp (cat_from_family F)) u3 ≈ (catcmp
(cat_from_family F)) u4) as R3343.
apply ((R43 ⁻¹) ⊙ R33).
clear R33.
clear R43.
clear r2.
destruct h as [hd  [hc h]].
destruct k as [kd  [kc k]].
simpl in p.
unfold Jointly_monic_setoidmaps in H.
simpl in H.
destruct s as [sd  [sc s]].
destruct t as [td  [tc t]].
simpl in Q1.
simpl in Q2.
simpl.

destruct u1 as [[u11d  [u11c u11]]  [[u12d  [u12c
u12]] p1]]
```

```
u12]] p1]].
simpl in R11.

simpl in R12.
simpl in p1.
destruct u2 as [[u21d  [u21c u21]]  [[u22d  [u22c
u22]] p2]].
simpl in R21.
simpl in R22.
simpl in p2.
destruct u3 as [[u31d  [u31c u31]]  [[u32d  [u32c
u32]] p3]].
simpl in R31.
simpl in R32.
simpl in p3.
destruct u4 as [[u41d  [u41c u41]]  [[u42d  [u42c
u42]] p4]].
simpl in R41.
simpl in R42.
simpl in p4.
simpl in R1323.
simpl in R3343.
destruct R11 as [R11d  [R11e R11]].
destruct R12 as [R12d  [R12e R12]].
destruct R21 as [R21d  [R21e R21]].
destruct R22 as [R22d  [R22e R22]].
destruct R31 as [R31d  [R31e R31]].
destruct R32 as [R32d  [R32e R32]].
destruct R41 as [R41d  [R41e R41]].
destruct R42 as [R42d  [R42e R42]].
destruct R1323 as [R1323d  [R1323e R1323]].
destruct R3343 as [R3343d  [R3343e R3343]].
exists Q1.
exists Q2.
intro x.


assert (∀x,  s x  ≈  F ∘ R11e (u11 (F ∘ (R11d ⁻¹)
x))) as R11'
```

```
x))) as R11'.
apply arrsolve1.

apply R11.
clear R11.

assert (∀x,  h x ≈ F • R12e (u12 (F • (R12d ⁻¹) x)) )
as  R12'.
apply arrsolve1.
apply R12.
clear R12.

assert (∀x,  t x  ≈ F • R21e (u21 (F • (R21d ⁻¹) x)))
as R21'.
apply arrsolve1.
apply R21.
clear R21.

assert (∀x,  h x ≈ F • R22e (u22  (F • (R22d ⁻¹) x)))
as  R22'.
apply arrsolve1.
apply R22.
clear R22.

assert (∀x,  s x ≈ F • R31e (u31 (F • (R31d ⁻¹) x)))
as R31'.
apply arrsolve1.
apply R31.
clear R31.

assert (∀x,   k x  ≈ F • R32e (u32 (F • (R32d ⁻¹)
x))) as  R32'.
apply arrsolve1.
apply R32.
clear R32.

assert (∀x,  t x  ≈ F • R41e (u41 (F • (R41d ⁻¹) x)))
as R41'.
apply arrsolve1.
```

```
apply arrsolve1.
apply R41.

clear R41.

assert (∀x,  k x ≈F • R42e (u42 (F • (R42d ⁻¹) x)) )
as  R42'.
apply arrsolve1.
apply R42.
clear R42.

assert ( F • ((p ⁻¹) ⊙ R32d ⊙ p3 ⊙ (R31e ⁻¹) ⊙ (Q2
⁻¹)) (F • Q2 (s x))  ≈
         F • ((p ⁻¹) ⊙ R32d ⊙ p3 ⊙ (R31e ⁻¹) ⊙ (Q2
⁻¹)) (t (F • Q1 x)) ) as goaleq.
apply H.
clear H.
specialize (R12' (F • (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙
Q2 ⁻¹) (F • Q2 (s x)))).
specialize (R22' (F • (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙
Q2 ⁻¹) (t (F • Q1 x)))).
assert (F • R12e
           (u12
              (F • R12d ⁻¹
                 (F • (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙
Q2 ⁻¹) (F • Q2 (s x)))))
≈
F • R12e
           (u12
              (F • R12d ⁻¹
                 (F • (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙
Q2 ⁻¹) (F • Q2 (F • R11e (u11 (F • R11d ⁻¹ x))))))))

) as Ldeveloped.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
```

```
apply setoidmapextensionality.
apply R11'.

assert (F ● R22e
           (u22
              (F ● R22d ⁻¹
                 (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙
Q2 ⁻¹) (t (F ● Q1 x)))))
  ≈ F ● R22e
           (u22
              (F ● R22d ⁻¹
                 (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙
Q2 ⁻¹) (F ● R21e (u21 (F ● R21d ⁻¹ (F ● Q1 x)))))))
) as Rdeveloped.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply R21'.
assert (F ● R12e
                 (u12
                    (F ● R12d ⁻¹
                       (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e
⁻¹ ⊙ Q2 ⁻¹)
                          (F ● Q2 (F ● R11e (u11 (F ●
R11d ⁻¹ x)))))))
≈
        F ● R22e
                 (u22
                    (F ● R22d ⁻¹
                       (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e
⁻¹ ⊙ Q2 ⁻¹)
                          (F ● R21e (u21 (F ● R21d ⁻¹
(F ● Q1 x)))))))
) as LR.
specialize (R1323 (F ● R11d ⁻¹ x)).
assert ( F ● R22e (F ● R1323e (u12 (F ● p1 (u11 (F ●
R11d ⁻¹ x))))) ≈
```

```
F ● R22e (u22 (F ● p2 (u21 (F ● R1323d (F ● R11d ⁻¹
x))))))) as R1323'.

apply setoidmapextensionality.
apply R1323.
clear R1323.

assert ( F ● R12e
     (u12
        (F ● R12d ⁻¹
           (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙ Q2 ⁻¹)
              (F ● Q2 (F ● R11e (u11 (F ● R11d ⁻¹
x)))))))) ≈
F ● R22e (F ● R1323e (u12 (F ● p1 (u11 (F ● R11d ⁻¹
x)))))
)  as LDevR1313.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirr.

assert (F ● R22e
     (u22
        (F ● R22d ⁻¹
           (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙ Q2 ⁻¹)
              (F ● R21e (u21 (F ● R21d ⁻¹ (F ● Q1
x)))))))) ≈
F ● R22e (u22 (F ● p2 (u21 (F ● R1323d (F ● R11d ⁻¹
x)))))
)  as RDevR1313.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
```

```
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.

apply setoidmapextensionality.
apply setoidfamilycmpgeneral_3.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirr.
swesetoid.
apply (R22'⁻¹ ⊙ Rdeveloped⁻¹ ⊙ LR ⊙ Ldeveloped ⊙
R12').

specialize (R32' (F ● p (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙
R31e ⁻¹ ⊙ Q2 ⁻¹) (F ● Q2 (s x))))).
specialize (R42' (F ● p (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙
R31e ⁻¹ ⊙ Q2 ⁻¹) (t (F ● Q1 x))))).
assert (F ● R32e
           (u32
              (F ● R32d ⁻¹
                 (F ● p
                    (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹
⊙ Q2 ⁻¹) (F ● Q2 (s x))))))
  ≈
F ● R32e
           (u32
              (F ● R32d ⁻¹
                 (F ● p
                    (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹
⊙ Q2 ⁻¹) (F ● Q2 (F ● R31e (u31 (F ● R31d ⁻¹
x)))))))))) as LDev.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply R31'.
assert ( F ● R42e
```

```
           (u42
              (F ● R42d ⁻¹
                 (F ● p
                    (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹
⊙ Q2 ⁻¹) (t (F ● Q1 x))))))
         ≈,{ F kc }
         F ● R42e
           (u42
              (F ● R42d ⁻¹
                 (F ● p
                    (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹
⊙ Q2 ⁻¹) ( F ● R41e (u41 (F ● R41d ⁻¹ (F ● Q1
x)))))))   ) as RDev.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply R41'.
assert (F ● R32e
           (u32
              (F ● R32d ⁻¹
                 (F ● p
                    (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹
⊙ Q2 ⁻¹)
                       (F ● Q2 (F ● R31e (u31 (F ●
R31d ⁻¹ x)))))))
≈ F ● R42e
           (u42
              (F ● R42d ⁻¹
                 (F ● p
                    (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹
⊙ Q2 ⁻¹)
                       (F ● R41e (u41 (F ● R41d ⁻¹ (F
● Q1 x)))))))

) as LR.
```

```
specialize (R3343 (F ● R31d ⁻¹ x)).

