```
(*
   Families of setoids and
   locally cartesian closure of the
   e-category of setoids.

   Erik Palmgren, October 15, 2012.

   This code illustrates how to prove
   that the e-category of setoids is
   LCC by using setoid versions of
   Pi and Sigma for proof-irrelevant
   families of setoids.


   Written in
   Coq 8.3pl2/Coq 8.3pl3 with UTF8-encoding.

*)

(* We use Olov Wilander's development and notation
for "Swedish style" setoids, i.e. setoids where the
truth-values of equivalence relations are in Set
rather than in Prop as in Coq ("French style"
setoids).
*)

Require Import PropasTypesUtf8Notation
PropasTypesBasics_mod SwedishSetoids_mod.

(* First some generalities about families of setoids
  and dependent setoid constructions from forthcoming
  joint work with Olov Wilander *)

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).

(* Proof irrelevant setoid families *)
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 familymapcomposition {A B: setoid} (F:
setoidfamily B) (f: A ⇒ B)
  : setoidfamily A.
apply (Build_setoidfamily A
  (λ a, F (f a))
  (λ a a' p, F • (f ↱ p))).
intros ?; apply setoidfamilyrefgeneral.
intros ? ? ? ?; apply setoidfamilyirr.
intros ? ? ? ? ?; apply setoidfamilycmpgeneral.
Defined.

Notation "F ★ f" := (familymapcomposition F f) (at
level 9).

Definition Σsetoid (A: setoid) (F: setoidfamily A):
setoid.
apply (Build_setoid (∃a: A, F a)
  (λ p q, match (p, q) with (existT a b, existT a'
b') =>
            ∃e: a ≈ a', F•e b ≈ b'
          end)).
intros [a b]. exists (setoidrefl _ _). apply
setoidfamilyref.
intros [a b] [a' b'] [p e]. exists p⁻¹. apply
setoidtra with (F•p⁻¹(F•p b)).
apply setoidmapextensionality; apply setoidsym;
```

```
assumption.
apply setoidtra with (F•(p⁻¹⊙p)b). apply
setoidfamilycmp.
apply setoidfamilyrefgeneral.
intros [a b] [a' b'] [a'' b''] [p e] [q e'].
exists (q⊙p). apply setoidtra with (F•q (F•p b)).
apply setoidsym.
apply setoidfamilycmp.
apply setoidtra with (F•q b'). apply
setoidmapextensionality; assumption.
assumption.
Defined.

Definition Πsetoid (A: setoid) (F: setoidfamily A):
setoid.
apply (Build_setoid (∃f: ∀a: A, F a, ∀a a': A, ∀p: a
≈ a', F•p (f a) ≈ f a')
  (λ F G, match (F, G) with (existT f fext, existT g
gext) =>
            ∀a: A, f a ≈ g a
          end)).
intros [f _] ?; apply setoidrefl.
intros [f _] [g _] p a; apply setoidsym; auto.
intros [f _] [g _] [h _] p q a; eauto using
setoidtra.
Defined.

(* End of generalities about dependent types *)

(* Now prove that the setoids are locally cartesian
  closed in the sense of E-categories.*)

(* The next few definitions and theorems gives the
basic properties of the E-category of slices of the
setoids above a fixed setoid. *)

Record slice (X:setoid) :=
```

```
  {
    overobj :> setoid;
    downarr : overobj ⇒ X
  }.

Definition mslice {X:setoid} (A:setoid)
   (a: A ⇒ X): slice X.
  apply (Build_slice X A a).
Defined.

Record slicemap {X:setoid}(a b:slice X):=
  {
    upper :> (overobj X a) ⇒ (overobj X b);
    triangle : ∀x: overobj X a,
                downarr X b (upper x)  ≈ downarr X a x
  }.

Definition slicemorph {X:setoid}(a b:slice X):setoid.
  apply (Build_setoid (slicemap a b)
        (fun  f g => upper a b f  ≈ upper a b g)).
  intro. swesetoid.
  intro. swesetoid.
  intro. swesetoid.
Defined.

Definition uppermap
   {X:setoid}{a b:slice X}(z: slicemorph a b):= upper
a b z.

Theorem slicemorph_eqlemma2
  {X:setoid}{a b:slice X}(f f': slicemorph a b)
  (p: f  ≈ f'): ∀x:a, uppermap f x  ≈ uppermap f' x.
  Proof.
  intro x.
  apply p.
Defined.
```

```
Definition trianglepf
   {X:setoid}{a b:slice X}(z: slicemorph a b):=
triangle a b z.

Definition id_slice {X:setoid}(a:slice X): slicemorph
a a.
  apply (Build_slicemap X a a idmap).
  intro x. simpl. swesetoid.
Defined.


Definition cmp_slice_helper2 {X:setoid}(a b c: slice
X):
   (slicemorph b c) ->
   (slicemorph a b) -> (slicemorph a c).
  intros f g.
  apply (Build_slicemap X a c
           ((upper b c f) ∘ (upper a b g))).
  intro x.
  destruct f as [uf pf].
  destruct g as [ug pg].
  simpl.
  specialize pf with (ug x).
  swesetoid.
