

(*

A formalization of Aczel's model for CZF
in Coq by Olov Wilander and Erik Palmgren.
Version March 2012.

Other known formalizations of this model
in proof assistants:

N.P. Mendler 1990 in LEGO

M. Takeyama mid 1990s in Agda 1

This file requires UTF8

*)

Require Import PropasTypesUtf8Notation
PropasTypesBasics_mod.

Require Import SwedishSetoids_mod.

Delimit Scope czf_scope with czf.
Open Scope czf_scope.

(* The type of well-founded trees, that is the
universe of CZF sets in the constructed model.*)

Inductive V: Type := sup (A: Set) (f: A → V).

Definition iV (s: V): Set
:= match s with sup A f => A end.
Coercion iV : V >-> Sortclass.

Definition pV (s: V): s → V

```
:= match s with sup A f => f end.  
Coercion pV : V >-> Funclass.
```

```
(* The equality relation is the least bisimulation on  
V *)
```

```
Reserved Notation "x ≐ y" (at level 70, no  
associativity).
```

```
Fixpoint eqV (x y: V): Set :=  
  (∀α: x, ∃β: y, x α ≐ y β) ∧ (∀β: y, ∃α: x, x α ≐ y  
β)  
  where "x ≐ y" := (eqV x y): czf_scope.
```

```
Lemma eqVsplit {x y: V}: (∀α: x, ∃β: y, x α ≐ y β) →  
(∀β: y, ∃α: x, x α ≐ y β) → x ≐ y.
```

```
Proof.
```

```
  destruct x; intros; split; assumption.
```

```
Defined.
```

```
Ltac eqVsplit := apply eqVsplit.
```

```
Ltac eqVassumption_canonical P0 P1 := intros [P0 P1];  
fold eqV in P0, P1; simpl in P0, P1.
```

```
Lemma eqVdestruct {x y: V}: x ≐ y → (∀α: x, ∃β: y, x  
α ≐ y β) ∧ (∀β: y, ∃α: x, x α ≐ y β).
```

```
Proof.
```

```
  destruct x; eqVassumption_canonical H H'; split;  
  assumption.
```

```
Defined.
```

```
Ltac eqVassumption P0 P1 := let P := fresh in  
  intros P; apply eqVdestruct in P; destruct P as [P0  
P1]; simpl in P0, P1.
```

```
(* The first thing to do is to prove that this is an
```

equivalence relation. *)

Theorem refV: $\forall x, x \doteq x$.

Proof.

intro x; induction x.

split; [intro x; exists x; trivial ..].

Qed.

Theorem symV: $\forall x y, x \doteq y \rightarrow y \doteq x$.

Proof.

intro x; induction x as [ix fx IH]. intro y.

eqVassumption P0 P1; eqVsplit;

match goal with [hyp: $\forall _ : ?a, \exists _ : ?b, _ \vdash \forall _ : ?a, \exists _ : ?b, _$] =>

let x := fresh in let y := fresh in

intro x; destruct hyp with x as [y]; exists y;

auto

end.

Qed.

Theorem traV: $\forall x y z, x \doteq y \rightarrow y \doteq z \rightarrow x \doteq z$.

Proof.

intros x; induction x as [ix fx IH]. intros y z.

eqVassumption P0 P1; eqVassumption Q0 Q1; eqVsplit;

match goal with [hyp0: $\forall _ : ?a, \exists b0 : ?b, _, hyp1: \forall b1 : ?b, \exists c0 : ?c, _ \vdash \forall a1 : ?a, \exists c1 : ?c, _$] =>

let a := fresh in let b := fresh in let c := fresh in

intro a; destruct hyp0 with a as [b]; destruct hyp1 with b as [c]; exists c; eauto

end.

Qed.

Hint Resolve refV : czf.

Hint Immediate symV : czf.

Hint Resolve traV : czf. (* The warning here is expected *)

```
Ltac czf := simpl; eauto with czf.
```

```
(* Next, define the membership relation... *)
```

```
Definition memV (x y: V): Set
```

```
:= ∃idx: y, x ≐ y idx.
```

```
Infix "∈" := memV (at level 70, no associativity) :  
czf_scope.
```

```
Notation "x ∉ y" := (¬(x ∈ y)) (at level 70) :  
czf_scope.
```

```
Lemma membership (x y: V) (a: y): x ≐ y a → x ∈ y.
```

```
Proof.
```

```
intro; exists a; auto.
```

```
Defined.
```

```
Hint Resolve membership : czf.
```

```
(* ... and prove that it respects the equality  
relation introduced earlier. *)
```

```
Lemma ext1: ∀x y z, x ≐ y → y ∈ z → x ∈ z.
```

```
Proof.
```

```
intros x y z H [idx eq]. czf.
```

```
Qed.
```

```
Lemma ext2: ∀x y z, x ∈ y → y ≐ z → x ∈ z.
```

```
Proof.
```

```
intros x y z [idx eq]. eqVassumption P0 P1; clear  
P1.
```

```
destruct P0 with idx as [idx' eq']. czf.
```

```
Qed.
```

```
(* The warnings here are expected *)
```

```
Hint Resolve ext1: czf.
```

```
Hint Resolve ext2: czf.
```

(* Introduce some more notation *)

Notation " $x \subseteq y$ " := ($\forall z: V, z \in x \rightarrow z \in y$) (at level 60) : czf_scope.

(* Now start proving that the axioms hold *)

Lemma ext3: $\forall x y, x \subseteq y \rightarrow y \subseteq x \rightarrow x \doteq y$.

Proof.

```
  intros x y xsuby ysubx. eqVsplit;
  match goal with [hyp: ?a  $\subseteq$  ?b |-  $\forall \_ : iV ?a, \exists \_ : iV ?b, \_$ ] =>
    let a := fresh in let  $\beta$  := fresh in
      intro a; destruct hyp with (a a) as [ $\beta$ ]; czf
    end.
```

Qed.

Hint Resolve ext3: czf.

Theorem Extensionality:

$\forall x, \forall y, (\forall z, z \in x \leftrightarrow z \in y) \rightarrow x \doteq y$.

Proof.

```
  intros x y hyp. apply ext3; intro z; apply (hyp z).
  Qed.
```

Definition pairV (x y: V): V

:= sup bool ($\lambda b, \text{match } b \text{ with true } \Rightarrow x \mid \text{false } \Rightarrow y$ end).

Notation " $\{\{ x, y \}\}$ " := (pairV x y)

(at level 0, x, y at level 69) : czf_scope.

Lemma pairVL1: $\forall x y, x \in \{\{ x, y \}\}$.

Proof.

```
  intros; exists true; czf.
  Qed.
```

Lemma pairVL2: $\forall x y, y \in \{\{x, y\}\}$.

Proof.

intros; exists false; czf.

Qed.

Hint Resolve pairVL1 pairVL2: czf.

Lemma pairingchar: $\forall a b y, y \in \{\{a, b\}\} \leftrightarrow y \doteq a \vee y \doteq b$.

Proof.

intros a b y; split.

intros [i e]. destruct i; auto.

intros [eq | eq]; czf.

Qed.

Hint Resolve pairingchar : czf.

Theorem Pairing:

$\forall a b, \exists c, \forall y, y \in c \leftrightarrow y \doteq a \vee y \doteq b$.

Proof.

czf.

(* intros; setexists $\{\{a, b\}\}$; apply pairingchar.
*)

Qed.

Lemma pairingcharR: $\forall a b y, y \in \{\{a, b\}\} \rightarrow y \doteq a \vee y \doteq b$.

Proof.

intros a b y; eapply pairingchar.

Qed.

Lemma pairingcharL: $\forall a b y, y \doteq a \vee y \doteq b \rightarrow y \in \{\{a, b\}\}$.

Proof.

intros a b y; eapply pairingchar.

Qed.

Hint Resolve pairingcharR pairingcharL: czf.

Definition unionV (x: V): V
:= sup (∃a: x, x a) (λ u, x (projT1 u) (projT2 u)).
Notation "∪ X" := (unionV X) (at level 1) :
czf_scope.

Lemma unionLm: ∀s: V, ∀x: s, s x ∈ s.

Proof.

czf.

(* intros s x; exists x; apply refV. *)

Qed.

Lemma unionLm1:

∀s: V, ∀A: Set, ∀f: A → V, ∀x: A, s ∈ f x → s ∈
∪(sup A f).

Proof.

intros s A f x [i eq].

exists (existT _ x i); assumption.

Qed.

Hint Resolve unionLm1 : czf.

Lemma unionLm2:

∀r s t, r ∈ s → s ∈ t → r ∈ ∪t.

Proof.

intros r s t rins [idx eq]. czf.

(* intros r s [it ft l a'] rs [idx eq].

apply unionLm1 with idx, ext2 with s;

[assumption..].

destruct s as [i f l a]; contradiction. *)

Qed.

Hint Resolve unionLm2 : czf.

Lemma unionchar:

∀a y, y ∈ ∪a ↔ ∃x, x ∈ a ∧ y ∈ x.

Proof.

```
intros a y; split. intros [[idx1 idx2] eq].
exists (a idx1). split. czf. apply membership with
idx2; auto.
intros [x h]. apply unionLm2 with x; tauto.
Qed.
```

Hint Resolve unionchar : czf.

Theorem Union: $\forall a, \exists b, \forall y, y \in b \leftrightarrow \exists x, x \in a \wedge y \in x$.

Proof.

```
czf.
```

Qed.

Notation "s u t" := ($\cup\{\{s, t\}\}$) (at level 65, right associativity) : czf_scope.

Theorem binunionchar:

$$\forall x \ s \ t, x \in s \cup t \leftrightarrow x \in s \vee x \in t.$$

Proof.

```
intros x s t; split.
intro xinbinu. apply unionchar in xinbinu; destruct
xinbinu as [y [p q]].
apply pairingcharR in p. destruct p; czf.
intros [mem l mem]; czf.
(* intros [[i i'] eq].
destruct i; [left | right]; [exists i'; assumption
..].
intros [[i eq] | [i eq]].
exists (existT _ true i); assumption.
exists (existT _ false i); assumption. *)
Qed.
```

Hint Resolve binunionchar : czf.

Lemma binunioncharL: $\forall x \ s \ t, x \in s \cup t \rightarrow x \in s \vee x \in t$.

Proof.

```
intros x s t [[idx1 idx2] e]. induction idx1; simpl
in *; czf.
```

Qed.

Lemma binunioncharR: $\forall x \ s \ t, x \in s \vee x \in t \rightarrow x \in s \cup t$.

Proof.

```
intros x s t [mem | mem]; czf.
```

Qed.

Hint Resolve binunioncharL binunioncharR : czf.

Definition emptyV : V

:= sup False (False_rect V).

Notation " $\emptyset$ " := emptyV : czf_scope.

Theorem EmptySet:

$\forall x, x \notin \emptyset$.

Proof.

```
intros _ [].
```

Qed.

Hint Resolve EmptySet: czf.

Theorem EmptySetAxiom:

$\exists x, \forall y, y \notin x$.

Proof.

czf.

(* setexists $\emptyset$; apply EmptySet. *)

Qed.

Definition extensionalV (P: V $\rightarrow$ Type): Type

:= $\forall x \ y, P \ x \rightarrow x = y \rightarrow P \ y$.

Notation " $\ll P \gg$ " := (extensionalV P) : czf_scope.

Theorem SetInduction (P: V $\rightarrow$ Type) (Pext: $\ll P \gg$):

$(\forall x, (\forall y, y \in x \rightarrow P\ y) \rightarrow P\ x) \rightarrow \forall x, P\ x.$

Proof.

intro IH. induction x as [I f IH']; apply IH.

intros y [i eq]. czf.

Qed.

Definition singletonV (x: V) := sup $\top$ ($\lambda _, x$).

Notation "{{ s }}" := (singletonV s) (at level 0, s
at level 69) : czf_scope.

(* Sanity check lemma for this notation *)

Lemma singleton_equals_pair (x: V): {{ x }} $\doteq$ {{ x, x }}.