assert (F ● R42e (F ● R3343e (u32 (F ● p3 (u31 (F ●
R31d ⁻¹ x))))) ≈
        F ● R42e (u42 (F ● p4 (u41 (F ● R3343d (F ●
R31d ⁻¹ x)))))) as R3343'.
apply setoidmapextensionality.
apply R3343.
clear R3343.
assert (F ● R32e
     (u32
        (F ● R32d ⁻¹
           (F ● p
              (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙ Q2
⁻¹)
                 (F ● Q2 (F ● R31e (u31 (F ● R31d ⁻¹
x)))))))) ≈
F ● R42e (F ● R3343e (u32 (F ● p3 (u31 (F ● R31d ⁻¹
x))))))) as Left.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirr.
assert (F ● R42e (u42 (F ● p4 (u41 (F ● R3343d (F ●
R31d ⁻¹ x)))))
≈  F ● R42e
     (u42
        (F ● R42d ⁻¹
           (F ● p
              (F ● (p ⁻¹ ⊙ R32d ⊙ p3 ⊙ R31e ⁻¹ ⊙ Q2
⁻¹)
                 (F ● R41e (u41 (F ● R41d ⁻¹ (F ● Q1
```

```
x)))))))) as Right.
apply setoidmapextensionality.

apply setoidmapextensionality.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidmapextensionality.
apply setoidfamilycmpgeneral_3.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirr.
swesetoid.
apply (R42'⁻¹ ⊙ RDev⁻¹ ⊙ LR ⊙ LDev ⊙ R32').
apply (setoidfamilyirrevgeneral F _ _ _ _ _ _
goaleq).
Defined.


Definition the_categ :cat := cat_from_family T.

Definition meetingarrowsbase (C:cat):Set :=
∃f: catarr C,∃g: catarr C, (catcod C f) ≈  (catcod C
g).

Definition meetingarrowseq (C:cat)(u
v:meetingarrowsbase C):Set.
destruct u as [f  [g p]].
destruct v as [f'  [g' p']].
apply ( f ≈ f'  ∧  g ≈ g').
Defined.

Definition meetingarrows (C:cat):setoid.
apply (Build_setoid (meetingarrowsbase C)
(meetingarrowseq C)).
(* Reflexivity *)
intro u.
```

```
destruct u as [f  [g p]].
simpl.

split.
apply setoidrefl.
apply setoidrefl.
(* Symmetry *)
intros u v.
destruct u as [f  [g p]].
destruct v as [f'  [g' p']].
simpl.
intro H.
destruct H as [H1 H2].
split.
apply (setoidsym _ _ _ H1).
apply (setoidsym _ _ _ H2).
intros u v w.
destruct u as [f  [g p]].
destruct v as [f'  [g' p']].
destruct w as [f''  [g'' p'']].
simpl.
intros H K.
destruct H as [H1 H2].
destruct K as [K1 K2].
split.
apply (setoidtra _ _ _ _ H1 K1).
apply (setoidtra _ _ _ _ H2 K2).
Defined.

Definition make_meetingarrows (C:cat)(f: catarr C)(g:
catarr C)(P:(catcod C f) ≈  (catcod C g)):
meetingarrows C.
exists f.
exists g.
apply P.
Defined.

Definition pullbackobmap: (meetingarrows the_categ) -
```

```
> (catobj the_categ).
intro m.

destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
destruct p as  [p q].
simpl in p.
induction p.
simpl in q.
simpl.
exists (triple sa sb sc).
simpl.
unfold starsetoidbase.
exists a.
exists b.
exists c.
exists d.
split.
apply q.
split.
apply f.
apply g.
Defined.


Definition pullbackob: (meetingarrows the_categ)  ⇒
(catobj the_categ).
apply (Build_setoidmap  (meetingarrows the_categ)
(catobj the_categ)
                        pullbackobmap).
intros m m' P.
destruct m as  [f  [a p]].
```

```
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].

destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
destruct m' as  [f'  [g' p']].
destruct f' as  [a'  [c' f']].
destruct g' as  [b'  [d' g']].
destruct a' as  [sa' a'].
destruct b' as  [sb' b'].
destruct c' as  [sc' c'].
destruct d' as  [sd' d'].
simpl in p.
simpl in p'.
destruct p as  [p q].
destruct p' as  [p' q'].
simpl in p. induction p.
simpl in p'. induction p'.
simpl in q.
simpl in q'.
destruct P as  [P1 P2].
destruct P1 as [[r1 R1]  [[s1 S1] Q1]].
destruct P2 as [[r2 R2]  [[s2 S2] Q2]].
simpl.
simpl in r1. induction r1.
simpl in s1. induction s1.
simpl in r2. induction r2.
exists (Identity_refl _).
simpl in R1.
simpl in S1.
simpl in R2.
assert (Identity (Identity_refl sc) s2) as H.
apply hedberg.
apply stages_DecId.
induction H.
simpl in S2.
```

```
exists R1.
exists R2.

exists S1.
exists S2.
split.
intro t.
clear Q2.
simpl in Q1.
simpl.
apply Q1.
simpl in Q2.
simpl.
apply Q2.
Defined.

Definition mapof (f: catarr the_categ):
T (catdom the_categ f) ⇒  T (catcod the_categ f).
destruct f as  [a  [b f]].
exact f.
Defined.

Definition elt_builder_pullbackob (m : meetingarrows
the_categ)
(x: T ((catdom the_categ) (projT1 m)))
(y: T ((catdom the_categ) (projT1 (projT2 m))))
(p: T • (projT2 (projT2 m))
  (mapof (projT1 m) x) ≈,{ T ((catcod the_categ)
(projT1 (projT2 m))) }
(mapof (projT1 (projT2 m))) y) :  T (pullbackob m).
destruct m as  [f  [g q]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
simpl in q.
simpl in x.
simpl in y.
simpl in p.
destruct q as  [sq q]
```

```
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
destruct q as [qpf q].
simpl in qpf.
induction qpf.
simpl in q.
simpl.
exists x.
exists y.
simpl in p.
apply p.
Defined.

Definition pullback1maphelperunder (m : meetingarrows
the_categ):
   T (pullbackob m) -> T ((catdom the_categ) (projT1
m)).
intro u.
destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
destruct p as  [p q].
simpl in p. induction p.
simpl in q.
simpl.
simpl in u.
apply (projT1 u).
Defined.


Definition pullback2maphelperunder (m : meetingarrows
```

```
the_categ):
   T (pullbackob m) -> T ((catdom the_categ) (projT1
(projT2 m))).
intro u.
destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
destruct p as  [p q].
simpl in p. induction p.
simpl in q.
simpl.
simpl in u.
apply (projT1 (projT2 u)).
Defined.


Definition pullback1maphelper (m : meetingarrows
the_categ):
   T (pullbackob m) ⇒ T ((catdom the_categ) (projT1
m)).
apply (Build_setoidmap  (T (pullbackob m))
          (T ((catdom the_categ) (projT1 m)))
(pullback1maphelperunder m)).
intros u v P.
destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
```

```
simpl in p.
destruct p as  [p q].
simpl in p. induction p.
simpl in q.
simpl in u.
simpl in v.
simpl.
destruct u as  [x  [y R]].
destruct v as  [x' [y' R']].
simpl.
simpl in P.
apply P.
Defined.


Definition pullback2maphelper (m : meetingarrows
the_categ):
   T (pullbackob m) ⇒ T ((catdom the_categ) (projT1
(projT2 m))).
apply (Build_setoidmap  (T (pullbackob m))
          (T ((catdom the_categ) (projT1 (projT2
m)))) (pullback2maphelperunder m)).
intros u v P.
destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
destruct p as  [p q].
simpl in p. induction p.
simpl in q.
simpl in u.
simpl in v.
simpl.
destruct u as  [x  [y R]].
```

```
destruct u as  [x  [y R]]].
destruct v as  [x' [y' R']].
simpl.

simpl in P.
apply P.
Defined.


Definition pullback1map: (meetingarrows the_categ) ->
(catarr the_categ).
intro m.
exists (pullbackob m).
exists (catdom the_categ (projT1 m)).
apply (pullback1maphelper m).
Defined.


Definition pullback2map: (meetingarrows the_categ) ->
(catarr the_categ).
intro m.
exists (pullbackob m).
exists (catdom the_categ (projT1 (projT2 m))).
apply (pullback2maphelper m).
Defined.