Defined.

Definition cmp_slice_helper {X:setoid}(a b c: slice
X):
   (slicemorph b c) ->
   (slicemorph a b) ⇒ (slicemorph a c).
  intro f.
  apply (Build_setoidmap
          (slicemorph a b) (slicemorph a c)
          (fun g => cmp_slice_helper2 a b c f g)).
  intros g g' P t.
  simpl.
  apply setoidmapextensionality.
```

```
  apply P.
Defined.

Definition cmp_slice {X:setoid}{a b: slice X}(c:slice
X):
   (slicemorph b c) ⇒ (slicemorph a b) ⇒ (slicemorph
a c).
  apply (Build_setoidmap
          (slicemorph b c)
          ((slicemorph a b) ⇒ (slicemorph a c))
       (fun f => (cmp_slice_helper a b c f))).
  intros f f' P g t.
  simpl.
  apply P.
Defined.


Theorem cmp_slice_lemma {X:setoid}{a b: slice X}
  (c:slice X)(f: slicemorph b c)(g:slicemorph a b)
  (x: a)(y: c):
  (uppermap (cmp_slice c  f g)) x  ≈ y
  ->
  (uppermap f) (uppermap g x) ≈ y.
Proof.
  intro H.
  apply H.
Defined.

Theorem cmp_slice_assoc (X:setoid)(a b c d:slice X)
  (f: slicemorph c d)
  (g: slicemorph b c)
  (h: slicemorph a b):
    (cmp_slice d (cmp_slice d f g) h)  ≈
    (cmp_slice d f (cmp_slice c g h)).
Proof.
  destruct f as [uf pf].
  destruct g as [ug pg].
```

```
  destruct h as [uh ph].
  simpl.
  intro x. apply setoidrefl.
Defined.

Theorem cmp_slice_idleft (X:setoid)(a b:slice X)
  (f: slicemorph a b):
    (cmp_slice b (id_slice b) f)  ≈ f.
Proof.
  destruct f as [uf pf].
  simpl.
  intro x. apply setoidrefl.
Defined.

Theorem cmp_slice_idright (X:setoid)(a b:slice X)
  (f: slicemorph a b):
    (cmp_slice b f (id_slice a))  ≈ f.
Proof.
  destruct f as [uf pf].
  simpl.
  intro x. apply setoidrefl.
Defined.

(* End of basic slice constructions *)

(* Construction of pullbacks in Setoids *)

Definition PullbackObj (X:setoid)(A B:setoid)
      (f:A ⇒ X)(g:B ⇒ X):setoid.
  apply (Build_setoid  (∃z:A ⊗ B,
      f (fst_setoid z) ≈ g (snd_setoid z))
      (λ p q, projT1 p ≈ projT1 q)
          ).
  intro p. destruct p. apply setoidrefl.
  intros p q. destruct p. destruct q. apply
setoidsym.
  intros p q r. destruct p. destruct q.  destruct r.
```

```
apply setoidtra.
Defined.

Definition PullbackProj1 {X:setoid}{A B:setoid}
     (f:A ⇒ X)(g:B ⇒ X): PullbackObj X A B f g  ⇒ A.
  apply (Build_setoidmap (PullbackObj X A B f g) A
        (fun p => fst_setoid (projT1 p))).
  intros p p' P. destruct p as  [z p].
  destruct p' as  [z' p'].  destruct z. destruct z'.
  simpl in P.
  simpl.
  apply P.
Defined.


Definition PullbackProj2 {X:setoid}{A B:setoid}
     (f:A ⇒ X)(g:B ⇒ X): PullbackObj X A B f g  ⇒ B.
  apply (Build_setoidmap (PullbackObj X A B f g) B
        (fun p => snd_setoid (projT1 p))).
  intros p p' P. destruct p as  [z p].
  destruct p' as  [z' p'].  destruct z. destruct z'.
  simpl in P.
  simpl.
  apply P.
Defined.


Definition PullbackBracket_helper {X A B Q:setoid}
     (f:A ⇒ X)(g:B ⇒ X)(p:Q ⇒ A)
       (q:Q ⇒ B)(P:f ∘ p  ≈ g ∘ q):
        Q -> PullbackObj X A B f g.
  intro t.
  exists ((p t), (q t)).
  simpl.
  apply (P t).
Defined.
```

```
Definition PullbackBracket{X A B Q:setoid}
     (f:A ⇒ X)(g:B ⇒ X)(p:Q ⇒ A)
       (q:Q ⇒ B)(P:f ∘ p  ≈ g ∘ q):
        Q ⇒ PullbackObj X A B f g.
   apply (Build_setoidmap  Q (PullbackObj X A B f g)
         (PullbackBracket_helper f g p q P)).
   intros x y H.
   simpl.
   split.
   apply setoidmapextensionality. assumption.
   apply setoidmapextensionality. assumption.