Proof.

simpl; split.

exists true; apply refV.

induction β ; repeat progress split; apply refV.

Qed.

Lemma singleton_char (x y: V): $x \in \{ \{ y \} \} \rightarrow x \doteq y.$

Proof.

firstorder.

Qed.

Hint Resolve singleton_char : czf.

Notation "s +" := (s u {{ s }}) (at level 1) :
czf_scope.

Lemma succL1:

$\forall s\ t:V, s \in t^+ \rightarrow s \doteq t \vee s \in t.$

Proof.

intros s t H. apply binunioncharL in H. destruct H;
czf.

Qed.

Lemma succL2:

$\forall s : V, s \doteq t \vee s \in t \rightarrow s \in t^+.$

Proof.

intros s t [eq | mem].

exists (existT _ false tt). trivial. czf.

Qed.

Hint Resolve succL1 succL2.

Fixpoint numeralV (n: $\mathbb{N}$): V

:= match n with

| 0 => $\emptyset$

| S m => (numeralV m)⁺

end.

Notation " $\ulcorner$ n $\urcorner$ " := (numeralV n) : czf_scope.

Definition ω : V

:= sup $\mathbb{N}$ numeralV.

(* Notation "' ω '" := (natV) : czf_scope. *)

Notation ttve x := ($\forall y z, z \in y \rightarrow y \in x \rightarrow z \in x$).

Theorem numeralsaretransitive:

$\forall n : \mathbb{N}, \text{ttve}(\ulcorner n \urcorner).$

Proof.

induction n.

intros y z _ mem; contradiction EmptySet with y.

intros y z mem1 mem2. apply succL1 in mem2;

destruct mem2; czf.

Qed.

Lemma eltnat:

$\forall n : \mathbb{N}, \forall x, x \in \ulcorner n \urcorner \rightarrow \exists m : \mathbb{N}, x \doteq \ulcorner m \urcorner.$

Proof.

induction n.

intros _ [].

intros x mem. apply succL1 in mem. firstorder.

Qed.

Lemma **omegattve**: ttve(ω).

Proof.

intros z y yinz (n, eq).

apply (eltnat n), ext2 with z; assumption.

Qed.

Lemma **succext**: $\forall x\ y, x \doteq y \rightarrow x^+ \doteq y^+$.

Proof.

intros x y eq. apply ext3;

intros z mem; apply succL1 in mem; destruct mem;

czf.

Qed.

Theorem **Infinity**:

$\exists X, \emptyset \in X \wedge \forall y, y \in X \rightarrow y^+ \in X$.

Proof.

exists ω . split. exists $\emptyset$. czf.

intros y [i e]. exists (S i). eapply traV. apply succext, e. czf.

Qed.

Definition **sepV** (s: V) (P: V $\rightarrow$ Set)

:= sup ($\exists a: s, P\ (s\ a)$) ($\lambda u, s\ (projT1\ u)$).

Notation "**{{** x $\in$ s | P **}}**" := (sepV s (fun x $\Rightarrow$ P))
(at level 0, x, s, P at level 69) : czf_scope.

Lemma **separationchar** (P: V $\rightarrow$ Set) (Pext: «P»):

$\forall s\ x, x \in \{ \{ y \in s \mid P\ y \} \} \leftrightarrow x \in s \wedge P\ x$.

Proof.

intros s x; split.

intros [[i e] p]; split. exists i; assumption. czf.

intros ((i, eq), pf).

apply (Pext x (s i)) in pf; [| apply eq].

exists (existT (fun u $\Rightarrow$ P (s u)) i pf);

assumption.

Qed.

Theorem Separation ($P: V \rightarrow \text{Set}$) ($P_{\text{ext}}: \llbracket P \rrbracket$):

$\forall s, \exists t, \forall x, x \in t \leftrightarrow x \in s \wedge P x.$

Proof.

intros; exists ({x ∈ s | P x}).

apply separationchar; assumption.

Qed.

Lemma sep_sub ($P: V \rightarrow \text{Set}$) ($P_{\text{ext}}: \llbracket P \rrbracket$): $\forall x, \{y \in x \mid P y\} \subseteq x.$

Proof.

intros x z [[idx pf] eq]. exists idx; assumption.

Qed.

Notation " $\forall x \in s, P$ " := ($\forall x: iV s, (\text{fun } x: V \Rightarrow P)$
($pV s x$)

($\text{at level } 200, x \text{ at level } 0$).

Notation " $\exists x \in s, P$ " := ($\exists x: iV s, (\text{fun } x: V \Rightarrow P)$
($pV s x$)

($\text{at level } 200, x \text{ at level } 0$).

Lemma Ballright:

$\forall a, \forall P: V \rightarrow \text{Set}, \llbracket P \rrbracket \rightarrow ((\forall x \in a, P x) \leftrightarrow \forall x, x \in a \rightarrow P x).$

Proof.

intros a P E; split.

intros bdd x (i, eq).

apply E with (a i), symV, eq; apply bdd.

intros full i; apply full, unionLm.

Qed.

Lemma Bexright:

$\forall a, \forall P: V \rightarrow \text{Set}, \llbracket P \rrbracket \rightarrow ((\exists x \in a, P x) \leftrightarrow \exists x: V, x \in a \wedge P x).$

Proof.

intros a P E. split

```

intros a f E, split.
intros (i, pf);
  exists (a i); split; [apply unionLm |
assumption].
intros (x, ((i, eq), pf)).
exists i.
apply E with x; [assumption..].
Qed.

```

Lemma SetInductionBdd ($P: V \rightarrow \text{Set}$) ($P_{\text{ext}}: \llbracket P \rrbracket$):
 $(\forall x, (\forall y \in x, P y) \rightarrow P x) \rightarrow \forall x, P x.$

Proof.

```

intros IH x; induction x; czf.
Qed.

```

Notation totalV a b R := $(\forall x \in a, \exists y \in b, R x y).$

Notation bitotalV a b R
:= $((\text{totalV } a \ b \ R) \wedge (\text{totalV } b \ a \ (\lambda x \ y, R y x))).$

Theorem StrongCollection:

$\forall R: V \rightarrow V \rightarrow \text{Type},$
 $\forall a, (\forall x \in a, \exists y, R x y) \rightarrow \exists b, \text{bitotalV } a \ b \ R.$

Proof.

```

intros r a bdd. destruct (TTAC a V _ bdd) as [f p].
exists (sup a f). czf.
Qed.

```

Definition sscV (a b: V): V

:= $(\text{sup } (a \rightarrow b) (\lambda f, \text{sup } a (\lambda x, b (f x)))).$

Theorem SubsetCollection:

$\forall Q: V \rightarrow V \rightarrow V \rightarrow \text{Type},$
 $\forall a \ b, \exists c, \forall u,$
 $((\forall x \in a, \exists y \in b, Q x y u) \rightarrow$
 $\exists d \in c, ((\forall x \in a, \exists y \in d, Q x y u) \wedge (\forall y \in d, \exists x$
 $\in a, Q x y u))).$

Proof.

```

intros Q a b. exists (sscV a b). intros u h.

```

```
destruct (TTAC a b _ h). czf.
Qed.
```

(* Now some more purely set theoretical work *)

```
Fixpoint TC (x: V): V :=
  match x with sup A f => x ∪ ∪(sup A (λ i, TC (f
i))) end.
```

```
Lemma TClemma: ∀x y, x ∈ y → x ∈ TC y.
```

```
Proof.
```

```
  intros x [I f] [i e]. simpl in *.
  unfold unionV; apply (sigrejig bool). exists true;
exists i; assumption.
Qed.
```

```
Lemma TC_ttve (x: V): ttve(TC x).
```

```
Proof.
```

```
  induction x as [A f IH]; intros x y mem mem2.
  apply binunioncharL in mem2; destruct mem2 as [[i
e] | [[i1 i2] e2]].
  apply binunioncharR; right. unfold unionV; apply
(sigrejig A).
  exists i. cut (y ∈ TC (f i)); trivial. apply
TClemma. apply ext2 with x; assumption.
  apply binunioncharR; right. unfold unionV; apply
(sigrejig A).
  exists i1. apply IH with x. assumption. exists
i2; assumption.
Qed.
```

(* Now start developing basic mathematics inside the model *)

```
Notation " < x , y > " := ({{ {{ x }}, {{ x, y }}
}}).
```

Lemma pairing_injective (x y z w: V): $\langle x, z \rangle \doteq \langle y, w \rangle \rightarrow x \doteq y \wedge z \doteq w$.

Proof.

```

intros [eq1 eq2]; fold eqV in *. simpl in *.
Local Ltac boolcases :=
  repeat match goal with [hyp:  $\forall \_ : \text{bool}, \_ \vdash \_$ ] =>
    destruct (hyp true); destruct (hyp
false); clear hyp
    end.
Local Ltac boolbranch := repeat match goal with
[hyp:  $\text{bool} \vdash \_$ ] => induction hyp end.
Local Ltac dedup :=
  repeat match goal with [hyp:  $\top \vdash \_$ ] => destruct
hyp end;
  repeat match goal with [hyp0:  $?a \doteq ?b$ , hyp1:  $?a \doteq ?b \vdash \_$ ] => clear hyp1 end;
  repeat match goal with [hyp0:  $?a \doteq ?b$ , hyp1:  $?b \doteq ?a \vdash \_$ ] => clear hyp1 end.
Local Ltac desingleton :=
  repeat match goal with [hyp:  $\{\{ \_ \} \} \doteq \{\{ \_ \} \} \vdash \_$ ] =>
    let P := fresh in
      destruct hyp as [P _]; fold eqV in P;
specialize P with tt; simpl in P; destruct P
    end.
Local Ltac singlepaireq :=
  repeat match goal with [hyp:  $\{\{ \_, \_ \} \} \doteq \{\{ \_ \} \} \vdash \_$ ] => apply symV in hyp end;
  repeat match goal with [hyp:  $\{\{ \_ \} \} \doteq \{\{ \_, \_ \} \} \vdash \_$ ] =>
    let eq1 := fresh in destruct hyp as [_
eq1]; fold eqV in eq1; simpl in eq1; boolcases
    end.
Local Ltac pairpaireq :=
  repeat match goal with [hyp:  $\{\{ \_, \_ \} \} \doteq \{\{ \_, \_ \} \} \vdash \_$ ] =>
    let e1 := fresh in let e2 := fresh in

```



```

      destruct hyp as [e1 e2]; simpl in e1,
e2; fold eqV in e1, e2; boolcases; boolbranch
end.

```

```

Local Ltac finish := eauto using symV, traV.
Time boolcases; boolbranch; dedup; desingleton;
singlepaireq; pairpaireq; dedup; finish.
Defined.

```

Lemma pairing_extensional ($x\ y\ z\ w : V$): $x \doteq z \rightarrow y \doteq w \rightarrow \langle x, y \rangle \doteq \langle z, w \rangle$.

Proof.

```

  intros xeqz yeqw.
  split; intro a; exists a.
  destruct a. repeat (split; simpl); assumption.
  split; intro a; exists a; destruct a; assumption.
  destruct a. repeat (split; simpl); assumption.
  split; intro a; exists a; destruct a; assumption.
Defined.

```

Definition prodV ($x\ y : V$): V
 $:= \text{sup } (x \wedge y) \text{ (fun } p \Rightarrow \text{let } (a, \beta) := p \text{ in } \langle x\ a, y\ \beta \rangle)$.

Notation " $x \times y$ " $:= (\text{prodV } x\ y) \text{ (at level 40)}$.

Lemma productchar ($x\ y : V$): $\forall z, z \in x \times y \leftrightarrow \exists a \in x, \exists \beta \in y, z \doteq \langle a, \beta \rangle$.

Proof.

```

  intro z; split.
  intros [[a β] eq]; simpl in eq. exists a. exists β.
assumption.
  intros [a [β eq]]; exists (a, β); assumption.
Qed.