Definition pullback1: (meetingarrows the_categ) ⇒
(catarr the_categ).
apply (Build_setoidmap (meetingarrows
the_categ)  (catarr the_categ)
       pullback1map).
intros m m' P.

destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a]
```

```
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
destruct p as  [p q].
simpl in p. induction p.
simpl in q.

destruct m' as  [f'  [g' p']].
destruct f' as  [a'  [c' f']].
destruct g' as  [b'  [d' g']].
destruct a' as  [sa' a'].
destruct b' as  [sb' b'].
destruct c' as  [sc' c'].
destruct d' as  [sd' d'].
simpl in p'.
destruct p' as  [p' q'].
simpl in p'. induction p'.
simpl in q'.

simpl in P.
destruct P as  [P1 P2].
destruct P1 as [[r1 R1]  [[s1 S1] Q1]].
destruct P2 as [[r2 R2]  [[s2 S2] Q2]].
simpl in r1. induction r1.
simpl in s1. induction s1.
simpl in r2. induction r2.
assert (Identity (Identity_refl sc) s2) as H.
apply hedberg.
apply stages_DecId.
induction H.

simpl.


assert (
ap : Identity (triple sa sb sc) (triple sa sb sc)
```

```
∃p : identity (triple sa sb sc) (triple sa sb sc),
     starsetoideq (projT1 (ifm sa)) (projT2 (ifm sa))
       (projT1 (ifm sb)) (projT2 (ifm sb)) (projT1
(ifm sc))
       (projT2 (ifm sc))
       (transport stages (λ k : stages, projT1 (ifm
k))
          (triple sa sb sc) (triple sa sb sc) p
          (existT
             (λ a0 : projT1 (ifm sa),
              ∃b0 : projT1 (ifm sb),
              ∃c0 : projT1 (ifm sc),
              ∃d0 : projT1 (ifm sc),
              c0 ≈,{ projT1 (ifm sc) }d0
              ∧ setoidmap ((projT2 (ifm sa)) a0)
((projT2 (ifm sc)) c0)
                ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0)) a
             (existT
                (λ b0 : projT1 (ifm sb),
                 ∃c0 : projT1 (ifm sc),
                 ∃d0 : projT1 (ifm sc),
                 c0 ≈,{ projT1 (ifm sc) }d0
                 ∧ setoidmap ((projT2 (ifm sa)) a)
((projT2 (ifm sc)) c0)
                   ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0))
                b
                (existT
                   (λ c0 : projT1 (ifm sc),
                    ∃d0 : projT1 (ifm sc),
                    c0 ≈,{ projT1 (ifm sc) }d0
                    ∧ setoidmap ((projT2 (ifm sa)) a)
((projT2 (ifm sc)) c0)
                      ∧ setoidmap ((projT2 (ifm sb))
b)
                           ((projT2 (ifm sc)) d0)) c
                (existT
```

```
                    (existT
                       (λ d0 : projT1 (ifm sc),
                        c ≈,{ projT1 (ifm sc) }d0

                        ∧ setoidmap ((projT2 (ifm sa))
a)
                            ((projT2 (ifm sc)) c)
                          ∧ setoidmap ((projT2 (ifm
sb)) b)
                              ((projT2 (ifm sc)) d0))
d
                       (q, (f, g))))))
        (existT
           (λ a0 : projT1 (ifm sa),
            ∃b0 : projT1 (ifm sb),
            ∃c0 : projT1 (ifm sc),
            ∃d0 : projT1 (ifm sc),
            c0 ≈,{ projT1 (ifm sc) }d0
            ∧ setoidmap ((projT2 (ifm sa)) a0)
((projT2 (ifm sc)) c0)
              ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0)) a'
           (existT
              (λ b0 : projT1 (ifm sb),
               ∃c0 : projT1 (ifm sc),
               ∃d0 : projT1 (ifm sc),
               c0 ≈,{ projT1 (ifm sc) }d0
               ∧ setoidmap ((projT2 (ifm sa)) a')
((projT2 (ifm sc)) c0)
                 ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0)) b'
              (existT
                 (λ c0 : projT1 (ifm sc),
                  ∃d0 : projT1 (ifm sc),
                  c0 ≈,{ projT1 (ifm sc) }d0
                  ∧ setoidmap ((projT2 (ifm sa)) a')
((projT2 (ifm sc)) c0)
                    ∧ setoidmap ((projT2 (ifm sb)) b')
((projT2 (ifm sc)) d0))
```

```
((projT2 (ifm sc)) d0))
                c'
                (existT

                   (λ d0 : projT1 (ifm sc),
                    c' ≈,{ projT1 (ifm sc) }d0
                    ∧ setoidmap ((projT2 (ifm sa))
a') ((projT2 (ifm sc)) c')
                      ∧ setoidmap ((projT2 (ifm sb))
b')
                           ((projT2 (ifm sc)) d0)) d'
                   (q', (f', g'))))))

) as e.

exists (Identity_refl (triple sa sb sc)).
simpl.
exists R1.
exists R2.
exists S1.
exists S2.
split.
apply Q1.
apply Q2.
exists e.
simpl in R1.
exists (existT (λ p, transport stages (λ k : stages,
projT1 (ifm k)) sa sa p a
≈,{ projT1 (ifm sa) }a')
(Identity_refl sa) R1).

intro L.
destruct L as  [x  [y L]].
simpl.
destruct e as  [e  H].
assert (Identity (Identity_refl (triple sa sb sc)) e)
as G.
apply hedberg.
apply stages_DecId
```

```
apply stages_decid.
induction G.
simpl in H.

destruct H as  [p1  [p2  [p3  [p4 [P1 P2]]]]].
simpl.
apply setoidfamilyirr.
Defined.


Definition pullback2: (meetingarrows the_categ) ⇒
(catarr the_categ).
apply (Build_setoidmap (meetingarrows
the_categ)  (catarr the_categ)
       pullback2map).
intros m m' P.

destruct m as  [f  [g p]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as  [sa a].
destruct b as  [sb b].
destruct c as  [sc c].
destruct d as  [sd d].
simpl in p.
destruct p as  [p q].
simpl in p. induction p.
simpl in q.

destruct m' as  [f'  [g' p']].
destruct f' as  [a'  [c' f']].
destruct g' as  [b'  [d' g']].
destruct a' as  [sa' a'].
destruct b' as  [sb' b'].
destruct c' as  [sc' c'].
destruct d' as  [sd' d'].
simpl in p'.
destruct p' as  [p' q'].
```

```
simpl in p'. induction p'.
simpl in q'.


simpl in P.
destruct P as  [P1 P2].
destruct P1 as [[r1 R1]  [[s1 S1] Q1]].
destruct P2 as [[r2 R2]  [[s2 S2] Q2]].
simpl in r1. induction r1.
simpl in s1. induction s1.
simpl in r2. induction r2.
assert (Identity (Identity_refl sc) s2) as H.
apply hedberg.
apply stages_DecId.
induction H.

simpl.

assert (
∃p : Identity (triple sa sb sc) (triple sa sb sc),
     starsetoideq (projT1 (ifm sa)) (projT2 (ifm sa))
       (projT1 (ifm sb)) (projT2 (ifm sb)) (projT1
(ifm sc))
       (projT2 (ifm sc))
       (transport stages (λ k : stages, projT1 (ifm
k))
          (triple sa sb sc) (triple sa sb sc) p
          (existT
             (λ a0 : projT1 (ifm sa),
              ∃b0 : projT1 (ifm sb),
              ∃c0 : projT1 (ifm sc),
              ∃d0 : projT1 (ifm sc),
              c0 ≈,{ projT1 (ifm sc) }d0
              ∧ setoidmap ((projT2 (ifm sa)) a0)
((projT2 (ifm sc)) c0)
                ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0)) a
            (existT
```

```
                  (λ b0 : projT1 (ifm sb),
                   ∃c0 : projT1 (ifm sc),
                   ∃d0 : projT1 (ifm sc),

                   c0 ≈,{ projT1 (ifm sc) }d0
                   ∧ setoidmap ((projT2 (ifm sa)) a)
((projT2 (ifm sc)) c0)
                     ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0))
                  b
                  (existT
                     (λ c0 : projT1 (ifm sc),
                      ∃d0 : projT1 (ifm sc),
                      c0 ≈,{ projT1 (ifm sc) }d0
                      ∧ setoidmap ((projT2 (ifm sa)) a)
((projT2 (ifm sc)) c0)
                        ∧ setoidmap ((projT2 (ifm sb))
b)
                            ((projT2 (ifm sc)) d0)) c
                     (existT
                        (λ d0 : projT1 (ifm sc),
                         c ≈,{ projT1 (ifm sc) }d0
                         ∧ setoidmap ((projT2 (ifm sa))
a)
                             ((projT2 (ifm sc)) c)
                           ∧ setoidmap ((projT2 (ifm
sb)) b)
                               ((projT2 (ifm sc)) d0))
d
                        (q, (f, g))))))
       (existT
          (λ a0 : projT1 (ifm sa),
           ∃b0 : projT1 (ifm sb),
           ∃c0 : projT1 (ifm sc),
           ∃d0 : projT1 (ifm sc),
           c0 ≈,{ projT1 (ifm sc) }d0
           ∧ setoidmap ((projT2 (ifm sa)) a0)
((projT2 (ifm sc)) c0)
```