Defined.

Theorem Pullback_eq_lemma  (X:setoid)(A B:setoid)
      (f:A ⇒ X)(g:B ⇒ X)(z z': PullbackObj X A B f
g):
   (PullbackProj1 f g z)  ≈ (PullbackProj1 f g z') ->
   (PullbackProj2 f g z)  ≈ (PullbackProj2 f g z')
    -> z ≈ z'.
Proof.
    destruct z as [z p]. destruct z' as [z' p'].
    destruct z as [x y]. destruct z' as [x' y'].
    simpl.
    intros.
    split. assumption. assumption.
Defined.


Theorem IsPullback (X:setoid)(A B:setoid)
     (f:A ⇒ X)(g:B ⇒ X):
     f ∘ (PullbackProj1 f g)  ≈ g ∘  (PullbackProj2 f
g)
      ∧
     (∀Q:setoid, ∀p:Q ⇒ A,  ∀q:Q ⇒ B,
        ∀H:(f ∘ p  ≈ g ∘ q),
```

```
          ((PullbackProj1 f g) ∘
             (PullbackBracket f g p q H)  ≈ p)
        ∧
         ((PullbackProj2 f g) ∘
             (PullbackBracket f g p q H)  ≈ q)
        ∧ (∀r:Q ⇒ (PullbackObj X A B f g),
           ((PullbackProj1 f g) ∘ r  ≈ p)
        ->
         ((PullbackProj2 f g) ∘ r  ≈ q)
        ->  r  ≈ (PullbackBracket f g p q H))).
Proof.
  split.
  intro t.
  destruct t as [z p].
  destruct z as [x y].
  simpl.
  apply p.

  intros Q p q H.
  split.
  intro t.
  simpl.
  apply setoidrefl.
  split.
  intro t.
  simpl.
  apply setoidrefl.

  intros r H1 H2.
  intro t.
  apply Pullback_eq_lemma.
  apply (H1 t).
  apply (H2 t).
Defined.

(* Next construct the pullback functor between slices
of setoids: f^*: Setoids/Y -> Setoids/X *)
```

```
Theorem pbfunctor_obs
  (X Y:setoid)(f: X ⇒ Y)(a :slice Y): slice X.
Proof.
  destruct a as [A a].
  apply (Build_slice X (PullbackObj Y X A f a)
                       (PullbackProj1 f a)).
Defined.


Theorem pbfunctor_mor_helper
  (X Y:setoid)(f: X ⇒ Y)(a b :slice Y):
  slicemorph a b -> slicemorph (pbfunctor_obs X Y f
a)
                               (pbfunctor_obs X Y f
b).
Proof.
  intro g.
  destruct a as [A a].
  destruct b as [B b].
  destruct g as [g pf].
  simpl in g.
  simpl in pf.
  assert (f ◦ (PullbackProj1 f a) ≈
     b ◦ (g ◦ (PullbackProj2 f a)))
   as H.
  intro t.
  specialize pf with (PullbackProj2 f a t).
  assert  (f ◦ (PullbackProj1 f a)  ≈
            a ◦ (PullbackProj2 f a)) as H1.
  apply (IsPullback Y X A f a).
  assert  (f  (PullbackProj1 f a t)  ≈
            a  (PullbackProj2 f a t)) as H2.
  apply (H1 t).
  swesetoid.

  apply
```

```
  apply
   (Build_slicemap X
           (pbfunctor_obs X Y f {| overobj := A;
downarr := a |})
           (pbfunctor_obs X Y f {| overobj := B;
downarr := b |})
           (PullbackBracket f b
              (PullbackProj1 f a)
              (g ∘ (PullbackProj2 f a))
              H)).
  intro x.
  unfold pbfunctor_obs.
  simpl.
  apply setoidrefl.
Defined.


Theorem pbfunctor_mor
  (X Y:setoid)(f: X ⇒ Y)(a b :slice Y):
  slicemorph a b  ⇒ slicemorph (pbfunctor_obs X Y f
a)
                               (pbfunctor_obs X Y f
b).
Proof.
  apply (Build_setoidmap (slicemorph a b)
          (slicemorph (pbfunctor_obs X Y f a)
                      (pbfunctor_obs X Y f b))
         (pbfunctor_mor_helper X Y f a b)).
  intros g h H.
  intros t.
  unfold pbfunctor_obs in t.
  destruct a.  destruct b. destruct g. destruct h.
  destruct t.

  unfold pbfunctor_mor_helper.
  simpl.
  split.
  apply setoidrefl
```

```
  apply setoidrefl.
  simpl in H.
  apply H.
Defined.


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

Theorem pbfunctor_id
  (X Y:setoid)(f: X ⇒ Y)(a:slice Y):
  pbfunctor_mor X Y f a a (id_slice a)
     ≈ id_slice (pbfunctor_obs X Y f a).
Proof.
  intro z.
  destruct a.
  destruct z.
  destruct x.
  simpl.
  split. apply setoidrefl. apply setoidrefl.