```

Lemma productchar_altproof ($x\ y : V$): $\forall z, z \in x \times y \leftrightarrow \exists a \in x, \exists \beta \in y, z \doteq \langle a, \beta \rangle$.
 proof.

```

let z:V.
focus on (z ∈ x × y → ∃a ∈ x, ∃β ∈ y, z ≐ ⟨a, β⟩ ).
  given index such that eq: (z ≐ (x × y) index).
  claim (z ≐ (x × y) index → ∃a ∈ x, ∃β ∈ y, z ≐ ⟨a,
β⟩ ).
    consider ix: x, iy: y from index.
    assume (z ≐ (x × y) (ix, iy)). take ix. take iy.
hence thesis.
  end claim.
  hence thesis by eq.
end focus.
  given a, β such that (z ≐ ⟨x a, y β⟩ ). take (a,
β). hence thesis.
end proof.
Qed.

```

Theorem Products: $\forall x \ y, \exists z, \forall w, w \in z \leftrightarrow \exists a \in x, \exists \beta \in y, w \doteq \langle a, \beta \rangle$.

Proof.

```

intros x y. exists (x × y). apply productchar.
Qed.

```

Definition is_rel (x y: V) (R: V): Type
:= R ⊆ x × y.

Definition is_total (x y: V) (R: V): Type
:= $\forall a \in x, \exists \beta \in y, \langle a, \beta \rangle \in R$.

Definition is_functional (R: V): Type
:= $\forall x \ y \ y', \langle x, y \rangle \in R \wedge \langle x, y' \rangle \in R \rightarrow y \doteq y'$.

Definition is_function (x y: V) (F: V): Type
:= is_rel x y F ∧ is_total x y F ∧ is_functional F.

Definition identity (x: V): V
:= $\{\{ p \in x \times x \mid \exists y \in x, p \doteq \langle y, y \rangle \}\}$.

.....

```

Lemma id_is_function {x: V}: is_function x x
(identity x).
Proof.
  split. intros a [[ai11 ai12] ai2] eq]. exists
(ai11, ai12). assumption.
  split. intro a. exists a. unfold identity. unfold
sepV.
  apply (sigrejig (x  $\wedge$  x)). exists (a, a). apply
(sigrejig x). exists a.
  exists (refV  $\langle$ x a, x a $\rangle$ ). simpl.
  split; intro y; exists y; auto using refV.
  intros y z z' [[yzi yzeq] [yz'i yz'eq]].
  repeat match goal with [hyp: ( $\langle$ y, ?a $\rangle \doteq$  (?b ?c))
|- _] =>
    let i := fresh in let j := fresh in let r :=
fresh in let s := fresh in let t := fresh in
    destruct c as [[i j] r];
    change ((x  $\times$  x) (i, j)) with ( $\langle$ x i, x j $\rangle$ ) in
r;
    destruct r as [s t];
    change ((identity x) (existT _ (i, j) _))
with ( $\langle$ x i, x j $\rangle$ ) in hyp
    end.
  repeat match goal with [hyp: ( $\_$ ,  $\_$ )  $\doteq$  ( $\_$ ,  $\_$ ) |-
_] =>
    apply pairing_injective in hyp; destruct
hyp
    end.
  apply traV with y;
  eauto using symV, traV.
Qed.

```

```

Definition functionspace (x y:V): V
:= sup ( $\exists f:x \rightarrow y$ ,  $\forall \alpha \beta:x$ ,  $x \alpha \doteq x \beta \rightarrow y (f \alpha) \doteq y (f \beta)$ )
    (fun p => match p with existT f _ =>
      sup x (fun  $\alpha$  =>  $\langle$ x  $\alpha$ , y (f  $\alpha$ ) $\rangle$ ) end).

```

(* Triple arrow UTF-8 notation for function spaces in V *)

Notation " $A \Rightarrow B$ " := (functionspace A B) (at level 55, right associativity).

Lemma function_space_char (x y: V)

: $\forall z, z \in x \Rightarrow y \leftrightarrow \text{is_function } x \ y \ z$.

Proof.

intros z. split.

intros [[f pf] eq]. simpl in eq. split.

intros w [i eq2]. eapply ext1 with (z i).

assumption.

apply eqVdestruct in eq; simpl in eq. destruct eq as [eq1 _]. destruct eq1 with i.

exists (x0, f x0). assumption.

split.

intros a. exists (f a). apply eqVdestruct in eq; simpl in eq; destruct eq as [_ eq].

destruct eq with a as [x0 eq']. exists x0. finish.

intros a β β' [[iy ey] [iy' ey']]. apply eqVdestruct in eq; simpl in eq; destruct eq as [eq1 _].

destruct eq1 with iy as [y eqy]. destruct eq1 with iy' as [y' eqy']. clear eq1.

assert (claim1: $\langle a, \beta \rangle \doteq \langle x \ y, y \ (f \ y) \rangle$); finish.

assert (claim2: $\langle a, \beta' \rangle \doteq \langle x \ y', y \ (f \ y') \rangle$);

finish.

clear ey ey' eqy eqy'.

apply pairing_injective in claim1; apply pairing_injective in claim2.

destruct claim1, claim2.

assert (y (f y) = y (f y')). apply pf. finish.

finish.

intros [relz [totz funz]]. unfold is_total in totz;

```

apply !IAC in totZ. destruct totZ as [r p].
  unfold functionspace; apply (sigrejig (x → y)).
exists f. simpl. split.
  intros a β e; apply funz with (x a). split. apply
p. eapply ext1.
  eapply pairing_extensional. apply e. apply refV.
apply p.
  apply eqVsplit. unfold is_rel in relz. intro a.
simpl. destruct relz with (z a) as [[β γ] e]. czf.
  exists β. specialize p with β. apply traV with (⟨x
β, γ γ⟩). apply e. apply pairing_extensional. czf.
  unfold is_functional in funz. apply funz with (x
β). split. exists a. finish. assumption.
  simpl. intro β. destruct p with β as [i e]. exists
i; finish.
Defined.

```

Theorem FunctionSpace: $\forall x \ y, \exists z, \forall w, w \in z \leftrightarrow$
 $\text{is_function } x \ y \ w.$

Proof.

```

intros x y. exists (x ⇒ y). apply
functionspacechar.
Qed.

```

Definition π_1 (x y:V): V
 $:= \sup (x \times y) (\lambda p, \text{let } (a, \beta) := p \text{ in } \langle \langle x \ a, y$
 $\beta \rangle, x \ a \rangle).$

Lemma $\pi_1\text{fun}$ (x y: V): $\text{is_function } (x \times y) \ x \ (\pi_1 \ x \ y).$
Proof.

```

repeat split.
  intros z [[a β] eq]. simpl in eq.
  exists ((a, β), a). assumption.
  intros [a β]. exists a. exists (a, β). apply refV.
  intros a β β'. intros [[idx11 idx12] eq1] [[idx21
idx22] eq2]].
  repeat match goal with [hyp: _ ≐ pV (π1 x y) (?a, ?
..

```

```

b) | - _ | =>
      change (( $\pi_1$  x y) (a, b)) with < <x a, y
b> , x a> in hyp
      end.
  repeat match goal with [hyp: <_, _>  $\doteq$  <_, _> | -
_] =>
      apply pairing_injective in hyp; destruct
hyp
      end.
  eapply traV in e1; [| apply symV; apply e].
  apply pairing_injective in e1; destruct e1 as [eq
_].
  eauto using traV, symV.
Qed.

```

Definition `compose_relV` ($x\ y\ z: V$) ($R\ S: V$): V
 $:= \{ \{ p \in x \times z \mid \exists \alpha \in x, \exists \beta \in y, \exists \gamma \in z, \langle \alpha, \beta \rangle \in R \wedge \langle \beta, \gamma \rangle \in S \wedge \langle \alpha, \gamma \rangle \doteq p \} \}$.

Lemma `composed_rel_is_relV` { $x\ y\ z: V$ } ($R\ S: V$)
 $: \text{is_rel } x\ y\ R \rightarrow \text{is_rel } y\ z\ S \rightarrow \text{is_rel } x\ z$
 $(\text{compose_relV } x\ y\ z\ R\ S)$.

Proof.

```

  intros _ _. unfold is_rel. auto. apply sep_sub.
  intros w w' [a [b [c [aRb [bSy eq]]]]] weqw'.
  exists a. exists b. exists c. split. assumption.
split. assumption. finish.
Qed.

```

(* Define a functor from V to the (E-)category of setoids *)

Definition `obfn`: $V \rightarrow \text{setoid}$.

```

  intro x.
  apply (Build_setoid x ( $\lambda a\ b, x\ a \doteq x\ b$ )); eauto
using refV, symV, traV.
Defined

```

Defined.

Definition arfn (x y: V): (x $\Rightarrow$ y) $\rightarrow$ (obfn x) $\Rightarrow$ (obfn y).

intros [f fext].

apply (Build_setoidmap (obfn x) (obfn y) f). simpl.

assumption.

Defined.

(* Must now verify functoriality *)

Definition id_index (x: V): x $\Rightarrow$ x.

exists (λ a, a). auto.

Defined.

Lemma id_index_identity (x: V): (x $\Rightarrow$ x) (id_index x) $\doteq$ identity x.

Proof.

split. intro a. unfold identity. unfold sepV. apply (sigrejig (x $\times$ x)). exists (a, a). apply (sigrejig x). exists a.

exists (refV _). apply refV.

intros [[a β] [γ eq]]. exists γ . unfold projT1.

apply symV, eq.

Qed.

Lemma functoriality_id (x: V): arfn x x (id_index x) $\approx$ idmap.

Proof.

intros a. simpl. apply refV.

Qed.

Definition compose_index (x y z: V): y $\Rightarrow$ z $\rightarrow$ x $\Rightarrow$ y $\rightarrow$ x $\Rightarrow$ z.

intros [f fext] [g gext].

exists (λ a, f (g a)). intros; apply fext; apply

gext; **assumption**.

Defined.

Lemma functoriality_composition (x y z: V) (F: y $\Rightarrow$ z)
(G: x $\Rightarrow$ y)
: arfn x z (compose_index x y z F G) $\approx$ (arfn y z F)
◦ (arfn x y G).

Proof.

destruct F **as** [f fext]; destruct G **as** [g gext].
simpl. intro; apply refV.

Qed.

(* Finally, proofs that this functor is full and faithful *)

Lemma faithful (x y: V) (F G: x $\Rightarrow$ y): arfn x y F $\approx$
arfn x y G $\rightarrow$ (x $\Rightarrow$ y) F $\doteq$ (x $\Rightarrow$ y) G.

Proof.

destruct F **as** [f fext]; destruct G **as** [g gext].
unfold arfn. intro feqq. simpl **in** feqq.
split. intro a. exists a. eapply traV. apply
pairing_extensional.
apply refV. apply feqq. apply refV.
intro β . exists β . eapply traV. apply
pairing_extensional. apply refV. apply feqq. apply
refV.

Qed.

Lemma full (x y: V): $\forall f$: setoidmap (obfn x) (obfn y),
 $\exists G$: x $\Rightarrow$ y, arfn x y G $\approx$ f.

Proof.

intros [f fext].
unfold functionspace. apply (sigrejig (x $\rightarrow$ y)).
exists f. exists fext. simpl. intro; apply refV.

Qed.

(* The functor maps any singleton to a terminal setoid. This together with fact that both categories are generated by the terminal object, and the full faithfulness of the functors, yield that if $\text{obfn } x$ is projective in setoids, then so is x in V . In other words if $\text{obfn } x$ admits axiom of choice, then so does x . *)

Definition `canon1_map` ($x:V$) : ($\text{obfn } \{\{ x \}\}$) $\Rightarrow$ `unit_setoid`.
`apply` (`Build_setoidmap` ($\text{obfn } \{\{ x \}\}$) `unit_setoid` ($\lambda a, tt$)).
`trivial`.
 Defined.