```
            ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0)) a'
          (existT

             (λ b0 : projT1 (ifm sb),
              ∃c0 : projT1 (ifm sc),
              ∃d0 : projT1 (ifm sc),
              c0 ≈,{ projT1 (ifm sc) }d0
              ∧ setoidmap ((projT2 (ifm sa)) a')
((projT2 (ifm sc)) c0)
                ∧ setoidmap ((projT2 (ifm sb)) b0)
((projT2 (ifm sc)) d0)) b'
             (existT
                (λ c0 : projT1 (ifm sc),
                 ∃d0 : projT1 (ifm sc),
                 c0 ≈,{ projT1 (ifm sc) }d0
                 ∧ setoidmap ((projT2 (ifm sa)) a')
((projT2 (ifm sc)) c0)
                   ∧ setoidmap ((projT2 (ifm sb)) b')
((projT2 (ifm sc)) d0))
                c'
                (existT
                   (λ d0 : projT1 (ifm sc),
                    c' ≈,{ projT1 (ifm sc) }d0
                    ∧ setoidmap ((projT2 (ifm sa))
a') ((projT2 (ifm sc)) c')
                      ∧ setoidmap ((projT2 (ifm sb))
b')
                           ((projT2 (ifm sc)) d0)) d'
                   (q', (f', g'))))))
) as e.


exists (Identity_refl (triple sa sb sc)).
simpl.
exists R1.
exists R2.
exists S1.
```

```
exists S2.
split.
apply Q1.

apply Q2.
exists e.
exists (existT (λ p,
transport stages (λ k : stages, projT1 (ifm k)) sb sb
p b
     ≈,{ projT1 (ifm sb) }b') (Identity_refl sb) R2).

intro L.
destruct L as  [x  [y L]].
simpl.
destruct e as  [e  H].
assert (Identity (Identity_refl (triple sa sb sc)) e)
as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in H.
destruct H as  [p1  [p2  [p3  [p4 [P1 P2]]]]].
simpl.
apply setoidfamilyirr.
Defined.

Definition makediaghelper  (m: meetingarrows
the_categ):
T (pullbackob m) -> T ((catcod the_categ) (projT1
m)).
intro w.
destruct  m as [f  [g P]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as [a  sa].
destruct b as [b  sb].
destruct c as [c  sc].
destruct d as [d  sd].
```

```
simpl in P.
destruct P as [p P].
simpl.

simpl in p. induction p.
simpl in P.
simpl in w.
destruct w as [x  [y Q]].
apply f.
apply x.
Defined.

Definition makecms1  (m: meetingarrows the_categ):
catcms the_categ.
exists (pullback1 m).
exists (projT1 m).
destruct  m as [f  [g P]].
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as [a  sa].
destruct b as [b  sb].
destruct c as [c  sc].
destruct d as [d  sd].
simpl in P.
destruct P as [p P].
simpl in p. induction p.
simpl in P.
simpl.
exists (Identity_refl a).
apply setoidrefl.
Defined.


Definition makecms2  (m: meetingarrows the_categ):
catcms the_categ.
exists (pullback2 m).
exists (projT1 (projT2 m)).
destruct  m as [f  [g P]].
```

```
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
destruct a as [a  sa].

destruct b as [b  sb].
destruct c as [c  sc].
destruct d as [d  sd].
simpl in P.
destruct P as [p P].
simpl in p. induction p.
simpl in P.
simpl.
exists (Identity_refl b).
apply setoidrefl.
Defined.


Lemma Is_square_lemma (m: meetingarrows the_categ):
Square (projT1 m) (projT1 (projT2 m)) (pullback1 m)
(pullback2 m).
unfold Square.
exists (catcmp the_categ (makecms1 m)).
unfold Comp.
split.
exists (makecms1 m).
split.
apply setoidrefl.
split.
apply setoidrefl.
apply setoidrefl.
exists (makecms2 m).
split.
apply setoidrefl.
split.
apply setoidrefl.

destruct  m as [f  [g P]].
destruct f as  [a  [c f]].
```

```
destruct g as  [b  [d g]].
simpl in P.


assert (d ≈,{ U }c) as P'.
apply setoidsym.
apply P.
destruct a as [a  sa].
destruct b as [b  sb].
destruct c as [c  sc].
destruct d as [d  sd].
simpl in f.
simpl in g.
destruct P as [p P].
simpl in p. induction p.
simpl.

assert (
∃p : Identity (triple a b c) (triple a b c),
     starsetoideq (projT1 (ifm a)) (projT2 (ifm a))
       (projT1 (ifm b)) (projT2 (ifm b)) (projT1 (ifm
c))
       (projT2 (ifm c))
       (transport stages (λ k : stages, projT1 (ifm
k))
          (triple a b c) (triple a b c) p
          (existT
             (λ a0 : projT1 (ifm a),
              ∃b0 : projT1 (ifm b),
              ∃c0 : projT1 (ifm c),
              ∃d : projT1 (ifm c),
              c0 ≈,{ projT1 (ifm c) }d
              ∧ setoidmap ((projT2 (ifm a)) a0)
((projT2 (ifm c)) c0)
                ∧ setoidmap ((projT2 (ifm b)) b0)
((projT2 (ifm c)) d)) sa
             (existT
                (λ b0 : projT1 (ifm b),
```

```
                  ∃c0 : projT1 (ifm c),
                  ∃d : projT1 (ifm c),
                  c0 ≈,{ projT1 (ifm c) }d

                  ∧ setoidmap ((projT2 (ifm a)) sa)
((projT2 (ifm c)) c0)
                    ∧ setoidmap ((projT2 (ifm b)) b0)
((projT2 (ifm c)) d)) sb
                 (existT
                    (λ c0 : projT1 (ifm c),
                     ∃d : projT1 (ifm c),
                     c0 ≈,{ projT1 (ifm c) }d
                     ∧ setoidmap ((projT2 (ifm a)) sa)
((projT2 (ifm c)) c0)
                       ∧ setoidmap ((projT2 (ifm b))
sb) ((projT2 (ifm c)) d))
                    sc
                    (existT
                       (λ d : projT1 (ifm c),
                        sc ≈,{ projT1 (ifm c) }d
                        ∧ setoidmap ((projT2 (ifm a))
sa)
                            ((projT2 (ifm c)) sc)
                          ∧ setoidmap ((projT2 (ifm
b)) sb)
                              ((projT2 (ifm c)) d)) sd
                       (P, (f, g))))))
       (existT
          (λ a0 : projT1 (ifm a),
           ∃b0 : projT1 (ifm b),
           ∃c0 : projT1 (ifm c),
           ∃d : projT1 (ifm c),
           c0 ≈,{ projT1 (ifm c) }d
           ∧ setoidmap ((projT2 (ifm a)) a0) ((projT2
(ifm c)) c0)
             ∧ setoidmap ((projT2 (ifm b)) b0)
((projT2 (ifm c)) d)) sa
          (existT
```

```
              (λ b0 : projT1 (ifm b),
               ∃c0 : projT1 (ifm c),
               ∃d : projT1 (ifm c),

               c0 ≈,{ projT1 (ifm c) }d
               ∧ setoidmap ((projT2 (ifm a)) sa)
((projT2 (ifm c)) c0)
                 ∧ setoidmap ((projT2 (ifm b)) b0)
((projT2 (ifm c)) d)) sb
              (existT
                 (λ c0 : projT1 (ifm c),
                  ∃d : projT1 (ifm c),
                  c0 ≈,{ projT1 (ifm c) }d
                  ∧ setoidmap ((projT2 (ifm a)) sa)
((projT2 (ifm c)) c0)
                    ∧ setoidmap ((projT2 (ifm b)) sb)
((projT2 (ifm c)) d)) sc
                 (existT
                    (λ d : projT1 (ifm c),
                     sc ≈,{ projT1 (ifm c) }d
                     ∧ setoidmap ((projT2 (ifm a)) sa)
((projT2 (ifm c)) sc)
                       ∧ setoidmap ((projT2 (ifm b))
sb) ((projT2 (ifm c)) d))
                    sd (P, (f, g))))))
) as proppen.


exists (Identity_refl (triple a b c)).
simpl.
exists (setoidrefl _ sa).
exists (setoidrefl _ sb).
exists (setoidrefl _ sc).
exists (setoidrefl _ sd).
split.
intro x.
assert ((projT2 (ifm c))
   • (setoidrefl (projT1 (ifm c)) sc)
```