Defined.

Theorem pbfunctor_cmp
  (X Y:setoid)(f: X ⇒ Y)(a b c:slice Y)
    (g: slicemorph b c)(h: slicemorph a b):
  pbfunctor_mor X Y f a c (cmp_slice c g h)
     ≈
   (cmp_slice (pbfunctor_obs X Y f c)
              (pbfunctor_mor X Y f b c g)
              (pbfunctor_mor X Y f a b h)).
Proof.
  intro z.
  destruct a. destruct b.  destruct c.
  destruct g.   destruct h.
  simpl.
  split.
  apply setoidrefl
```

```
  apply setoidrefl.
  apply setoidrefl.
Defined.

(* End of pullback functor related definitions *)

(* We need to consider fibers f^-1(y) of a
setoid map f. *)

Definition fib_setoid  {X Y:setoid}(f:X ⇒ Y)
(y:Y):setoid.
  apply (Build_setoid (∃x:X, f x ≈ y)
            (λ p q, projT1 p ≈ projT1 q)).
  intro z.
  destruct z as  [x p].
  apply setoidrefl.
  intros z z'.
  destruct z as  [x p].
  destruct z' as  [x' p'].
  apply setoidsym.
  intros z z' z''.
  destruct z as  [x p].
  destruct z' as  [x' p'].
  destruct z'' as  [x'' p''].
  apply setoidtra.
Defined.


Definition fib_map_helper {X Y:setoid}(f:X ⇒ Y)
   (y y':Y)(p:y ≈ y'):
     (fib_setoid f y) -> (fib_setoid f y').
  intro z.
  exists (projT1 z).
  apply setoidtra with y.
  apply (projT2 z).
  exact p.
Defined.
```

```
Definition fib_map {X Y:setoid}(f:X ⇒ Y)
  (y y':Y)(p:y ≈ y'):
   (fib_setoid f y) ⇒ (fib_setoid f y').
  apply (Build_setoidmap
         (fib_setoid f y) (fib_setoid f y')
         (fib_map_helper f y y' p)).
  intros z z'.
  destruct z as  [x q].
  destruct z' as  [x' q'].
  simpl.
  intro H. assumption.
Defined.

Definition fib {X Y:setoid}(f:X ⇒ Y): setoidfamily Y.
  apply (Build_setoidfamily Y
            (fib_setoid f)
            (fib_map f)).
  intros x z.
  destruct z as  [u q].
  simpl.
  apply setoidrefl.
  intros x x' H H1 z.
  destruct z as  [u q].
  simpl.
  apply setoidrefl.
  intros x x' x'' H H1 z.
  destruct z as  [u q].
  simpl.
  apply setoidrefl.
Defined.

Definition fib_proj {X Y:setoid}(f:X ⇒ Y)(y:Y):
  (fib f y)   ⇒ X.
  apply (Build_setoidmap (fib f y) X (fun z => projT1
z)).
  intros z z' H.
```

```
  destruct z as  [u q].
  destruct z' as  [u' q'].
  simpl in H.
  simpl.
  assumption.
Defined.

(* Now work towards constructing the right adjoint
to the pullback functor.
*)

Definition Pi_setoid {A X Y:setoid}
   (a: A ⇒ X)(f:X  ⇒ Y)(y:Y):=
  Πsetoid (fib f y) ((fib a)  ★ (fib_proj f y)).


Definition Pi_setoid_map_helper {X Y:setoid}
   (F: setoidfamily X)(f:X  ⇒ Y)(y y':Y)(p: y ≈ y')
   (h:  ∀x : (fib f) y, F ★ (fib_proj f y) x):
   (∀x : (fib f) y', F ★ (fib_proj f y') x).
  intro x.
  specialize h
     with ((fib f) • p ⁻¹ x).
  simpl in h.
  simpl.
  simpl in x.
  destruct x as  [u q'].
  simpl.
  simpl in h.
  exact h.
Defined.

Definition Pi_setoid_map_helper2 {X Y:setoid}
   (F: setoidfamily X)(f:X  ⇒ Y)(y y':Y)(p: y ≈ y'):
    Πsetoid ((fib f) y)  (F ★ (fib_proj f y)) ->
    Πsetoid ((fib f) y') (F ★ (fib_proj f y'))
```

```
    Πsetoid ((fib f) y') (F ★ (fib_proj f y')).
  intro h.
  destruct h as [h pf].
  exists (Pi_setoid_map_helper F f y y' p h).
  intros x x' q.
  specialize (pf ((fib f) • p ⁻¹ x)).
  specialize (pf ((fib f) • p ⁻¹ x')).
  destruct x as  [u r].
  destruct x' as  [u' r'].
  simpl in pf.
  simpl in q.
  specialize (pf q).

  simpl.
  assumption.
Defined.