Definition `canon2_map` ($x:V$) : `unit_setoid` $\Rightarrow$ ($\text{obfn } \{\{ x \}\}$).
`apply` (`Build_setoidmap` `unit_setoid` ($\text{obfn } \{\{ x \}\}$) ($\lambda a, tt$)).
`intros`.
`apply` `setoidrefl`.
 Defined.

Lemma `singleton_to_terminal` ($x:V$):
 $\exists F:(\text{obfn } \{\{ x \}\}) \Rightarrow \text{unit_setoid}$,
 $\exists G:\text{unit_setoid} \Rightarrow (\text{obfn } \{\{ x \}\})$,
 $F \circ G \approx \text{idmap} \wedge G \circ F \approx \text{idmap}$.
 Proof.

`exists` (`canon1_map` x).
`exists` (`canon2_map` x).
`split`.
`intro`.
`swesetoid`.
`intro`.
`swesetoid`.

```

-----
    apply refV.
Qed.

```

(* A series of proofs that certain set-theoretic expressions are extensional in one or another variable *)

```

Theorem expr_extensional_1 (z a :V):
«(fun u => z ≐ ⟨ a, u ⟩ )».

```

```

Proof.
  unfold extV.
  intros x y P Q.
  assert (⟨ a, x ⟩ ≐ ⟨ a, y ⟩ ) as H.
  apply pairing_extensional.
  apply refV.
  apply Q.
  apply (traV _ _ _ P H).

```

Defined.

```

Theorem expr_extensional_2 (z a v :V):
«(fun v =>  ∃ b ∈ v, z ≐ ⟨a, b⟩ )».

```

```

Proof.
  unfold extV.
  intros x y P Q.
  assert (∃b: V,  b ∈ x ∧ z ≐ ⟨a, b⟩ ) as H.
  apply Bexright.
  apply expr_extensional_1.
  apply P.
  assert (∃b: V,  b ∈ y ∧ z ≐ ⟨a, b⟩ ) as H1.
  destruct H as [b R].
  exists b.
  split.
  destruct R as [R1 R2].
  apply (ext2 _ x _).
  apply R1.
  apply Q.
  _

```

```

    apply R.
    apply ((snd (Bexright y (fun u => z ≐ ⟨ a, u ⟩ )
      (expr_extensional_1 z a)))) H1).
Defined.

```

Theorem `expr_extensional_3` (`z v :V`):
`«(fun a => ∃b ∈ v, z ≐ ⟨a, b⟩ )».`

Proof.

```

  unfold extensionalV.
  intros x y.
  intros P Q.
  destruct P as [b R].
  exists b.
  assert ( ⟨ x, v b ⟩ ≐ ⟨ y, v b ⟩ ) as H.
  apply pairing_extensional.
  apply Q.
  apply refV.
  apply (traV _ _ _ R H).
Defined.

```

(Alternative characterization of
 cartesian product *)*

Lemma `productchar_2_lemma1` (`x y: V`):
 $\forall z, (\exists a \in x, \exists \beta \in y, z \doteq \langle a, \beta \rangle)$
 $\leftrightarrow$
 $(\exists a: V, a \in x \wedge \exists \beta \in y, z \doteq \langle a, \beta \rangle)$.

Proof.

```

  intro z.
  apply Bexright.
  apply (expr_extensional_3 z y).
Defined.

```

Lemma `productchar_2_lemma2` (`x y: V`):
 $\forall z, (\exists a \in x, \exists \beta \in y, z \doteq \langle a, \beta \rangle)$
 $\leftrightarrow (\exists a: V, a \in x \wedge$

$(\exists \beta: V, \beta \in y \wedge z \doteq \langle \alpha, \beta \rangle)).$

Proof.

```

intro z.
split.
intro H.
assert  $(\exists \alpha: V, \alpha \in x \wedge \exists \beta \in y, z \doteq \langle \alpha, \beta \rangle )$  as
H1.
apply (productchar_2_lemma1 x y).
apply H.
destruct H1 as [a R].
exists a.
split.
apply R.
apply Bexright.
apply expr_extensional_1.
apply R.
intro H.
apply (productchar_2_lemma1 x y).
destruct H as [a R].
exists a.
split.
apply R.
apply (snd (Bexright y _ (expr_extensional_1 _
_))).
apply R.
Defined.

```

Theorem `productchar_2` ($x \ y: V$):

$\forall z, z \in x \times y \leftrightarrow \exists \alpha: V, \alpha \in x \wedge$
 $(\exists \beta: V, \beta \in y \wedge z \doteq \langle \alpha, \beta \rangle).$

Proof.

```

intro z.
split.
intro P.
assert  $(\exists \alpha \in x, \exists \beta \in y, z \doteq \langle \alpha, \beta \rangle )$  as H1.
apply productchar.

```

```

    apply P.
    apply productchar_2_lemma2.
    apply H1.
    intro Q.
    apply productchar.
    apply productchar_2_lemma2.
    apply Q.
Defined.

```

Theorem `product_elim` (`x y z w:V`):
 $\langle z, w \rangle \in x \times y \rightarrow z \in x \wedge w \in y.$

Proof.

```

    intro H.
    assert (∃α: V, α ∈ x ∧
            (∃β: V, β ∈ y ∧ ⟨z, w⟩ = ⟨α, β⟩ ))
as H2.
    apply productchar_2.
    apply H.
    destruct H2 as [a [P [b [P2 P3]]]].
    assert (z = a ∧ w = b) as P31.
    apply (pairing_injective _ _ _ P3).
    split.
    apply (ext1 _ _ (fst P31) P).
    apply (ext1 _ _ (snd P31) P2).
Qed.

```

Theorem `product_intro` (`x y z w:V`):
 $z \in x \wedge w \in y \rightarrow \langle z, w \rangle \in x \times y.$

Proof.

```

    intro H.
    assert (∃α: V, α ∈ x ∧
            (∃β: V, β ∈ y ∧ ⟨z, w⟩ = ⟨α, β⟩ ))
as H2.
    exists z.
    split.
    apply H.

```

```

exists w.
split.
apply H.
apply refV.
apply productchar_2.
apply H2.
Qed.

```

Theorem prod_extensional_half (x y u v:V):
 $x \doteq u \rightarrow y \doteq v \rightarrow x \times y \subseteq u \times v$.

Proof.

```

intros P Q z R.
assert (∃ a : V, a ∈ x ∧ (∃ β : V, β ∈ y ∧ z ≐ ⟨ a, β
> )) as H.
apply productchar_2.
apply R.
apply productchar_2.
destruct H as [a [P1 [b [P2 P3]]]].
exists a.
split.
apply (ext2 _ x _ P1 P).
exists b.
split.
apply (ext2 _ y _ P2 Q).
apply P3.

```

Defined.

Theorem prod_extensional (x y u v:V):
 $x \doteq u \rightarrow y \doteq v \rightarrow x \times y \doteq u \times v$.

Proof.

```

intros P Q. apply ext3.
apply prod_extensional_half.
apply P. apply Q.
apply prod_extensional_half.
apply symV. apply P. apply symV. apply Q.

```

Defined.

Theorem `is_rel_extensional` (`x y R x' y' R' : V`):
 $x \doteq x' \rightarrow y \doteq y' \rightarrow R \doteq R' \rightarrow \text{is_rel } x \ y \ R \rightarrow \text{is_rel } x' \ y' \ R'$.

Proof.

```

intros H1 H2 H3 P.
unfold is_rel in P.
intros z Q.
assert (x × y ≐ x' × y') as H4.
apply prod_extensional.
apply H1. apply H2.
assert (z ∈ x × y) as H5.
apply P.
apply (ext2 _ _ _ Q (symV _ _ H3)).
apply (ext2 _ _ _ H5 H4).

```

Defined.

Theorem `expr_extensional_4` (`a u R : V`):
 $\ll \text{fun } u \Rightarrow \langle a, u \rangle \in R \gg$.

Proof.

```

intros x y P Q.
assert (⟨ a, x ⟩ ≐ ⟨ a, y ⟩) as H.
apply pairing_extensional.
apply refV.
apply Q.
apply (ext1 _ _ _ (symV _ _ H) P).

```

Defined.

Theorem `expr_extensional_5` (`v R : V`):
 $\ll \text{fun } a \Rightarrow \exists b \in v, \langle a, b \rangle \in R \gg$.

Proof.

```

intros x y P Q.
destruct P as [b S].

```

```

exists b.
assert ( < x, v b > ≡ < y, v b > ) as H.
apply pairing_extensional.
apply Q.
apply refV.
apply (ext1 _ _ (symV _ _ H) S).
Defined.

```

Theorem `is_total_char_lemma1` (`x y R:V`):

`is_total x y R` ↔

$\forall a:V, a \in x \rightarrow \exists \beta \in y, \langle a, \beta \rangle \in R.$

Proof.

```

  apply Ballright.

```

```

  apply expr_extensional_5.

```

Defined.

Theorem `is_total_char` (`x y R:V`):

`is_total x y R` ↔

$\forall a:V, a \in x \rightarrow \exists \beta:V, \beta \in y \wedge \langle a, \beta \rangle \in R.$

Proof.

```

  split.

```

```

  intros P a Q.

```

```

  assert (∀a:V, a ∈ x -> ∃β ∈ y, <a,β> ∈ R) as H.

```

```

  apply is_total_char_lemma1.

```

```

  apply P.

```

```

  specialize (H a Q).

```

```

  simpl in H.

```

```

  apply Bexright.

```

```

  apply (expr_extensional_4 _ y _).

```

```

  apply H.

```

```

intro P.

```

```

apply is_total_char_lemma1.

```

```

intros a Q.

```

```

specialize (P a Q).

```

```

annlv /end /Bexright v

```



```

    apply (sym (ext2 _ _ _ y _))
    (expr_extensional_4 _ y _)).
  apply P.
Defined.

```

Theorem is_total_extensional ($x\ y\ R\ x'\ y'\ R':V$):
 $x \doteq x' \rightarrow y \doteq y' \rightarrow R \doteq R' \rightarrow \text{is_total } x\ y\ R \rightarrow$
 $\text{is_total } x'\ y'\ R'.$

Proof.

```

  intros P1 P2 Q1 Q2.
  assert ( $\forall a:V, a \in x \rightarrow \exists \beta:V, \beta \in y \wedge \langle a, \beta \rangle \in R$ )
as H1.
  apply (is_total_char x y R).
  apply Q2.
  assert ( $\forall a:V, a \in x' \rightarrow \exists \beta:V, \beta \in y' \wedge \langle a, \beta \rangle \in$ 
 $R'$ ) as H2.
  intros a H3.
  assert ( $a \in x$ ) as H4.
  apply (ext2 _ _ _ H3).
  apply (symV _ _ P1).

  specialize (H1 a H4).
  destruct H1 as [b [H5 H6 ]].
  exists b.
  split.
  apply (ext2 _ _ _ H5).
  apply P2.
  apply (ext2 _ _ _ H6 Q1).
  apply (is_total_char x' y' R').
  apply H2.
Defined.

```

Theorem functionspace_extensional_half ($x\ y\ u\ v:V$):
 $x \doteq u \rightarrow y \doteq v \rightarrow x \Rightarrow y \subseteq u \Rightarrow v.$

Proof.

```

  intros P Q z H.
  assert (is_function x y z) as H2.

```

```

    apply functionspacechar.
  apply H.
  clear H.
  assert (is_function u v z) as H3.
  destruct H2 as [I [T F]].
  split.
  apply (is_rel_extensional _ _ _ _ _ P Q (refV z)
I).
  split.
  apply (is_total_extensional x y z).
  apply P.
  apply Q.
  apply refV.
  apply T.
  apply F.
  apply functionspacechar.
  apply H3.
Defined.

```

(* The function space construction is extensional
as function of domain and codomain *)

Theorem functionspace_extensional (x y u v:V):
 $x \doteq u \rightarrow y \doteq v \rightarrow x \Rightarrow y \doteq u \Rightarrow v$.