```
     (f x) ≈ f x) as im1.
apply setoidfamilyref.
assert (f x  ≈ f

     ((projT2 (ifm a))
      • (setoidrefl (projT1 (ifm a)) sa) x)) as im2.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilyref.
swesetoid.
intro x.
assert ((projT2 (ifm c))
   • (setoidrefl (projT1 (ifm c)) sd)
     (g x) ≈ g x) as im1.
apply setoidfamilyref.
assert (g x  ≈ g
     ((projT2 (ifm b))
      • (setoidrefl (projT1 (ifm b)) sb) x)) as im2.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilyref.
swesetoid.
exists proppen.
exists P'.
destruct P' as [p'  P'].
assert (Identity (Identity_refl c) p') as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl.
intro u.
destruct u as  [x [y  K]].
simpl.
destruct proppen as  [r proppen].
assert (Identity (Identity_refl (triple a b c)) r) as
G.
apply hedberg.
apply stages_DecId.
```

```
induction G.
simpl.
simpl in proppen.

destruct proppen as  [p1 [p2 [p3 [p4  [pr1  pr2]]]]].
simpl.
simpl in P.
simpl in P'.
assert ((projT2 (ifm c)) ● P'
     (g
        ((projT2 (ifm b))
         ● (setoidrefl (projT1 (ifm b)) sb) y))
   ≈ (projT2 (ifm c)) ● P'
     (g y)) as im1.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidfamilyref.
assert ( f
     ((projT2 (ifm a))
      ● (setoidrefl (projT1 (ifm a)) sa)
        ((projT2 (ifm a)) ● p1 x))
≈ f ((projT2 (ifm a)) ● p1 x)) as im2.
apply setoidmapextensionality.
apply setoidfamilyref.
assert (f ((projT2 (ifm a)) ● p1 x) ≈(projT2 (ifm c))
● P' (g y)) as im3.
specialize (pr1 x).
specialize (pr2 y).
assert  ((projT2 (ifm c)) ● P'
((projT2 (ifm c)) ● P (f x))
      ≈  (projT2 (ifm c)) ● P' (g y)) as K'.
apply setoidmapextensionality.
apply K.
clear K.
assert (f ((projT2 (ifm a)) ● p1 x)
   ≈ (projT2 (ifm c)) ● P'
        ((projT2 (ifm c)) ● P (f x))) as K2.
assert ((projT2 (ifm c)) ● p3 (f x)
```

```
          ≈  (projT2 (ifm c)) • P'
      ((projT2 (ifm c)) • P (f x))) as K3.
apply  setoidfamilycmpgeneral_3.

apply setoidfamilyirr.
assert ( f ((projT2 (ifm a)) • p1 x) ≈
(projT2 (ifm c)) • p3 (f x)) as K4.
apply setoidsym.
apply pr1.
swesetoid.
swesetoid.
swesetoid.
Defined.

Lemma equal_in_pullback
  (m:meetingarrows the_categ)
  (u v : T (pullbackob m))(p:pullback1maphelper m u ≈
pullback1maphelper m v)
  (q:pullback2maphelper m u ≈ pullback2maphelper m
v): u≈v.
Proof.
destruct m as [f  [g P]].
destruct f as [df  [cf f]].
destruct g as [dg  [cg g]].
destruct df as [sdf df].
destruct cf as [scf cf].
destruct dg as [sdg dg].
destruct cg as [scg cg].

simpl in P.
simpl in f.
simpl in g.
destruct P as  [pfP P].
simpl in pfP.
induction pfP.
simpl in P.
simpl in u.
simpl in v.
```

```
destruct u as  [pu  [qu Pu]].
destruct v as  [pv  [qv Pv]].
simpl.

simpl in p.
simpl in q.
split.
apply p.
apply q.
Defined.

Definition pb_pairing_helper
  (f : catarr the_categ)
  (g : catarr the_categ)
  (P' : (catcod the_categ) f ≈,{ the_categ }(catcod
the_categ) g)
  (h : catarr the_categ)
  (k : catarr the_categ)
  (P : Square' f g h k):  (* changed Square to
Square' *)
  T (catdom the_categ h) -> T (pullbackob
(make_meetingarrows the_categ f g P')).
intro x.
destruct f as  [a  [c f]].
destruct g as  [b  [d g]].
simpl in P'.
destruct P as [w [z P]].
destruct P as  [P1 [P2 [P3 [P4 P5]]]].
destruct h as [hd  [hc hm]].
destruct k as [kd  [kc km]].
destruct w as [[w1d  [w1c w1]]  [[w2d  [w2c w2]]
pw]].
destruct z as [[z1d  [z1c z1]]  [[z2d  [z2c z2]]
pz]].
simpl in x.
destruct P1 as [p1 [q1 P1]].
destruct P2 as [p2 [q2 P2]].
destruct P3 as [p3 [q3 P3]].
```

```
destruct P4 as [p4 [q4 P4]].
destruct P5 as [p5 [q5 P5]].
assert (z1c  ≈,{ U } z2d) as pz'.

apply pz.

apply (elt_builder_pullbackob (make_meetingarrows
the_categ
           (existT (λ a0 : U, ∃b0 : U, T a0 ⇒ T b0) a
              (existT (λ b0 : U, T a ⇒ T b0) c f))
           (existT (λ a0 : U, ∃b0 : U, T a0 ⇒ T b0) b
              (existT (λ b0 : U, T b ⇒ T b0) d g))
P')
(T ● (p2 ⊙ pw ⊙ (q1⁻¹)) (hm x)) (T ● (p4 ⊙ pz' ⊙
q3⁻¹) (km  (T ● (p3 ⊙ p5 ⊙ (p1⁻¹)) x)))).

destruct a as [sa a].
destruct c as [sc c].
destruct b as [sb b].
destruct d as [sd d].
simpl in f.
simpl in g.
destruct P' as  [p' P'].
simpl in p'. induction p'.
simpl in P'.

destruct w1d as [sw1d w1d].
destruct w1c as [sw1c w1c].
destruct w2d as [sw2d w2d].
destruct w2c as [sw2c w2c].
simpl in w1.
simpl in w2.
simpl in pw.
destruct pw as [pfpw pw].
simpl in pfpw.
induction pfpw.
simpl in pw.
```

```
destruct z1d as  [pfz1d z1d].
destruct z1c as  [pfz1c z1c].

destruct z2d as  [pfz2d z2d].
destruct z2c as  [pfz2c z2c].
simpl in z1.
simpl in z2.

simpl in pz.
destruct pz as [pfpz pz].
simpl in pfpz.
induction pfpz.
simpl in pz.


destruct hd as [shd hd].
destruct hc as [shc hc].
simpl in hm.
destruct kd as [skd kd].
destruct kc as [skc kc].
simpl in km.

destruct p1 as [sp1 p1].
simpl in sp1.
induction sp1.
simpl in p1.
destruct q1 as [sq1 q1].
simpl in sq1.
induction sq1.
simpl in q1.
simpl in P1.
destruct p2 as [sp2 p2].
simpl in sp2.
induction sp2.
simpl in p2.
destruct q2 as [sq2 q2].
simpl in sq2.
```

```
simpl in sq2.
induction sq2.
simpl in q2.
simpl in P2.


destruct p3 as [sp3 p3].
simpl in sp3.
induction sp3.
simpl in p3.
destruct q3 as [sq3 q3].
simpl in sq3.
induction sq3.
simpl in q3.
simpl in P3.

destruct p4 as [sp4 p4].
simpl in sp4.
induction sp4.
simpl in p4.
destruct q4 as [sq4 q4].
simpl in sq4.
induction sq4.
simpl in q4.
simpl in P4.

destruct p5 as [sp5 p5].
simpl in sp5.
induction sp5.
simpl in p5.
destruct q5 as [sq5 q5].
simpl in sq5.
assert (Identity (Identity_refl pfz2c) sq5) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in q5.
simpl in P5.
simpl in pz'.
```

```
simpl in pz'.
destruct pz' as [pfpz' pz'].
simpl in pfpz'.
assert (Identity (Identity_refl pfz1c) pfpz') as G.

apply hedberg.
apply stages_DecId.
induction G.
simpl in pz'.
simpl.

assert ( ∀x,
        hm x ≈ (projT2 (ifm sw1c)) • q1 (w1 ((projT2
(ifm sw1d)) • (p1 ⁻¹) x))
       ) as P1'.
apply arrsolve1.
apply P1. clear P1.
assert (∀x,
       f x  ≈ (projT2 (ifm pfz2c)) • q2 (w2 ((projT2
(ifm sw1c)) • (p2 ⁻¹) x)))
        as P2'.
apply arrsolve1.
apply P2. clear P2.
assert (∀x,
       km x ≈ (projT2 (ifm pfz1c)) • q3 (z1  ((projT2
(ifm sw1d)) • (p3 ⁻¹) x))) as P3'.
apply arrsolve1.
apply P3. clear P3.
assert (∀x,
        g x ≈ (projT2 (ifm pfz2c)) • q4 (z2 ((projT2
(ifm pfz1c)) • (p4 ⁻¹) x))
       ) as P4'.
apply arrsolve1.
apply P4. clear P4.