Definition Pi_setoid_map_helper3 {X Y:setoid}
   (F: setoidfamily X)(f:X  ⇒ Y)(y y':Y)(p: y ≈ y'):
    Πsetoid ((fib f) y)  (F ★ (fib_proj f y))  ⇒
    Πsetoid ((fib f) y') (F ★ (fib_proj f y')).
   apply
     (Build_setoidmap
       (Πsetoid ((fib f) y)  (F ★ (fib_proj f y)))
       (Πsetoid ((fib f) y') (F ★ (fib_proj f y')))
       (Pi_setoid_map_helper2 F f y y' p)).
   intros g h P.
   destruct g as [g pg].
   destruct h as [h ph].

   simpl.
   simpl in P.
   intro t.
   destruct t as [t pt].
   simpl.
   specialize (P  (existT (λ x : X, f x ≈,{ Y }y) t
```

```
(p ⁻¹ ⊙ pt))).
   assumption.
Defined.



Definition Pi_family {X Y:setoid}
   (F: setoidfamily X)(f:X  ⇒ Y): setoidfamily Y.
   apply (Build_setoidfamily Y
            (fun y => Πsetoid ((fib f) y)  (F ★
(fib_proj f y)))
            (Pi_setoid_map_helper3 F f)).
  intros x h.
  destruct h as [h ph].
  simpl.
  intro t.
  destruct t as [t pt].
  simpl.
  specialize (ph (existT (λ x0 : X, f x0 ≈,{ Y }x) t
((setoidrefl Y x) ⁻¹ ⊙ pt))).
  specialize (ph (existT (λ x0 : X, f x0 ≈,{ Y }x) t
pt)).
  assert (existT (λ x0 : X, f x0 ≈,{ Y }x) t
((setoidrefl Y x) ⁻¹ ⊙ pt)
        ≈,{ (fib f) x }existT (λ x0 : X, f x0 ≈,{ Y
}x) t pt) as p.
  simpl.
  apply setoidrefl.
  specialize (ph p).
  simpl in ph.
  assert (F • p
        (h (existT (λ x0 : X, f x0 ≈,{ Y }x) t
((setoidrefl Y x) ⁻¹ ⊙ pt))) ≈
     h (existT (λ x0 : X, f x0 ≈,{ Y }x) t
((setoidrefl Y x) ⁻¹ ⊙ pt))).
  apply setoidfamilyrefgeneral.
```

```
  swesetoid.
  intros x y p q h.
  destruct h as [h ph].
  simpl.
  intro t.
  destruct t as [t pt].
  simpl.
  specialize (ph (existT (λ x0 : X, f x0 ≈,{ Y }x) t
(p ⁻¹ ⊙ pt))).
  specialize (ph (existT (λ x0 : X, f x0 ≈,{ Y }x) t
(q ⁻¹ ⊙ pt))).
  simpl in ph.
  specialize (ph (setoidrefl X t)).
  assert ( F • (setoidrefl X t)
        (h (existT (λ x0 : X, f x0 ≈,{ Y }x) t (p ⁻¹
⊙ pt))) ≈ (h (existT (λ x0 : X, f x0 ≈,{ Y }x) t (p
⁻¹ ⊙ pt)))).
  apply setoidfamilyrefgeneral.
  swesetoid.

  intros x y z p q h.
  destruct h as [h ph].
  simpl.
  intro t.
  destruct t as [t pt].
  simpl.
  specialize (ph (existT (λ x0 : X, f x0 ≈,{ Y }x) t
(p ⁻¹ ⊙ q ⁻¹ ⊙ pt)) ).
  specialize (ph (existT (λ x0 : X, f x0 ≈,{ Y }x) t
((q ⊙ p) ⁻¹ ⊙ pt))).
  simpl in ph.
  specialize (ph (setoidrefl X t)).
  assert (  F • (setoidrefl X t)
        (h (existT (λ x0 : X, f x0 ≈,{ Y }x) t (p ⁻¹
⊙ q ⁻¹ ⊙ pt))) ≈  (h (existT (λ x0 : X, f x0 ≈,{ Y
}x) t (p ⁻¹ ⊙ q ⁻¹ ⊙ pt)))).
  apply setoidfamilyrefgeneral.
  swesetoid.
```

```
Defined.

(* convenient lemma for some tactics *)

Theorem  Πsetoid_eq_lemma
  (A: setoid) (F: setoidfamily A)(h h': Πsetoid A F):
   (forall x:A, projT1 h x  ≈  projT1 h' x) ->
    h ≈ h'.
Proof.
  intro H.
  destruct h as [h hext].
  destruct h' as [h' hext'].
  intro x.
  apply H.
Defined.

(* The Pi-object and its projection *)

Definition PiObj {A X Y:setoid}
   (a: A ⇒ X)(f:X  ⇒ Y):=
    Σsetoid Y (Pi_family (fib a) f).

Definition PiObj_proj  {A X Y:setoid}
   (a: A ⇒ X)(f:X  ⇒ Y):
    Σsetoid Y (Pi_family (fib a) f)  ⇒ Y.
  apply (Build_setoidmap
           (Σsetoid Y (Pi_family (fib a) f))
           Y
           (fun z => projT1 z)).
  intros z z' H.
  destruct z as [y r]. destruct z' as [y' r'].
  destruct r. destruct r'.
  simpl in H.
  destruct H.
  simpl.
  assumption.