Proof.

```

  intros P Q. apply ext3.
  apply functionspace_extensional_half.
  apply P. apply Q.
  apply functionspace_extensional_half.
  apply symV. apply P. apply symV. apply Q.
Defined.

```

(* A proof that certain
set-theoretic expression is
extensional in a variable *)

Theorem `expr_extensional_7` (`a b c f g:V`):
 $\llcorner (\text{fun } p \Rightarrow \exists a \in a, \exists \beta \in b, \exists \gamma \in c, \langle a, \beta \rangle \in f \wedge \langle \beta, \gamma \rangle \in g \wedge \langle a, \gamma \rangle \doteq p) \rrcorner$.

Proof.

```

intros p p' H K.
destruct H as [ag [bg [cg [P1 [P2 P3]]]]].
exists ag.
exists bg.
exists cg.
split.
apply P1.
split.
apply P2.
apply (traV _ _ _ P3 K).

```

Qed.

(* A characterization of the composition
of two relations *)

Theorem `compose_relV_char` (`a b c f g:V`)
(`P: is_rel a b f`)(`Q: is_rel b c g`):
 $\forall x:V, \forall z:V,$
 $\langle x, z \rangle \in \text{compose_relV } a \ b \ c \ f \ g \leftrightarrow$
 $x \in a \wedge z \in c \wedge$
 $\exists y:V, y \in b \wedge \langle x, y \rangle \in f \wedge \langle y, z \rangle \in g.$

Proof.

```

intros x z.
split.
intro H.
assert (⟨ x, z ⟩ ∈ a × c ∧ ∃ a ∈ a, ∃ β ∈ b, ∃ γ ∈ c,
⟨ a, β ⟩ ∈ f ∧ ⟨ β, γ ⟩ ∈ g ∧ ⟨ a, γ ⟩ ≐ ⟨ x, z ⟩) as
H2.
apply (fst (separationchar
(fun p => ∃ a ∈ a, ∃ β ∈ b, ∃ γ ∈ c, ⟨ a, β ⟩ ∈ f ∧ ⟨ β,
γ ⟩ ∈ g ∧ ⟨ a, γ ⟩ ≐ p)
(expr_extensional_7 (a × c) (f / × g)

```

```

(⟨ x, z ⟩ ∈ a × c ∧ ∃ a' ∈ a, ∃ β ∈ b, ∃ γ ∈ c,
  ⟨ a', β ⟩ ∈ f ∧ ⟨ β, γ ⟩ ∈ g ∧ ⟨ a', γ ⟩ = ⟨ x, z ⟩ ) as
> ))) .
  apply H.
  destruct H2 as [H3 H4].
  split.
  apply (product_elim _ _ _ H3).
  split.
  apply (product_elim _ _ _ H3).
  destruct H4 as [ag [bg [cg [H5 [H6 H7]]]]].
  exists (b bg).
  split.
  unfold memV. exists bg. apply refV.
  split.
  assert ( ⟨ a ag, b bg ⟩ = ⟨ x, b bg ⟩ ) as H8.
  apply pairing_extensional.
  apply (pairing_injective _ _ _ H7).
  apply refV.
  apply (ext1 _ _ _ (symV _ _ H8) H5).
  assert ( ⟨ b bg, z ⟩ = ⟨ b bg, c cg ⟩ ) as H9.
  apply pairing_extensional.
  apply refV.
  apply symV.
  apply (pairing_injective _ _ _ H7).
  apply (ext1 _ _ _ H9 H6).

intro H.
destruct H as [H1 [H2 [y [H3 [H4 H5]]]]].
assert ( ⟨ x, z ⟩ ∈ a × c ∧ ∃ a' ∈ a, ∃ β ∈ b, ∃ γ ∈ c,
  ⟨ a', β ⟩ ∈ f ∧ ⟨ β, γ ⟩ ∈ g ∧ ⟨ a', γ ⟩ = ⟨ x, z ⟩ ) as
H6.
  split.
  apply (product_intro).
  split. apply H1. apply H2.
  unfold memV in H1. destruct H1 as [ag H1'].
  unfold memV in H2. destruct H2 as [cg H2'].
  unfold memV in H3. destruct H3 as [bg H3'].
  exists ag.
  exists ba.

```

```

exists cg.
split.
assert (⟨ x, y ⟩ ≡ ⟨ a ag, b bg ⟩) as E1.
apply pairing_extensional.
apply H1'.
apply H3'.
apply (ext1 _ _ (symV _ _ E1) H4).
split.
assert (⟨ y, z ⟩ ≡ ⟨ b bg, c cg ⟩) as E2.
apply pairing_extensional.
apply H3'.
apply H2'.
apply (ext1 _ _ (symV _ _ E2) H5).
apply pairing_extensional.
apply (symV _ _ H1').
apply (symV _ _ H2').

apply (snd (separationchar
(fun p => ∃ a ∈ a, ∃ β ∈ b, ∃ γ ∈ c, ⟨ a, β ⟩ ∈ f ∧ ⟨ β,
γ ⟩ ∈ g ∧ ⟨ a, γ ⟩ ≡ p)
(expr_extensional_7 _ _ _ _ _ ) (a × c) (⟨ x, z
⟩ )))).
apply H6.
Qed.

```

Theorem `compose_relV_aux` (`a b c f g t:V`):
`t ∈ compose_relV a b c f g -> ∃ x:V, x ∈ a ∧ ∃ y:V,`
`y ∈ c ∧ t ≡ ⟨ x, y ⟩ .`

Proof.

```

assert (compose_relV a b c f g ⊆ a × c) as H.
unfold compose_relV.
apply (sep_sub _ (expr_extensional_7 _ _ _ _ _ ) (a
× c)).
intro P.
assert (t ∈ a × c) as H1

```

```

assert (c ∈ a ∧ c) as H1.
apply H.
apply P.
apply productchar_2.
apply H1.
Qed.

```

Theorem `compose_relV_char_2` (a b c f g:V)
(P: is_rel a b f)(Q: is_rel b c g):
 $\forall t:V, t \in \text{compose_relV } a \ b \ c \ f \ g \leftrightarrow$
 $(\exists x:V, \exists z:V,$
 $t \doteq \langle x, z \rangle \wedge x \in a \wedge z \in c \wedge$
 $\exists y:V, y \in b \wedge \langle x, y \rangle \in f \wedge \langle y, z \rangle \in g).$

Proof.

```

intro t.
split.
intro H.
assert (∃ x:V, x ∈ a ∧ ∃ y:V,
y ∈ c ∧ t ≐ ⟨x, y⟩) as H1.
apply (compose_relV_aux a b c f g t).
apply H.
destruct H1 as [x [H2 [y [H3 H4]]]].
exists x.
exists y.
split.
apply H4.
apply compose_relV_char.
apply P.
apply Q.
apply (ext1 _ _ (symV _ _ H4) H).

intro H.
destruct H as [x [z [H1 H2]]].
assert (⟨x, z⟩ ∈ compose_relV a b c f g).
apply compose_relV_char.
apply D

```

```

    apply r.
    apply Q.
    apply H2.
    apply (ext1 _ _ _ H1 H).
Qed.

```

Theorem `compose_relV_extensional_half` ($a\ b\ c\ f\ g:V$)
 $(P: \text{is_rel } a\ b\ f)(Q: \text{is_rel } b\ c\ g)$
 $(a'\ b'\ c'\ f'\ g':V)$
 $(P': \text{is_rel } a'\ b'\ f')(Q': \text{is_rel } b'\ c'\ g')$
 $(E1: a \doteq a')(E2: b \doteq b')(E3: c \doteq c')$
 $(E4: f \doteq f')(E5: g \doteq g')$:
 $\text{compose_relV } a\ b\ c\ f\ g \subseteq \text{compose_relV } a'\ b'\ c'\ f'\ g'.$

Proof.

```

    intros t H.
    assert (∃ x:V, x ∈ a ∧ ∃ z:V,
z ∈ c ∧ t ≐ ⟨x, z⟩) as H1.
    apply (compose_relV_aux a b c f g).
    apply H.
    destruct H1 as [x [H2 [z [H3 H4]]]].
    assert (⟨x, z⟩ ∈ compose_relV a b c f g).
    apply (ext1 _ _ _ (symV _ _ H4) H).
    assert (x ∈ a ∧ z ∈ c ∧
    ∃y:V, y ∈ b ∧ ⟨x, y⟩ ∈ f ∧ ⟨y, z⟩ ∈ g) as
H5.
    apply compose_relV_char.
    apply P.
    apply Q.
    apply H0.
    assert (x ∈ a' ∧ z ∈ c' ∧
    ∃y:V, y ∈ b' ∧ ⟨x, y⟩ ∈ f' ∧ ⟨y, z⟩ ∈ g') as
H6.
    split.
    apply (ext2 _ _ _ H2 E1).
    split.
    apply (ext2 _ _ _ H3 E3)

```



```

-----
apply compose_relV_extensional_half.

assumption.
assumption.
assumption.
assumption.
apply symV. assumption.
apply symV. assumption.
apply symV. assumption.
apply symV. assumption.
apply symV. assumption.
Qed.

```

(* Corresponding to setoidmap we have .. *)

```

Record Typeoidmap (A B: Typeoid) :=
{
  Typeoidmapfunction :> A → B;
  Typeoidmapextensionality : ∀x y: A, x ≈≈ y →
Typeoidmapfunction x ≈≈ Typeoidmapfunction y
}.

```

(* Proof irrelevant Typeoid families *)

```

Record Typeoidfamily (A: Typeoid) :=
{
  Typeoidfamilyobj :> A → Typeoid;
  Typeoidfamilymap : ∀x y: A, ∀p: x ≈≈ y,
                    Typeoidmap
                    (Typeoidfamilyobj x)
                    (Typeoidfamilyobj y);
  Typeoidfamilyref : ∀x: A, ∀y: Typeoidfamilyobj
x,
    Typeoidfamilymap x x (Typeoidrefl A x) y ≈≈ y;
  Typeoidfamilyirr : ∀x y: A, ∀p q: x ≈≈ y, ∀z:
Typeoidfamilyobj x.

```

```

Typeoidfamilymap x y p z ≈≈≈
Typeoidfamilymap x y q z;
  Typeoidfamilycmp : ∀x y z: A, ∀p: x ≈≈≈ y, ∀q: y
≈≈≈ z, ∀w: Typeoidfamilyobj x,
    (Typeoidfamilymap y z q)
((Typeoidfamilymap x y p) w)
    ≈≈≈ Typeoidfamilymap x z
  (Typeoidtra _ _ _ _ p q) w
}.

```

(* The typeoid of V-sets *)

```

Definition VTypeoid:Typeoid.
  apply (Build_Typeoid V (fun x y => x ÷ y)).
  apply refV.
  apply symV.
  apply traV.
Defined.

```

(* Transform a setoid to a Typeoid *)

```

Definition Magnify (x:setoid): Typeoid.
  apply (Build_Typeoid x (fun u v => u ≈ v)).
  apply setoidrefl.
  apply setoidsym.
  apply setoidtra.
Defined.

```

(* From a proof equality of two V-sets we obtain
two operations push and pull that tells
how the elements of the V-sets correspond. *)

```

Definition push {ax:Set}{fx:ax->V}
  {ay:Set}{fy:ay->V}
  (p: (sup ax fx) ÷ (sup ay fy))
(†:ax):= nroiT1((fst n) †):av.

```

`(t:ax):= projT1((snd p) t):ay.`

Theorem `pushprop` $\{ax:Set\}\{fx:ax \rightarrow V\}$
 $\{ay:Set\}\{fy:ay \rightarrow V\}$
 $(p: (\sup ax\ fx) \doteq (\sup ay\ fy)):$
 $(\forall t: ax, fx\ t \doteq fy\ (push\ p\ t)).$

Proof.

`intro t.`

`apply (projT2 (fst p t)).`

Defined.