assert (
(projT2 (ifm pfz2c)) • P'
     (f ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙ q1 ⁻¹) (hm
x)))
```

```
x)))
≈
(projT2 (ifm pfz2c)) • P'
     ((projT2 (ifm pfz2c)) • q2 (w2 ((projT2 (ifm
sw1c)) • (p2 ⁻¹)  ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙
q1 ⁻¹) ((projT2 (ifm sw1c)) • q1 (w1 ((projT2 (ifm
sw1d)) • p1 ⁻¹ x)))))))
) as LHS.
apply setoidmapextensionality.
specialize (P2' ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙ q1
⁻¹) (hm x))).
assert ((projT2 (ifm pfz2c)) • q2
          (w2
             ((projT2 (ifm sw1c)) • p2 ⁻¹
                ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙ q1
⁻¹) (hm x))))
≈
((projT2 (ifm pfz2c)) • q2 (w2 ((projT2 (ifm sw1c)) •
(p2 ⁻¹)  ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙ q1 ⁻¹)
((projT2 (ifm sw1c)) • q1 (w1 ((projT2 (ifm sw1d)) •
p1 ⁻¹ x)))))))) as LHSlemma.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply P1'.
swesetoid.
clear P1'.
clear P2'.
assert ( g
     ((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙ q3 ⁻¹)
        (km ((projT2 (ifm sw1d)) • (p3 ⊙ p5 ⊙ p1 ⁻¹)
x)))
≈  (projT2 (ifm pfz2c)) • q4 (z2 ((projT2 (ifm
pfz1c)) • p4 ⁻¹
((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙ q3 ⁻¹)
        (  (projT2 (ifm pfz1c)) • q3 (z1 ((projT2
(ifm sw1d)) • p3 ⁻¹  ((projT2 (ifm sw1d)) • (p3 ⊙ p5
```

```
(ifm sw1d)) • p5     ((projT2 (ifm sw1d)) • (p5 ⊙ p5
⊙ p1 ⁻¹) x)))   ))))
) as RHS.
specialize (P4' ((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙
q3 ⁻¹)
        (km ((projT2 (ifm sw1d)) • (p3 ⊙ p5 ⊙ p1 ⁻¹)
x)))).
assert ((projT2 (ifm pfz2c)) • q4
          (z2
             ((projT2 (ifm pfz1c)) • p4 ⁻¹
                ((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙
q3 ⁻¹)
                   (km ((projT2 (ifm sw1d)) • (p3 ⊙
p5 ⊙ p1 ⁻¹) x)))))
≈ (projT2 (ifm pfz2c)) • q4
     (z2
        ((projT2 (ifm pfz1c)) • p4 ⁻¹
           ((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙ q3 ⁻¹)
              ((projT2 (ifm pfz1c)) • q3
                 (z1
                    ((projT2 (ifm sw1d)) • p3 ⁻¹
                       ((projT2 (ifm sw1d)) • (p3 ⊙
p5 ⊙ p1 ⁻¹) x)))))))
) as RHSlemma.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply P3'.
swesetoid.
clear P3'.
clear P4'.

assert ((projT2 (ifm pfz2c)) • P'
          ((projT2 (ifm pfz2c)) • q2
             (w2
                ((projT2 (ifm sw1c)) • p2 ⁻¹
                   ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙
```

```
                    ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙
q1 ⁻¹)
                       ((projT2 (ifm sw1c)) • q1
                          (w1 ((projT2 (ifm sw1d)) •
p1 ⁻¹ x)))))))

≈ (projT2 (ifm pfz2c)) • q4
          (z2
             ((projT2 (ifm pfz1c)) • p4 ⁻¹
                ((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙
q3 ⁻¹)
                   ((projT2 (ifm pfz1c)) • q3
                      (z1
                         ((projT2 (ifm sw1d)) • p3 ⁻¹
                            ((projT2 (ifm sw1d)) •
(p3 ⊙ p5 ⊙ p1 ⁻¹) x)))))))
) as LR.

specialize (P5 ((projT2 (ifm sw1d)) • (p1 ⁻¹) x)).
assert (
(projT2 (ifm pfz2c)) • q4
((projT2 (ifm pfz2c)) • q5
         (w2 ((projT2 (ifm sw1c)) • pw (w1 ((projT2
(ifm sw1d)) • p1 ⁻¹ x)))))
       ≈
(projT2 (ifm pfz2c)) • q4
     (z2
         ((projT2 (ifm pfz1c)) • pz
            (z1 ((projT2 (ifm sw1d)) • p5 ((projT2
(ifm sw1d)) • p1 ⁻¹ x))))))
as P5'.
apply setoidmapextensionality.
apply P5.
clear P5.
assert ((projT2 (ifm pfz2c)) • q4
          ((projT2 (ifm pfz2c)) • q5
             (w2
                ((projT2 (ifm sw1c)) • pw
```

```
                ((projT2 (ifm sw1c)) • pw
                   (w1 ((projT2 (ifm sw1d)) • p1 ⁻¹
x)))))
        ≈  (projT2 (ifm pfz2c)) • P'

     ((projT2 (ifm pfz2c)) • q2
        (w2
           ((projT2 (ifm sw1c)) • p2 ⁻¹
              ((projT2 (ifm sw1c)) • (p2 ⊙ pw ⊙ q1
⁻¹)
                 ((projT2 (ifm sw1c)) • q1
                    (w1 ((projT2 (ifm sw1d)) • p1 ⁻¹
x)))))))
) as wpart.
apply setoidfamilycmpgeneral_3.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidrefl.
assert ((projT2 (ifm pfz2c)) • q4
          (z2
             ((projT2 (ifm pfz1c)) • pz
                (z1
                   ((projT2 (ifm sw1d)) • p5 ((projT2
(ifm sw1d)) • p1 ⁻¹ x)))))

≈
(projT2 (ifm pfz2c)) • q4
     (z2
        ((projT2 (ifm pfz1c)) • p4 ⁻¹
           ((projT2 (ifm pfz1c)) • (p4 ⊙ pz' ⊙ q3 ⁻¹)
              ((projT2 (ifm pfz1c)) • q3
                 (z1
                    ((projT2 (ifm sw1d)) • p3 ⁻¹
```

```
                    ((projT2 (ifm sw1d)) • p5 ⁻¹
                       ((projT2 (ifm sw1d)) • (p3 ⊙
p5 ⊙ p1 ⁻¹) x))))))) as zpart.
apply setoidmapextensionality.

apply setoidmapextensionality.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidmapextensionality.
apply setoidfamilycmpgeneral_3.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilyirrgeneral.
apply setoidrefl.
swesetoid.
swesetoid.
Defined.



Definition pb_pairing
  (f : catarr the_categ)
  (g : catarr the_categ)
  (P' : (catcod the_categ) f ≈,{ the_categ }(catcod
the_categ) g)
  (h : catarr the_categ)
  (k : catarr the_categ)
  (P : Square' f g h k):
  T (catdom the_categ h) ⇒ T (pullbackob
(make_meetingarrows the_categ f g P')).
apply (Build_setoidmap (T (catdom the_categ h)) (T
(pullbackob (make_meetingarrows the_categ f g P')))
(pb_pairing_helper f g P' h k P)).
intros x y H.


destruct f as  [a  [c f]].
```

```
destruct g as  ⌊b  ⌊d g⌋⌋.
simpl in P'.
destruct a as [sa a].
destruct c as [sc c].
destruct b as [sb b].
destruct d as [sd d].
destruct P' as  [p' P'].
simpl in p'. induction p'.
destruct h as  [n  [a' h]].
destruct k as  [n'  [b' k]].
simpl in P'.
simpl in f.
simpl in g.

destruct n as [sn n].
destruct n' as [sn' n'].
destruct P as [u [v [P1 [P2 [P3 [P4 P5]]]]]].
simpl in h.
simpl in k.
destruct u as [u1  [u2 pu]].
destruct v as [v1  [v2 pv]].


destruct u1 as [u1d  [u1c u1]].
destruct u2 as [u2d  [u2c u2]].
simpl in pu.
destruct v1 as [v1d  [v1c v1]].
destruct v2 as [v2d  [v2c v2]].
simpl in pv.

destruct u1d as [su1d u1d].
destruct u1c as [su1c u1c].
destruct u2d as [su2d u2d].
destruct u2c as [su2c u2c].
destruct v1d as [sv1d v1d].
destruct v1c as [sv1c v1c].
destruct v2d as [sv2d v2d].
```

```
destruct v2c as [sv2c v2c].
simpl in u1.
simpl in u2.
simpl in v1.

simpl in v2.