```

```
Defined.

Theorem PiObj_eq_lemma {A X Y:setoid}
   (a: A ⇒ X)(f:X  ⇒ Y)(w w': PiObj a f)(p:
   projT1 w  ≈ projT1 w'):
  (Pi_family (fib a) f)• p (projT2 w) ≈ projT2 w' ->
   w ≈ w'.
Proof.
   intro H.
   destruct w as [y r]. destruct w' as [y' r'].
   simpl in p.
   exists p.
   apply H.
Defined.

Definition PiObj_slice {A X Y:setoid}
   (a: A ⇒ X)(f:X  ⇒ Y): slice Y.
  apply (mslice (PiObj a f) (PiObj_proj a f)).
Defined.


Definition ev_map_op {A X Y:setoid}(a: A ⇒ X)(f:X  ⇒
Y):  overobj X (pbfunctor_obs X Y f (PiObj_slice a
f))  ->
     A.
  intro z.
  destruct z as [w p].
  assert (fib f (projT1 (snd_setoid w))) as b.
  exists (fst_setoid w).
  apply p.
  assert ((fib a) ★ (fib_proj f (projT1 (snd_setoid
w))) b) as H.
  apply (projT1 (projT2 (snd_setoid w))).
  apply (fib_proj a _ H).
Defined.

Definition ev_map_s {A X Y:setoid}(a: A ⇒ X)(f:X  ⇒
```

```
Y):  (pbfunctor_obs X Y f (PiObj_slice a f))    ⇒
    (mslice A a).
  apply (Build_setoidmap
           (pbfunctor_obs X Y f (PiObj_slice a f))
           (mslice A a)
           (ev_map_op a f)).
  intros z z'.
  destruct z as [w p].
  destruct w as [x h].
  destruct h as [y g].
  destruct g as [g q].
  destruct z' as [w' p'].
  destruct w' as [x' h'].
  destruct h' as [y' g'].
  destruct g' as [g' q'].
  simpl in p.
  simpl in p'.
  intros H.
  simpl.
  destruct H as  [H1 H2].
  destruct H2 as  [e H3].

  specialize (H3 (existT (λ x0 : X, f x0 ≈,{ Y }y')
x' p')).
  simpl in H3.


  assert (∀a0 a' : (fib f) y,
      ∀p : a0 ≈,{ (fib f) y }a',
      ((fib a) ★ (fib_proj f y)) • p (g a0) ≈,{ (fib
a) ★ (fib_proj f y) a' }
      g a') as H4.
  apply q.
  specialize (H4 (existT (λ x : X, f x ≈,{ Y }y) x'
(e ⁻¹ ⊙ p'))).
  specialize (H4 (existT (λ x0 : X, f x0 ≈,{ Y }y) x
```

```
p)).

  simpl in H4.
  assert (x' ≈ x) as H1'.
  apply setoidsym.
  apply H1.
  specialize (H4 H1').
  unfold fib_map_helper in H3.
  simpl in H3.
  swesetoid.
Defined.


Definition ev_map {A X Y:setoid}(a: A ⇒ X)(f:X  ⇒
Y):  slicemap
     (pbfunctor_obs X Y f (PiObj_slice a f))
     (mslice A a).
  apply (Build_slicemap X
         (pbfunctor_obs X Y f (PiObj_slice a f))
         (mslice A a)
         (ev_map_s a f)).
  intro z.
  destruct z as [w p].
  destruct w as [x h].
  destruct h as [y g].
  destruct g as [g q].
  simpl.
  simpl in p.
  simpl in g.
  apply (projT2 (g (existT (λ x0 : X, f x0 ≈,{ Y }y)
x p))).
Defined.

Definition ev {A X Y:setoid}(a: A ⇒ X)(f:X  ⇒ Y):=
   ev_map a f:
   slicemorph
     (pbfunctor_obs X Y f (PiObj_slice a f))
```

```
     (mslice A a).

Theorem PiObj_irr_lemma2
    (A X Y:setoid)(a: A ⇒ X)(f:X  ⇒ Y)
    (u: X)(h:PiObj_slice a f)(q q': f u ≈ projT1 h):
    projT1 (projT1 (projT2 h) (existT _ u q))
      ≈
    projT1 (projT1 (projT2 h) (existT _ u q')).
Proof.
    destruct h as [y P].
    destruct P as [g p].
    simpl.
    simpl in g.
    simpl in q.
    simpl in q'.
    simpl in p.
    specialize (p (existT (λ x : X, f x ≈,{ Y }y) u
q)).
    specialize (p (existT (λ x : X, f x ≈,{ Y }y) u
q')).
    apply p.
    simpl.
    apply setoidrefl.
Defined.