Definition `pull` $\{ax:Set\}\{fx:ax \rightarrow V\}$
 $\{ay:Set\}\{fy:ay \rightarrow V\}$
 $(p: (\sup ax\ fx) \doteq (\sup ay\ fy))$
 $(t:ay):= projT1((snd p)\ t):ax.$

Theorem `pullprop` $\{ax:Set\}\{fx:ax \rightarrow V\}$
 $\{ay:Set\}\{fy:ay \rightarrow V\}$
 $(p: (\sup ax\ fx) \doteq (\sup ay\ fy)):$
 $(\forall t: ay, fx\ (pull\ p\ t) \doteq fy\ t).$

Proof.

`intro t.`

`apply (projT2 (snd p t)).`

Defined.

(Rbar is V-sets as a Typeoid family over VTypeoid *)*

Definition `Rbar_ob` $(x:VTypeoid) := Magnify\ (obfn\ x).$

Definition `Rbar_transp_helper`:

$\forall x\ y: VTypeoid, \forall p: x \approx \approx y,$
 $(Rbar_ob\ x) \rightarrow$
 $(Rbar_ob\ y).$

`intros [ix fx ] [iy fy].`

`intros H t.`

`apply (push H t).`

Defined

Defined.

```
Definition Rbar_transp:  $\forall x\ y: VTypeoid, \forall p: x \approx\approx y,$   
Typeoidmap  
  (Rbar_ob x)  
  (Rbar_ob y).  
  
intros x y p.  
apply (Build_Typeoidmap (Rbar_ob x) (Rbar_ob y)  
      (Rbar_transp_helper x y p)).  
intros s t H.  
destruct x as [ix fx].  
destruct y as [iy fy].  
assert (fy (Rbar_transp_helper (sup ix fx) (sup iy  
fy) p s)  $\doteq$   
  (fy  
    (Rbar_transp_helper (sup ix fx) (sup iy fy) p t)))  
as H0.  
assert (fx s  $\doteq$  fy (Rbar_transp_helper (sup ix fx)  
(sup iy fy) p s)) as H1.  
apply (pushprop p s).  
assert (fx t  $\doteq$  fy (Rbar_transp_helper (sup ix fx)  
(sup iy fy) p t)) as H2.  
apply (pushprop p t).  
assert (fx s  $\doteq$  fx t) as H3.  
apply H.  
apply (traV _ _ _ (traV _ _ _ (symV _ _ H1) H3)  
H2).  
apply H0.  
Defined.
```

```
Definition Rbar : Typeoidfamily VTypeoid.  
  apply (Build_Typeoidfamily  
    VTypeoid Rbar_ob Rbar_transp).  
  intros [ix fx].  
  intro t.  
  apply Typeoidsym.  
  assert (fx t  $\doteq$ 
```

```

    assert (fx t =
      fx ((Rbar_transp (sup ix fx) (sup ix fx)
(Typeoidrefl VTypeoid (sup ix fx))) t)) as H1.
    destruct (Typeoidrefl VTypeoid (sup ix fx)) as
[p1 p2].
    apply (projT2 (p1 t)).
    apply H1.

intros x y p q t.
destruct x as [ix fx].
destruct y as [iy fy].
assert ( fy ((Rbar_transp (sup ix fx) (sup iy fy)
p) t) = fy ((Rbar_transp (sup ix fx) (sup iy fy)
q) t) ) as H1.
assert (fx t = fy ((Rbar_transp (sup ix fx) (sup
iy fy) p) t) ) as H2.
apply (pushprop p t).
assert (fx t = fy ((Rbar_transp (sup ix fx) (sup
iy fy) q) t) ) as H3.
apply (pushprop q t).
apply (traV _ _ _ (symV _ _ H2) H3).
apply H1.

intros x y z p q t.
destruct x as [ix fx].
destruct y as [iy fy].
destruct z as [iz fz].
assert (fx t = fy ((Rbar_transp (sup ix fx) (sup
iy fy) p) t) ) as H1.
apply (pushprop p t).

assert (fy ((Rbar_transp (sup ix fx) (sup iy fy) p)
t) = fz ((Rbar_transp (sup iy fy) (sup iz fz) q)
((Rbar_transp (sup ix fx) (sup iy fy) p) t))) as H2.
destruct q as [q1 q2].
apply (projT2 (q1 ((Rbar_transp (sup ix fx) (sup iy
fy) p) t))).
assert ( sup ix fx ~~~ f VTypeoid (sup iz fz) as

```

```

    assert (sup ix fx ≈≈, VTypeoid (sup iz fz) us
H3.

```

```

    apply (Typeoidtra VTypeoid (sup ix fx) (sup iy fy)
(sup iz fz) p q).

```

```

    assert (fx t ≐
fz ((Rbar_transp (sup ix fx) (sup iz fz)
      (Typeoidtra VTypeoid (sup ix fx) (sup iy fy)
(sup iz fz) p q)) t)) as H4.

```

```

    destruct (Typeoidtra VTypeoid (sup ix fx) (sup iy
fy) (sup iz fz) p q) as [r1 r2].

```

```

    apply (projT2 (r1 t)).

```

```

    apply (traV _ _ _ (traV _ _ _ (symV _ _ H2) (symV _
_ H1)) H4).

```

```

Defined.

```

```

(* Category of Type-size *)

```

```

Record Cat: Type :=

```

```

{

```

```

  Catobj  := Typeoid;

```

```

  Catarr  : Typeoid;

```

```

  Catcms  : Typeoid;

```

```

  Catid   : Typeoidmap Catobj Catarr;

```

```

  Catdom  : Typeoidmap Catarr Catobj;

```

```

  Catcod  : Typeoidmap Catarr Catobj;

```

```

  Catfst  : Typeoidmap Catcms Catarr;

```

```

  Catsnd  : Typeoidmap Catcms Catarr;

```

```

  Catcmp  : Typeoidmap 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 ->

```

```

CatK6 :  $\forall f g. \text{Catarr } f \approx \text{Catcod } g \rightarrow$ 
 $\exists u : \text{Catcms}, \text{Catsnd } u \approx f \wedge \text{Catfst } u \approx g;$ 

CatK7 :  $\forall u : \text{Catcms}, \forall y : \text{Catobj}, \text{Catfst } u \approx$ 
Catid y  $\rightarrow$ 
    Catcmp u  $\approx$  Catsnd u;
CatK8 :  $\forall u : \text{Catcms}, \forall x : \text{Catobj}, \text{Catsnd } u \approx$ 
Catid x  $\rightarrow$ 
    Catcmp u  $\approx$  Catfst u;
CatK9 :  $\forall u v : \text{Catcms}, \forall w z : \text{Catcms},$ 
    Catfst w  $\approx$  Catfst v  $\rightarrow$  Catsnd v  $\approx$  Catfst u
 $\rightarrow$ 
    Catsnd u  $\approx$  Catsnd z  $\rightarrow$  Catsnd w  $\approx$  Catcmp u
 $\rightarrow$ 
    Catcmp v  $\approx$  Catfst z  $\rightarrow$  Catcmp w  $\approx$  Catcmp z
}.

```

```

Record Functor (A B:Cat):Type :=
{Funob: Typeoidmap (Catobj A) (Catobj B);
Funarr: Typeoidmap (Catarr A) (Catarr B);
Funcms: Typeoidmap (Catcms A) (Catcms B);
Fun_id:  $\forall a : \text{Catobj } A,$ 
    Funarr (Catid A a)  $\approx$  (Catid B (Funob a));
Fun_dom:  $\forall a : \text{Catarr } A,$ 
    Funob (Catdom A a)  $\approx$  (Catdom B (Funarr
a));
Fun_cod:  $\forall a : \text{Catarr } A,$ 
    Funob (Catcod A a)  $\approx$  (Catcod B (Funarr
a));
Fun_fst:  $\forall a : \text{Catcms } A,$ 
    Funarr (Catfst A a)  $\approx$  (Catfst B (Funcms
a));
Fun_snd:  $\forall a : \text{Catcms } A,$ 
    Funarr (Catsnd A a)  $\approx$  (Catsnd B (Funcms
a));
Fun_cmp:  $\forall a : \text{Catcms } A,$ 
    Funarr (Catcmp A a)  $\approx$  (Catcmp B (Funcms
a))
}

```

}.

(* TWO DIFFERENT CATEGORIES

Two different categories of V-sets are defined
and proved isomorphic

*)

(* Build the category of V-sets *)

Definition ObV:=VTypeoid.

(* predicate for a set to be an arrow *)

Definition is_arrow (u:V) :=
 $\exists a:V, \exists b:V, \exists f:V,$
 $u \doteq \langle \langle a, b \rangle, f \rangle \wedge \text{is_function } a \ b \ f.$

Definition ArrV_under := $\exists u:V, \text{is_arrow } u.$

Definition ArrV:Typeoid.
 apply (Build_Typeoid ArrV_under
 (fun u v =>
 projT1 u $\doteq$ projT1 v)).
 intro x. apply refV.
 intros x y. apply symV.
 intros x y z. apply traV.

Defined.

Definition make_ArrV

(a b f : V)
(P: is_function a b f):=
 (existT _ $\langle \langle a, b \rangle, f \rangle$
 (existT _ a
 (existT _ b
 (existT _ f
 ((refV _), P))))))): ArrV.

Theorem id_arrowpf (x:V): is_arrow $\langle \langle x, x \rangle$,
identity x $\rangle$.

Proof.

```
exists x. exists x. exists (identity x).
split.
apply pairing_extensional.
apply pairing_extensional.
apply refV. apply refV. apply refV.
apply id_is_function.
```

Defined.

(* Proofs that certain
set-theoretic expressions are
extensional in a variable *)

Theorem expr_extproof_1 (x:V):
«(fun z => $\exists y : x, z \doteq \langle x y, x y \rangle$)».

Proof.

```
unfold extensionalV.
intros z z' P Q.
destruct P as [y P].
exists y.
apply (traV _ _ (symV _ _ Q) P).
```

Defined.

Theorem expr_extproof_2 (z:V):
«(fun u => $z \doteq \langle u, u \rangle$)».

Proof.

```
unfold extensionalV.
intros u u' P Q.
assert (  $\langle u, u \rangle \doteq \langle u', u' \rangle$  ) as H.
apply pairing_extensional.
apply Q.
apply Q.
```

```

    apply (traV _ _ _ P H).
Defined.

```

Theorem `identity_ext_lemma_half` ($x\ y: V$)($p: x \doteq y$):
 $(\text{identity } x) \sqsubseteq (\text{identity } y)$.

Proof.

```

    intro z.
    intro P.
    unfold identity in P.
    assert (z ∈ x × x ∧ ∃y : x, z ≐ ⟨ x y, x y ⟩ ) as
H.
    apply (separationchar (fun z => ∃y : x, z ≐ ⟨ x y,
x y ⟩ )
    (expr_extproof_1 x) (x × x) z).
    apply P.
    assert (z ∈ y × y ∧ ∃t : y, z ≐ ⟨ y t, y t ⟩ ) as
H2.
    destruct H as [H0 H1].
    assert (∃w, w ∈ x ∧ z ≐ ⟨w, w⟩ ) as H3.
    apply Bexright.
    unfold extensionalV.
    intros u v Q R.
    assert (⟨ u, u ⟩ ≐ ⟨ v, v ⟩ ) as H4.
    apply pairing_extensional.
    apply R. apply R.
    apply (traV _ _ _ Q H4).
    apply H1.
    split.
    assert (x × x ≐ y × y) as H4.
    apply prod_extensional.
    apply p. apply p.
    apply (ext2 _ _ _ H0 H4).
    assert (∃w : V, w ∈ y ∧ z ≐ ⟨ w, w ⟩ ) as H5.
    destruct H3 as [w H6].
    exists w.
    split.

```

```

destruct H6 as [H6a H6b].
apply (ext2 _ x _). apply H6a. apply p.
apply H6.
apply ((snd (Bexright y (fun u => z ≐ ⟨ u, u ⟩ )
  (expr_extproof_2 z)))) H5).
unfold identity.
apply separationchar.
apply (expr_extproof_1).
apply H2.
Defined.