destruct pv as [pfpv pv].
simpl in pfpv. induction pfpv.
simpl in pv.

destruct pu as [pfpu pu].
simpl in pfpu. induction pfpu.
simpl in pu.




destruct P1 as [p1  [q1 P1]].
destruct p1 as [pfp1 p1].
simpl in pfp1.

induction pfp1.
simpl in p1.


destruct a' as [sa' a'].
destruct b' as [sb' b'].

destruct q1 as [pfq1 q1].
simpl in pfq1. induction pfq1.
simpl in q1.
simpl in P1.


destruct P2 as [p2  [q2 P2]].
destruct p2 as [pfp2 p2].
simpl in pfp2.
```

```
induction pfp2.
simpl in p2.


destruct q2 as [pfq2 q2].
simpl in pfq2. induction pfq2.
simpl in q2.
simpl in P2.


simpl in P3.

destruct P3 as  [p3   [q3  P3]].
destruct p3 as [pfp3 p3].
simpl in pfp3. induction pfp3.
simpl in p3.
destruct q3 as [pfq3 q3].
simpl in pfq3. induction pfq3.
simpl in q3.
simpl in P3.


simpl in P4.
destruct P4 as  [p4   [q4  P4]].
destruct p4 as [pfp4 p4].
simpl in pfp4. induction pfp4.
simpl in p4.
destruct q4 as [pfq4 q4].
simpl in pfq4. induction pfq4.
simpl in q4.
simpl in P4.


simpl in P5.

destruct P5 as  [p5   [q5  P5]].
destruct p5 as [pfp5 p5].
```

```
simpl in ptp5. induction ptp5.
simpl in p5.
destruct q5 as [pfq5 q5].
simpl in pfq5.

assert (Identity (Identity_refl sv2c) pfq5) as G.
apply hedberg.
apply stages_DecId.
induction G.

simpl in q5.
simpl in P5.
simpl in x.
simpl in y.
simpl in H.


simpl.
split.
apply setoidmapextensionality.
apply setoidmapextensionality.
exact H.
apply setoidmapextensionality.
apply setoidmapextensionality.
apply setoidmapextensionality.
exact H.
Defined.


Definition make_pbpairing_arr
  (m : meetingarrows the_categ)
  (h : catarr the_categ)
  (k : catarr the_categ)
  (P : Square'  (projT1 m) (projT1 (projT2 m)) h k):
catarr the_categ.
  exists (catdom the_categ h).
  exists (pullbackob  m).
  destruct  m as [f  [g P']].
```

```
  simpl in P.
  simpl.
  apply (pb_pairing f g P' h k P).
Defined.


Definition makecomposable
(f g: catarr the_categ)(p: catdom the_categ f ≈
catcod the_categ g):
catcms the_categ.
exists g.
exists f.
apply setoidsym.
simpl.
simpl in p.
exact p.
Defined.

Lemma dom_pullback1: (forall m: meetingarrows
the_categ,
(catdom the_categ (pullback1 m) ≈ pullbackob m)).
Proof.
intro m.
apply  setoidrefl.
Defined.

Lemma dom_pullback2: (forall m: meetingarrows
the_categ,
(catdom the_categ (pullback2 m) ≈ pullbackob m)).
Proof.
intro m.
apply setoidrefl.
Defined.


Lemma Jointly: (forall m : meetingarrows the_categ,
   Jointly_monic (pullback1 m) (pullback2 m)).
Proof.
```

```
intro m.
assert ((catdom (cat_from_family T) (pullback1 m)) ≈
(catdom (cat_from_family T) (pullback1 m))) as thep.
apply (setoidrefl _(pullbackob m)).

apply (Jointly_monic_special _ _  (pullback1 m)
(pullback2 m) thep).

destruct m as  [f   [g  P]].
destruct f as  [df   [cf  f]].
destruct g as  [dg   [cg  g]].
simpl in P.

destruct df as [sdf df].
destruct cf as [scf cf].
destruct dg as [sdg dg].
destruct cg as [scg cg].

unfold Jointly_monic_setoidmaps.
intros x y.
simpl in f.
simpl in g.
destruct P as  [pfP P].
simpl in pfP.
induction pfP.
simpl in P.
destruct thep  as [sthep thep].
simpl in sthep.

assert (Identity (Identity_refl (triple sdf sdg scf))
sthep) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in thep.

destruct thep as [p1 thep].
destruct thep as [p2 thep].
```

```
destruct thep as [p3 thep].
destruct thep as [p4 thep].
destruct thep as [Q1 Q2].


simpl.
simpl in x.
simpl in y.
destruct x as [x1  [x2 Px]].
destruct y as [y1  [y2 Py]].
simpl.
intros H H'.
split.
apply H.
apply (setoidfamilyirrevgeneral _ _ _ _ _ _ _ H').
Defined.



Lemma cod_pbpairing
  (m : meetingarrows the_categ)
  (h : catarr the_categ)
  (k : catarr the_categ)
  (P : Square'  (projT1 m) (projT1 (projT2 m)) h k):
(catcod the_categ (make_pbpairing_arr m h k P) ≈
pullbackob m).
Proof.
apply setoidrefl.
Defined.


Theorem Pullbackexists (m: meetingarrows the_categ):
Pullback (projT1 m) (projT1 (projT2 m)) (pullback1
m)  (pullback2 m).
unfold Pullback.
split.
apply Square_to_Square'.
apply Is_square_lemma.
```

```
split.
(* joint monicity of projection *)
apply Jointly.
(* universal map exists*)

intros h k P.
exists (make_pbpairing_arr m h k P).
split.
unfold Comp.

assert (catcod the_categ (make_pbpairing_arr m h k P)
≈ catdom the_categ (pullback1 m)) as q.
apply cod_pbpairing.

exists (makecomposable (pullback1 m)
(make_pbpairing_arr m h k P) q).
split.
apply setoidrefl.
split.
apply setoidrefl.

destruct m as [f  [g P']].
destruct f as [df  [cf f]].
destruct g as [dg  [cg g]].
simpl in P'.
unfold Square' in P.
destruct P as [u  [v  [P1  [P2  [P3  [P4  P5]]]]]].
destruct u as [u1  [u2 pu]].
destruct v as [v1  [v2 pv]].
destruct u1 as [u1d  [u1c u1m]].
destruct u2 as [u2d  [u2c u2m]].
destruct v1 as [v1d  [v1c v1m]].
destruct v2 as [v2d  [v2c v2m]].
simpl in pu.
simpl in pv.

destruct df as [sdf df].
destruct cf as [scf cf].
```

```
destruct dg as [sdg dg].
destruct cg as [scg cg].
simpl in f.
simpl in g.


destruct P' as [pfP' P'].
simpl in pfP'.
induction pfP'.
simpl in P'.

destruct u1d as [pfu1d u1d].
destruct u1c as [pfu1c u1c].
destruct u2d as [pfu2d u2d].
destruct u2c as [pfu2c u2c].

destruct v1d as [pfv1d v1d].
destruct v1c as [pfv1c v1c].
destruct v2d as [pfv2d v2d].
destruct v2c as [pfv2c v2c].
simpl in u1m.
simpl in u2m.

destruct pu as [pfpu pu].
simpl in pfpu.
induction pfpu.
simpl in pu.

simpl in v1m.
simpl in v2m.
destruct pv as [pfpv pv].
simpl in pfpv.
induction pfpv.
simpl in pv.
destruct h as [dh  [ch h]].
destruct k as [dk  [ck k]].

destruct dh as [sdh dh].
```

```
destruct ch as [sch ch].
destruct dk as [sdk dk].
destruct ck as [sck ck].
simpl in h.

simpl in k.

simpl in P1.
simpl in P2.
simpl in P3.
simpl in P4.
simpl in P5.

destruct P1 as [p1 [q1 P1]].
destruct P2 as [p2 [q2 P2]].
destruct P3 as [p3 [q3 P3]].
destruct P4 as [p4 [q4 P4]].
destruct P5 as [p5 [q5 P5]].

destruct p1 as  [pfp1 p1].
simpl in pfp1.
induction pfp1.
simpl in p1.
destruct q1 as  [pfq1 q1].
simpl in pfq1.
induction pfq1.
simpl in q1.
destruct p2 as  [pfp2 p2].
simpl in pfp2.
induction pfp2.
simpl in p2.
destruct q2 as  [pfq2 q2].
simpl in pfq2.
induction pfq2.
simpl in q2.
destruct p3 as  [pfp3 p3].
simpl in pfp3.
induction pfp3.
```

```
simpl in p3.
destruct q3 as  [pfq3 q3].
simpl in pfq3.
induction pfq3.