Theorem Pi_family_eq_lemma
  (A X Y : setoid)
  (a : A ⇒ X)
  (f : X ⇒ Y)
  (u : X)
  (y y': PiObj_slice a f)
  (p: projT1 y ≈ projT1 y')
  (q: f u ≈ projT1 y)
  (r: f u ≈ projT1 y') :
(projT1
(projT1 ((Pi_family (fib a) f) • p (projT2 y))
```

```
        (existT (λ x0 : X, f x0 ≈,{ Y }projT1 y') u
     r )))
≈
(projT1
(projT1 (projT2 y)
        (existT (λ x0 : X, f x0 ≈,{ Y }projT1 y) u
q))).

Proof.
  destruct y as [x s]. destruct y' as [x' s'].
  destruct s as [s1 s2]. destruct s' as [s1' s2'].
  simpl.
  simpl in p.
  simpl in q.
  simpl in r.
  unfold fib_map_helper.
  simpl in s2.
  specialize (s2 (existT (λ x0 : X, f x0 ≈,{ Y }x)
           (projT1 (existT (λ x0 : X, f x0 ≈,{ Y }x')
u r))
           (p ⁻¹ ⊙ projT2 (existT (λ x0 : X, f x0 ≈,{
Y }x') u r)))).
  specialize (s2 (existT (λ x0 : X, f x0 ≈,{ Y }x) u
q)).
  apply s2.
  simpl.
  apply setoidrefl.
Defined.

Theorem PiUniversal_map_upper_op_helper
     (A X Y:setoid)(a: A ⇒ X)(f:X  ⇒ Y)
     (B:setoid)(b: B ⇒ Y)
     (h: slicemorph
     (pbfunctor_obs X Y f (mslice B b))
     (mslice A a))
     (z:  (overobj Y (mslice B b))):
     (∀w : ∃x : X, f x ≈,{ Y }b z, ∃x : A,
```

```
          a x ≈,{ X } projT1 w).
Proof.
  intro w.

  exists ((uppermap h) (existT _ ((projT1 w),z)
(projT2 w))).
  apply ((trianglepf h) (existT _ ((projT1 w),z)
(projT2 w))).
Defined.

Theorem PiUniversal_map_upper_op
     (A X Y:setoid)(a: A ⇒ X)(f:X  ⇒ Y)
     (B:setoid)(b: B ⇒ Y)
     (h: slicemorph
     (pbfunctor_obs X Y f (mslice B b))
     (mslice A a)):
     (overobj Y (mslice B b)) ->
      (overobj Y (PiObj_slice a f)).
Proof.
  intro z.
  exists (b z).
  unfold Pi_family.
  simpl.
  exists (PiUniversal_map_upper_op_helper A X Y a f B
b h z).
  intros w w' P.
  destruct w as [x qf].
  destruct w' as [x' qf'].
  simpl in P.
  unfold PiUniversal_map_upper_op_helper.
  simpl.
  apply setoidmapextensionality.
  split.
  assumption.
  swesetoid.
Defined.
```

```
Theorem PiUniversal_map_upper
      (A X Y:setoid)(a: A ⇒ X)(f:X  ⇒ Y)
      (B:setoid)(b: B ⇒ Y)
      (h: slicemorph
      (pbfunctor_obs X Y f (mslice B b))
      (mslice A a)):
      (overobj Y (mslice B b)) ⇒
       (overobj Y (PiObj_slice a f)).
Proof.
  apply (Build_setoidmap
            (overobj Y (mslice B b))
            (overobj Y (PiObj_slice a f))
            (PiUniversal_map_upper_op A X Y a f B b
h)).
  intros x y H.
  simpl.
  assert (b x ≈ b y) as e.
  apply setoidmapextensionality.
  apply H.
  exists e.
  intro z.
  destruct z.
  simpl.
  apply setoidmapextensionality.
  split.
  apply setoidrefl.
  swesetoid.
Defined.

Theorem PiUniversal_map
      (A X Y:setoid)(a: A ⇒ X)(f:X  ⇒ Y)
      (B:setoid)(b: B ⇒ Y)
      (h: slicemorph
      (pbfunctor_obs X Y f (mslice B b))
      (mslice A a)):
      slicemorph (mslice B b) (PiObj_slice a f).
Proof.
```

```
Proof.
  apply (Build_slicemap Y
    (mslice B b) (PiObj_slice a f)
    (PiUniversal_map_upper A X Y a f B b h)).
  intro x.
  simpl.
  apply setoidrefl.
Defined.

(* The universal property of the Pi-object and its
evaluation map. This gives right adjoint to the
pullback functor f^*.

This part can probably be improved greatly
by planning the equational reasoning a little
more prudently ...