```

Theorem `identity_ext_lemma` ($x\ y: V$)($p: x \doteq y$):
 $(\text{identity } x) \doteq (\text{identity } y)$.

Proof.

```

apply ext3.
apply identity_ext_lemma_half.
apply p.
apply identity_ext_lemma_half.
apply symV. apply p.

```

Defined.

Definition `idV`:Typeoidmap ObV ArrV.

```

apply (Build_Typeoidmap ObV ArrV
  (fun x => existT _ ⟨ ⟨x, x⟩ , identity x ⟩
(id_arrowpf x))).
intros x y P.
assert ( ⟨ ⟨ x, x ⟩ , identity x ⟩ ≐ ⟨ ⟨ y, y
⟩ , identity y ⟩ ) as H.
apply pairing_extensional.
apply pairing_extensional.
apply P. apply P.
apply identity_ext_lemma.
apply P.
apply H.

```

Defined.

```

Definition domV: Typeoidmap ArrV ObV.
  apply (Build_Typeoidmap ArrV ObV
    (fun w => projT1 (projT2 w))).
  intros w w' P.
  destruct w as [u [a [b [f [Q R]]]]].
  destruct w' as [u' [a' [b' [f' [Q' R']]]]].
  simpl.
  simpl in P.
  assert (⟨ ⟨ a, b ⟩ , f ⟩ ≐ ⟨ ⟨ a', b' ⟩ , f'
  ⟩ ) as H.
  apply (traV _ _ _ (symV _ _ Q) (traV _ _ _ P Q')).
  assert (⟨ a, b ⟩ ≐ ⟨ a', b' ⟩ ) as H2.
  apply (pairing_injective _ _ _ _ H).
  apply (pairing_injective _ _ _ _ H2).
Defined.

```

```

Definition codV: Typeoidmap ArrV ObV.
  apply (Build_Typeoidmap ArrV ObV
    (fun w => projT1 (projT2 (projT2 w)))).
  intros w w' P.
  destruct w as [u [a [b [f [Q R]]]]].
  destruct w' as [u' [a' [b' [f' [Q' R']]]]].
  simpl.
  simpl in P.
  assert (⟨ ⟨ a, b ⟩ , f ⟩ ≐ ⟨ ⟨ a', b' ⟩ , f'
  ⟩ ) as H.
  apply (traV _ _ _ (symV _ _ Q) (traV _ _ _ P Q')).
  assert (⟨ a, b ⟩ ≐ ⟨ a', b' ⟩ ) as H2.
  apply (pairing_injective _ _ _ _ H).
  apply (pairing_injective _ _ _ _ H2).
Defined.

```

```

Definition CmsV_under := ∃ w:V,
  ∃ u:ArrV, ∃ v:ArrV,
  w ≐ ⟨projT1 u, projT1 v⟩ ∧
  codV u ≐ domV v.

```

Definition CmsV:Typeoid.

```
apply (Build_Typeoid CmsV_under
  (fun u v =>
    projT1 u ≐ projT1 v)).
intro x. apply refV.
intros x y. apply symV.
intros x y z. apply traV.
```

Defined.

Definition make_CmsV

```
(c d g : V)
(P : is_function c d g)
(a b f : V)
(Q : is_function a b f)
(H : c ≐ b):=
(existT _ (⟨ ⟨ ⟨ a, b ⟩ , f ⟩ , ⟨ ⟨ c, d ⟩ , g ⟩
⟩ )
(existT _ (make_ArrV a b f Q)
  (existT _ (make_ArrV c d g P)
    ((refV _), (symV _ _ H)))))) : CmsV.
```

Definition fstV: Typeoidmap CmsV ArrV.

```
apply (Build_Typeoidmap CmsV ArrV
  (fun z => projT1 (projT2 z))).
intros [w [u [v Q]]] [w' [u' [v' Q']]] P.
simpl in P.
simpl.
assert (⟨ projT1 u, projT1 v ⟩ ≐ ⟨ projT1 u',
projT1 v' ⟩) as H.
apply (traV _ _ _ (symV _ _ (fst Q))
  (traV _ _ _ P (fst Q')))).
assert (projT1 u ≐ projT1 u' ∧ projT1 v ≐ projT1
v') as H2.
apply pairing_injective.
```

```

    apply H.
    apply H2.

```

Defined.

Definition `sndV`: Typeoidmap CmsV ArrV.

```

    apply (Build_Typeoidmap CmsV ArrV
      (fun z => projT1 (projT2 (projT2 z)))).
    intros [w [u [v Q]]] [w' [u' [v' Q']]] P.
    simpl in P.
    simpl.
    assert ( < projT1 u, projT1 v >  = < projT1 u',
projT1 v' > ) as H.
    apply (traV _ _ _ (symV _ _ (fst Q))
      (traV _ _ _ P (fst Q')))).
    assert (projT1 u = projT1 u' ^ projT1 v = projT1
v') as H2.
    apply pairing_injective.
    apply H.
    apply H2.

```

Defined.

Definition `cmpV_helper`: CmsV -> ArrV.

```

    intros [w [[fbl P] [[gbl Q] R]]].
    destruct P as [a [b [f [P1 P2]]]].
    destruct Q as [c [d [g [Q1 Q2]]]].
    simpl in R.
    destruct R as [R1 R2].
    exists < < a, d > , (compose_relV a b d f g) > .
    exists a.
    exists d.
    exists (compose_relV a b d f g).
    split.
    apply refV.
    split.
    unfold is_rel.

```

```

apply sep_sub.
intros m m' [a [β [γ [αRβ [βSγ eq]]]]] meqm'.
exists a. exists β. exists γ.
split. apply αRβ.
split. apply βSγ.
apply (traV _ _ _ eq meqm').
split.
destruct P2 as [P20 [P21 P22]].
destruct Q2 as [Q20 [Q21 Q22]].
assert (is_total b d g) as H.
apply (is_total_extensional c d g b d g
(symV _ _ R2) (refV _) (refV _) Q21).
  assert (∀x:V, x ∈ b -> ∃y:V, y ∈ d ∧ ⟨x,y⟩ ∈ g)
as H1.
  apply is_total_char.
  apply H.
  assert (∀x:V, x ∈ a -> ∃y:V, y ∈ b ∧ ⟨x,y⟩ ∈ f)
as H2.
  apply is_total_char.
  apply P21.
  assert (∀x:V, x ∈ a -> ∃y:V, y ∈ d ∧ ⟨x,y⟩ ∈
(compose_relV a b d f g) ) as H3.
  intros x N.
  specialize (H2 x N).
  destruct H2 as [y [M P]].
  specialize (H1 y M).
  destruct H1 as [z [M' P']].
  exists z.
  split.
  apply M'.
  assert (is_rel b d g) as Q20'.
  apply (is_rel_extensional c d g).
  apply (symV _ _ R2).
  apply refV.
  apply refV.
  apply Q20.
  apply (snd (compose_relV_char a b d f g P20 Q20' x

```

```

z)).
  split.
  apply N.
  split.
  apply M'.
  exists y.
  split.
  apply M.
  split.
  apply P.
  apply P'.
  apply is_total_char.
  apply H3.
  unfold is_functional.
  intros x y y' [H1 H2].
  unfold is_function in P2.
  unfold is_function in Q2.
  assert (x ∈ a ∧ y ∈ d ∧
    ∃z:V, z ∈ b ∧ ⟨ x, z ⟩ ∈ f ∧ ⟨ z, y ⟩ ∈ g) as
H1'.
  apply compose_relV_char.
  apply P2.
  apply (is_rel_extensional c d g).
  apply (symV _ _ R2).
  apply refV.
  apply refV.
  apply Q2.
  apply H1.
  destruct H1' as [H11 [H12 [z H13]]].
  assert (x ∈ a ∧ y' ∈ d ∧
    ∃z:V, z ∈ b ∧ ⟨ x, z ⟩ ∈ f ∧ ⟨ z, y' ⟩ ∈ g) as
H2'.
  apply compose_relV_char.
  apply P2.
  apply (is_rel_extensional c d g).
  apply (symV _ _ R2).
  apply refV.

```



```

apply refV.
apply Q2.

apply H2.
destruct H2' as [H21 [H22 [z' H23]]].
destruct P2 as [_ [_ P2']].
unfold is_functional in P2'.
assert (z ≐ z') as H4.
apply (P2' x).
split.
apply H13.
apply H23.
assert (⟨ z', y' ⟩ ≐ ⟨ z, y' ⟩) as H5.
apply pairing_extensional.
apply (symV _ _ H4).
apply refV.
assert (⟨ z, y' ⟩ ∈ g) as H6.
apply (ext1 _ _ _ (symV _ _ H5)).
apply H23.
destruct Q2 as [_ [_ Q2']].
unfold is_functional in Q2'.
apply (Q2' z).
split.
apply H13.
apply H6.
Defined.

```

Lemma singleton_extensional (x y: V):

$x \doteq y \rightarrow \{\{ x \}\} \doteq \{\{ y \}\}.$

Proof.

```

intro H.
apply (Extensionality {\{ x \}} {\{ y \}} ).
intro z.
split.
intro H1.
unfold memV in H1.
destruct H1 as [ix P].

```

```

exists ix.
  simpl in P.

  simpl.
  apply (traV _ _ _ P H).
  intro H1.
  unfold memV in H1.
  destruct H1 as [iy P].
  exists iy.
  simpl in P.
  simpl.
  apply (traV _ _ _ P (symV _ _ H)).
Qed.

```

(* A proof that certain
set-theoretic expression is
extensional in a variable *)

Theorem `expr_extensional_8` (`z:V`):
«(`fun p => (∃v : z, p ≡ < z v, z v > )`)».

Proof.

```

intros x y P Q.
destruct P as [v P2].
exists v.
apply (traV _ _ _ (symV _ _ Q) P2).
Qed.

```

Theorem `identity_lemma` (`x y z:V`):
`< x, y > ∈ identity z -> x ≡ y.`

Proof.

```

intro H.
unfold identity in H.
assert (( < x, y > ∈ z × z) ∧ (∃v : z, < x, y > ≡
< z v, z v > )) as H2.
apply (fst (separationchar
(fun p => (∃v : z. p ≡ < z v. z v > )))

```

```

(expr_extensional_8 z) (z x z) < x, y >)).
  apply H.
  destruct H2 as [_ [v H3]].
  assert (x = z v) as H4.
  apply (pairing_injective _ _ _ H3).
  assert (y = z v) as H5.
  apply (pairing_injective _ _ _ H3).
  apply (traV _ _ H4 (symV _ _ H5)).
Qed.

```

Theorem identity_lemma_2 (x y:V):
 $x \in y \rightarrow \langle x, x \rangle \in \text{identity } y$.

Proof.

```

  intro H.
  unfold identity.
  assert ((⟨ x, x ⟩ ∈ y × y) ∧ (∃ v : y, ⟨ x, x ⟩ =
  ⟨ y v, y v ⟩)) as H2.
  split.
  apply product_intro.
  split.
  apply H.
  apply H.
  unfold memV in H.
  destruct H as [idx H2].
  exists idx.
  apply pairing_extensional.
  apply H2.
  apply H2.
  apply (snd (separationchar
  (fun p => (∃ v : y, p = ⟨ y v, y v ⟩)))
  (expr_extensional_8 y) (y × y) < x, x >)).
  apply H2.
Qed.

```

Theorem is_rel_pairs (a b r x:V)(P:is_rel a b r):
 $x \in r \rightarrow \exists v : V. \exists z : V. v \in a \wedge z \in b$

$\wedge x \doteq \langle y, z \rangle$.

Proof.

```

intro H.
unfold is_rel in P.
assert (x ∈ a × b) as H2.
apply P.
apply H.
assert ((∃ y : V, y ∈ a ∧ (∃ z : V, z ∈ b ∧ x ≐ ⟨ y,
z ⟩ ))) as H3.
apply productchar_2.
apply H2.
destruct H3 as [y [H4 [z [H5 H6]]]].
exists y.
exists z.
split. apply H4.
split. apply H5. apply H6.