simpl in q3.
destruct p4 as  [pfp4 p4].
simpl in pfp4.
induction pfp4.
simpl in p4.
destruct q4 as  [pfq4 q4].
simpl in pfq4.
induction pfq4.
simpl in q4.
destruct p5 as  [pfp5 p5].
simpl in pfp5.
induction pfp5.
simpl in p5.
destruct q5 as  [pfq5 q5].
simpl in pfq5.

assert (Identity (Identity_refl pfv2c) pfq5) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in P1.
simpl in P2.
simpl in P3.
simpl in P4.
simpl in P5.
simpl in q5.

simpl in q.
destruct q as [pfq q].
simpl in pfq.

assert (Identity (Identity_refl (triple pfu1c pfv1c
pfv2c)) pfq) as G.
```

```
apply hedberg.
apply stages_DecId.
induction G.
simpl in q.


destruct q as [qe1 [qe2 [qe3 [qe4 q]]]].
simpl.

assert (omegalimsetoideq ifm (existT (λ n : stages,
projT1 (ifm n)) pfu1d dh)
(existT (λ n : stages, projT1 (ifm n)) pfu1d dh)) as
epart.
exists (Identity_refl pfu1d).
simpl.
apply setoidrefl.
exists epart.
assert (omegalimsetoideq ifm (existT (λ n : stages,
projT1 (ifm n)) pfu1c df)
       (existT (λ n : stages, projT1 (ifm n)) pfu1c
ch)) as dpart.
exists (Identity_refl pfu1c).
simpl.
apply (q1 ⊙ (pu ⁻¹) ⊙ (p2 ⁻¹)).
exists dpart.
destruct dpart as [dpart1 dpart2].
destruct epart as [epart1 epart2].
simpl in epart1.
assert (Identity (Identity_refl pfu1d) epart1) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in epart2.
simpl in dpart1.
assert (Identity (Identity_refl pfu1c) dpart1) as G.
apply hedberg.
apply stages_DecId.
induction G.
```

```
simpl in dpart2.
destruct q as [qL qR].
intro x.
simpl.

apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.
apply setoidsym.
assert ((projT2 (ifm pfu1c)) • ((dpart2 ⊙ qe1 ⁻¹) ⊙
p2 ⊙ pu ⊙ q1 ⁻¹) (h x)
≈  (h x)) as eq.
apply setoidfamilyrefgeneral.
assert (h x ≈ h ((projT2 (ifm pfu1d)) • epart2 x)) as
eq2.
apply setoidmapextensionality.
apply setoidsym.
apply setoidfamilyrefgeneral.
swesetoid.

(* second equation *)


unfold Comp.

assert (catcod the_categ (make_pbpairing_arr m h k P)
≈ catdom the_categ (pullback2 m)) as q.
apply cod_pbpairing.

exists (makecomposable (pullback2 m)
(make_pbpairing_arr m h k P) q).
split.
apply setoidrefl.
split.
apply setoidrefl.

destruct m as [f  [g P']].
destruct f as ⌈df  ⌈cf f⌉⌉.
```

```
destruct g as [dg  [cg g]].
simpl in P'.
unfold Square' in P.
destruct P as [u  [v  [P1  [P2  [P3  [P4  P5]]]]]].

destruct u as [u1  [u2 pu]].
destruct v as [v1  [v2 pv]].
destruct u1 as [u1d  [u1c u1m]].
destruct u2 as [u2d  [u2c u2m]].
destruct v1 as [v1d  [v1c v1m]].
destruct v2 as [v2d  [v2c v2m]].
simpl in pu.
simpl in pv.

destruct df as [sdf df].
destruct cf as [scf cf].
destruct dg as [sdg dg].
destruct cg as [scg cg].
simpl in f.
simpl in g.

destruct P' as [pfP' P'].
simpl in pfP'.
induction pfP'.
simpl in P'.

destruct u1d as [pfu1d u1d].
destruct u1c as [pfu1c u1c].
destruct u2d as [pfu2d u2d].
destruct u2c as [pfu2c u2c].

destruct v1d as [pfv1d v1d].
destruct v1c as [pfv1c v1c].
destruct v2d as [pfv2d v2d].
destruct v2c as [pfv2c v2c].
simpl in u1m.
simpl in u2m.
```

```
destruct pu as [pfpu pu].
simpl in pfpu.
induction pfpu.
simpl in pu.


simpl in v1m.
simpl in v2m.
destruct pv as [pfpv pv].
simpl in pfpv.
induction pfpv.
simpl in pv.
destruct h as [dh  [ch h]].
destruct k as [dk  [ck k]].

destruct dh as [sdh dh].
destruct ch as [sch ch].
destruct dk as [sdk dk].
destruct ck as [sck ck].
simpl in h.
simpl in k.

simpl in P1.
simpl in P2.
simpl in P3.
simpl in P4.
simpl in P5.

destruct P1 as [p1 [q1 P1]].
destruct P2 as [p2 [q2 P2]].
destruct P3 as [p3 [q3 P3]].
destruct P4 as [p4 [q4 P4]].
destruct P5 as [p5 [q5 P5]].

destruct p1 as  [pfp1 p1].
simpl in pfp1.
induction pfp1.
simpl in p1.
```

```
simpl in p2.
destruct q1 as  [pfq1 q1].
simpl in pfq1.
induction pfq1.
simpl in q1.

destruct p2 as  [pfp2 p2].
simpl in pfp2.
induction pfp2.
simpl in p2.
destruct q2 as  [pfq2 q2].
simpl in pfq2.
induction pfq2.
simpl in q2.
destruct p3 as  [pfp3 p3].
simpl in pfp3.
induction pfp3.
simpl in p3.
destruct q3 as  [pfq3 q3].
simpl in pfq3.
induction pfq3.
simpl in q3.
destruct p4 as  [pfp4 p4].
simpl in pfp4.
induction pfp4.
simpl in p4.
destruct q4 as  [pfq4 q4].
simpl in pfq4.
induction pfq4.
simpl in q4.
destruct p5 as  [pfp5 p5].
simpl in pfp5.
induction pfp5.
simpl in p5.
destruct q5 as  [pfq5 q5].
simpl in pfq5.

assert (Identity (Identity_refl pfv2c) pfq5) as G.
apply hedberg.
```

```
apply hedberg.
apply stages_DecId.
induction G.
simpl in P1.
simpl in P2.

simpl in P3.
simpl in P4.
simpl in P5.
simpl in q5.

simpl in q.
destruct q as [pfq q].
simpl in pfq.

assert (Identity (Identity_refl (triple pfu1c pfv1c
pfv2c)) pfq) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in q.

destruct q as [qe1 [qe2 [qe3 [qe4 q]]]].
simpl.

assert (omegalimsetoideq ifm (existT (λ n : stages,
projT1 (ifm n)) pfu1d dh)
(existT (λ n : stages, projT1 (ifm n)) pfu1d dk)) as
epart.
exists (Identity_refl pfu1d).
simpl.
apply  (p3 ⊙ p5 ⊙ (p1 ⁻¹)).
exists epart.
assert (omegalimsetoideq ifm (existT (λ n : stages,
projT1 (ifm n)) pfv1c dg)
       (existT (λ n : stages, projT1 (ifm n)) pfv1c
ck)) as dpart.
exists (Identity_refl pfv1c).
simpl
```

```
simpl.
apply (q3 ⊙ (pv ⁻¹) ⊙ (p4 ⁻¹)).
exists dpart.
destruct dpart as [dpart1 dpart2].
destruct epart as [epart1 epart2].

simpl in epart1.
assert (Identity (Identity_refl pfu1d) epart1) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in epart2.
simpl in dpart1.
assert (Identity (Identity_refl pfv1c) dpart1) as G.
apply hedberg.
apply stages_DecId.
induction G.
simpl in dpart2.
destruct q as [qL qR].
intro x.
simpl.
apply setoidsym.
apply setoidfamilycmpgeneral_3.
apply setoidfamilycmpgeneral_3.

assert (k ((projT2 (ifm pfu1d)) • epart2 x) ≈
k((projT2 (ifm pfu1d)) • (p3 ⊙ p5 ⊙ p1 ⁻¹) x)) as
eq1.

apply setoidmapextensionality.
apply setoidfamilyirr.
assert ((projT2 (ifm pfv1c)) • ((dpart2 ⊙ qe2 ⁻¹) ⊙
p4 ⊙ pv ⊙ q3 ⁻¹)
     (k ((projT2 (ifm pfu1d)) • (p3 ⊙ p5 ⊙ p1 ⁻¹) x))
≈
(k ((projT2 (ifm pfu1d)) • (p3 ⊙ p5 ⊙ p1 ⁻¹) x))) as
eq2.
apply setoidfamilyrefgeneral.
swesetoid
```

```
swesetold.
Defined.



(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
(* EOF *)
```

(* EOF *)
(* EOF *)
(* EOF *)