*)

Theorem PiUniversality
     (A X Y:setoid)(a: A ⇒ X)(f:X  ⇒ Y):
     ∀ B:setoid,  ∀ b: B ⇒ Y,
     ∀ h: slicemorph
     (pbfunctor_obs X Y f (mslice B b))
     (mslice A a),
     (∃ k: slicemorph (mslice B b) (PiObj_slice a f),
      cmp_slice _ (ev a f)
         (pbfunctor_mor X Y f (mslice B b)
(PiObj_slice a f)  k )   ≈ h)  ∧
     (∀ k k':  slicemorph (mslice B b) (PiObj_slice a
f),
      cmp_slice _ (ev a f)
         (pbfunctor_mor X Y f (mslice B b)
(PiObj_slice a f)  k )   ≈ cmp_slice _ (ev a f)
         (pbfunctor_mor X Y f (mslice B b)
(PiObj_slice a f)  k' )   ->  k ≈ k').
  Proof.
  intros B b h.
```

```
  split.
  exists (PiUniversal_map  A X Y a f B b h).
  intro t.
  destruct t as [xy qf].
  destruct xy as [x y].
  simpl.
  apply setoidmapextensionality.
  split.
  apply setoidrefl.
  apply setoidrefl.

  intros k k'.
  intro H.
  intro x.
  assert (projT1 (uppermap k x) ≈
         projT1 (uppermap k' x)) as p.

  destruct k as [k kt].
  destruct k' as [k' kt'].
  simpl in kt.
  simpl in kt'.
  swesetoid.
  apply (PiObj_eq_lemma a f
        (uppermap k x)
        (uppermap k' x) p).
  apply (Πsetoid_eq_lemma
         ((fib f)  (projT1 ((uppermap k') x)))
         ((fib a) ★ (fib_proj f  (projT1 ((uppermap
k') x))))).


  assert ( ∀z:  (pbfunctor_obs X Y f (mslice B b)),

    uppermap (((cmp_slice (mslice A a)) (ev a f))
        ((pbfunctor_mor X Y f (mslice B b)
         (PiObj_slice a f))
           k)) z
```

```
      ≈
    uppermap (((cmp_slice (mslice A a)) (ev a f))
         ((pbfunctor_mor X Y f (mslice B b)
          (PiObj_slice a f))
           k')) z)
  as H2.
  apply H.
  clear H.

  assert (  ∀z:  (pbfunctor_obs X Y f (mslice B b)),
    (uppermap (ev a f)
      (uppermap
       ((pbfunctor_mor X Y f (mslice B b)
(PiObj_slice a f))
                k)
         z))
      ≈
    (uppermap (ev a f)
       (uppermap
          ((pbfunctor_mor X Y f (mslice B b)
(PiObj_slice a f))
                 k')
           z))) as H3.
  intro z.
  apply cmp_slice_lemma.
  apply setoidsym.
  apply cmp_slice_lemma.
  apply setoidsym.
  apply H2.
  clear H2.
  clear h.

  assert (∀u:X, ∀d:B,
          ∀q:  f u ≈ b d,
       (uppermap (ev a f))
         ((uppermap ((pbfunctor_mor X Y f (mslice B
b) (PiObj_slice a f)) k))
```

```
            (existT _ (u,d) q)) ≈,{ mslice A a }

        (uppermap (ev a f))
          ((uppermap ((pbfunctor_mor X Y f (mslice B
b) (PiObj_slice a f)) k'))
             (existT _ (u,d) q))

) as H4.
  intros u d. intro q.
  apply (H3 (existT _ (u,d) q)).
  clear H3.

  intro w.
  destruct w as [u pf].

  destruct k' as [k' kt'].
  simpl.
  simpl in kt'.
  simpl in pf.
  simpl in p.

  assert (f u  ≈  b x) as q.
  swesetoid.
  specialize (H4 u x q).

  destruct k as [k kt].

  simpl in H4.
  simpl.
  simpl in kt.
  simpl in p.

  assert (
  projT1
       (projT1 (projT2 (k' x))
          (existT (λ x0 : X, f x0 ≈,{ Y }projT1 (k'
x)) u ((kt' x) ⁻¹ ⊙ q))) ≈
   projT1
```

```
     (projT1 (projT2 (k' x))
        (existT (λ x0 : X, f x0 ≈,{ Y }projT1 (k' x))
u pf))) as H5.
  apply PiObj_irr_lemma2.

  assert (

  projT1 (projT1
       ((Pi_family (fib a) f) • p
          (projT2 (k x)))
        (existT (λ x0 : X, f x0 ≈,{ Y }projT1 (k' x))
u pf))
  ≈,{ A }
   projT1
     (projT1 (projT2 (k' x))
        (existT (λ x0 : X, f x0 ≈,{ Y }projT1 (k' x))
u pf))

  ) as H6.
  assert (
    projT1
     (projT1 ((Pi_family (fib a) f) • p (projT2 (k
x)))
        (existT (λ x0 : X, f x0 ≈,{ Y }projT1 (k' x))
u pf)) ≈,{ A }
  projT1  (projT1 (projT2 (k x))
            (existT (λ x0 : X, f x0 ≈,{ Y }projT1 (k
x)) u ((kt x) ⁻¹ ⊙ q)))
  ) as H7.

  apply Pi_family_eq_lemma.
  swesetoid.
  apply H6.
Defined.
```