```

Qed.

Definition `cmpV`: Typeoidmap CmsV ArrV.

```

apply (Build_Typeoidmap CmsV ArrV cmpV_helper).
intros [w [[fbl P] [[gbl Q] [R1 R2]]]]
      [w' [[fbl' P'] [[gbl' Q'] [R1' R2']]]] H.
destruct P as [a [b [f [P1 P2]]]].
destruct P' as [a' [b' [f' [P1' P2']]]].
destruct Q as [c [d [g [Q1 Q2]]]].
destruct Q' as [c' [d' [g' [Q1' Q2']]]].
simpl in R1.
simpl in R2.
simpl in R1'.
simpl in R2'.
simpl in H.
simpl.
assert (⟨ fbl, gbl ⟩ ≐ ⟨ fbl', gbl' ⟩) as H2.
apply (traV _ _ _ (symV _ _ R1) (traV _ _ _ H
R1')).

```

```

-- ...
assert (fbl ≐ fbl') as H21.
apply (pairing_injective _ _ _ _ H2).

assert (gbl ≐ gbl') as H22.
apply (pairing_injective _ _ _ _ H2).
assert (⟨ ⟨ a, b ⟩ , f ⟩ ≐ ⟨ ⟨ a', b' ⟩ , f' ⟩ )
as H3.
apply (traV _ _ _ (symV _ _ P1) (traV _ _ _ H21
P1')).
assert (⟨ ⟨ c, d ⟩ , g ⟩ ≐ ⟨ ⟨ c', d' ⟩ , g' ⟩ )
as H4.
apply (traV _ _ _ (symV _ _ Q1) (traV _ _ _ H22
Q1')).
assert (a ≐ a') as H31.
apply (pairing_injective _ _ _ _
(fst (pairing_injective _ _ _ _ H3))).
assert (b ≐ b') as H32.
apply (pairing_injective _ _ _ _
(fst (pairing_injective _ _ _ _ H3))).
assert (f ≐ f') as H33.
apply (pairing_injective _ _ _ _ H3).
assert (c ≐ c') as H41.
apply (pairing_injective _ _ _ _
(fst (pairing_injective _ _ _ _ H4))).
assert (d ≐ d') as H42.
apply (pairing_injective _ _ _ _
(fst (pairing_injective _ _ _ _ H4))).
assert (g ≐ g') as H43.
apply (pairing_injective _ _ _ _ H4).
assert (⟨ ⟨ a, d ⟩ , compose_relV a b d f g ⟩
≐ ⟨ ⟨ a', d' ⟩ , compose_relV a' b' d' f' g' ⟩
) as H5.
apply pairing_extensional.
apply pairing_extensional.
apply H31.
apply H42.
apply compose_relV_extensional.
unfold is_function in P2

```

```

unfold is_function in P2.
apply P2.
assert (is_rel b d g) as Q21.
unfold is_function in Q2.
apply (is_rel_extensial c d g b d g).
apply (symV _ _ R2).
apply refV.
apply refV.
apply Q2.
apply Q21.
unfold is_function in P2'.
apply P2'.
assert (is_rel b' d' g') as Q21'.
unfold is_function in Q2'.
apply (is_rel_extensial c' d' g' b' d' g').
apply (symV _ _ R2').
apply refV.
apply refV.
apply Q2'.
apply Q21'.
assumption.
assumption.
assumption.
assumption.
assumption.
apply H5.
Defined.

```

Theorem Vcat: Cat.

Proof.

```

apply (Build_Cat ObV ArrV CmsV idV domV codV fstV
sndV cmpV).
intro x.
unfold idV.
unfold domV.
simpl.

```

```
apply refV.
```

```
intro x.  
unfold idV.  
unfold codV.  
simpl.  
apply refV.
```

```
intro u.  
destruct u as [x [arr1 [arr2 [PA PB]]]].  
destruct arr1 as [v [a [b [f [P11 P12]]]]].  
destruct arr2 as [w [c [d [g [P21 P22]]]]].  
simpl in PA.  
simpl in PB.  
simpl.  
apply refV.
```

```
intro u.  
destruct u as [x [arr1 [arr2 [PA PB]]]].  
destruct arr1 as [v [a [b [f [P11 P12]]]]].  
destruct arr2 as [w [c [d [g [P21 P22]]]]].  
simpl in PA.  
simpl in PB.  
simpl.  
apply refV.
```

```
intros u u' H1 H2.  
destruct u as [x [arr1 [arr2 [PA PB]]]].  
destruct arr1 as [v [a [b [f [P11 P12]]]]].  
destruct arr2 as [w [c [d [g [P21 P22]]]]].  
destruct u' as [x' [arr1' [arr2' [PA' PB']]].  
destruct arr1' as [v' [a' [b' [f' [P11' P12']]]].  
destruct arr2' as [w' [c' [d' [g' [P21' P22']]]].  
simpl in PA.  
simpl in PB.  
simpl in PA'.  
simpl in PB'.
```

```
--impl -- ... --
```

```
simpl in H1.
```

```
simpl in H2.
```

```
simpl.
```

```
assert ( < v, w > ≐ < v', w' > ) as H3.
```

```
apply pairing_extensional.
```

```
apply H1.
```

```
apply H2.
```

```
apply (traV _ _ _ PA (traV _ _ _ H3  
      (symV _ _ PA')))).
```

```
intros arr2 arr1 H.
```

```
destruct arr2 as [w [c [d [g [P21 P22]]]]].
```

```
destruct arr1 as [v [a [b [f [P11 P12]]]]].
```

```
unfold domV in H.
```

```
unfold codV in H.
```

```
simpl in H.
```

```
exists (make_CmsV c d g P22 a b f P12 H).
```

```
split.
```

```
unfold make_CmsV.
```

```
apply (symV _ _ P21).
```

```
apply (symV _ _ P11).
```

```
(* id *)
```

```
intros u y P.
```

```
destruct u as [x [arr1 [arr2 [PA PB]]]].
```

```
destruct arr1 as [v [a [b [f [P11 P12]]]]].
```

```
destruct arr2 as [w [c [d [g [P21 P22]]]]].
```

```
simpl in P.
```

```
simpl in PA.
```

```
simpl in PB.
```

```
unfold cmpV.
```

```
simpl.
```

```
assert ( < < a, d > , compose_relV a b d f g > ≐ w )  
as H.
```

```
assert ( < < a, b > , f > ≐ < < y, y > , identity  
v \ \ as H2
```


y / y as H5.

apply (traV _ _ _ (symV _ _ P11) P).
assert (⟨ a, b ⟩ ≐ ⟨ y, y ⟩) as H4.

apply (pairing_injective _ _ _ _ H3).

assert (a ≐ y) as H5.

apply (pairing_injective _ _ _ _ H4).

assert (b ≐ y) as H6.

apply (pairing_injective _ _ _ _ H4).

assert (f ≐ identity y) as H7.

apply (pairing_injective _ _ _ _ H3).

assert (⟨ ⟨ a, d ⟩ , compose_relV a b d f g ⟩ ≐
⟨ ⟨ c, d ⟩ , g ⟩) as H2.

apply pairing_extensional.

apply pairing_extensional.

apply (traV _ _ _ H5 (traV _ _ _ (symV _ _ H6)
PB)).

apply refV.

assert (compose_relV a b d (identity y) g ≐
compose_relV a b d f g) as H8.

apply compose_relV_extensional.

assert (is_rel y y (identity y)) as H9.

assert (is_function y y (identity y)) as HA.

apply id_is_function.

apply HA.

apply (is_rel_extensional y y (identity y) a b
(identity y) (symV _ _ H5) (symV _ _ H6) (refV _)
H9).

apply (is_rel_extensional c d g b d g
(symV _ _ PB)

(refV _) (refV _)).

apply P22.

apply P12.

apply (is_rel_extensional c d g b d g
(symV _ _ PB)

(refV _) (refV _)).

apply P22.

apply (refV _).

apply (refV _).

```

    apply (refV _).
    apply (refV _).
    apply (symV _ _ H7).

    apply (refV _).
    assert (compose_relV a b d (identity y) g  $\doteq$  g) as
HB.

```

```

    apply ext3.
    intros t Q.
    assert (( $\exists$  x':V,  $\exists$  z':V,
t  $\doteq$   $\langle$  x', z'  $\rangle$   $\wedge$  x'  $\in$  a  $\wedge$  z'  $\in$  d  $\wedge$ 
 $\exists$  y':V, y'  $\in$  b  $\wedge$   $\langle$  x', y'  $\rangle$   $\in$  (identity y)  $\wedge$   $\langle$ 
y', z'  $\rangle$   $\in$  g)) as X.
    apply compose_relV_char_2.
    assert (is_rel y y (identity y)) as H9.
    assert (is_function y y (identity y)) as HA.
    apply id_is_function.
    apply HA.
    apply (is_rel_extensional y y (identity y) a b
(identity y) (symV _ _ H5) (symV _ _ H6) (refV _)
H9).
    apply (is_rel_extensional c d g b d g
(symV _ _ PB)
(refV _) (refV _)).
    apply P22.
    apply Q.
    destruct X as [x' [z' [ar [PC [PD PE]]]]].
    destruct PE as [y' [PF [PG PH]]].
    assert (  $\langle$  x', z'  $\rangle$   $\doteq$   $\langle$  y', z'  $\rangle$  ) as X2.
    apply pairing_extensional.
    apply (identity_lemma _ _ _ PG).
    apply refV.
    assert (t  $\doteq$   $\langle$  y', z'  $\rangle$  ) as X3.
    apply (traV _ _ _ ar X2).
    apply (ext1 _ _ _ X3 PH).

```

```

intros t H.
assert (compose_relV a b d (identity y) g  $\doteq$  g) as

```

```

assert (∃ x' : V, ∃ z' : V,
t ≡ ⟨ x', z' ⟩ ∧ x' ∈ a ∧ z' ∈ d ∧
  ∃ y' : V, y' ∈ b ∧ ⟨ x', y' ⟩ ∈ (identity y) ∧ ⟨
y', z' ⟩ ∈ g) as G.
unfold is_function in P22.
assert (∃ x' : V, ∃ z' : V, x' ∈ c ∧ z' ∈ d
  ∧ t ≡ ⟨ x', z' ⟩) as X.
apply (is_rel_pairs c d g t).
apply P22.
apply H.
destruct X as [x' [z' [PC [PD PE]]]].
exists x'.
exists z'.
split.
apply PE.
split.
assert (c ≡ a) as H9.
apply (traV _ _ (symV _ _ PB) (traV _ _ _ H6
(symV _ _ H5))).
apply (ext2 _ _ _ PC H9).
split. apply PD.
exists x'.
split.
apply (ext2 _ _ _ PC (symV _ _ PB)).
split.
apply identity_lemma_2.
assert (c ≡ y) as HA.
apply (traV _ _ (symV _ _ PB) H6).
apply (ext2 _ _ _ PC HA).
apply (ext1 _ _ (symV _ _ PE) H).

apply compose_relV_char_2.

assert (is_rel y y (identity y)) as H9.
assert (is_function y y (identity y)) as HA.
apply id_is_function.
apply HA.
apply (is_rel_extensional y y (identity y) a b

```

```

    apply (is_rel_extensional y y (identity y) a b
(identity y) (symV _ _ H5) (symV _ _ H6) (refV _)
H9).

```

```

    apply (is_rel_extensional c d g b d g
(symV _ _ PB)
(refV _) (refV _)).

```

```

    apply P22.

```

```

    apply G.

```

```

    apply (traV _ _ _ (symV _ _ H8) HB).

```

```

    apply (traV _ _ _ H2 (symV _ _ P21)).

```

```

    apply H.

```

```

intros u y P.

```

```

destruct u as [x [arr1 [arr2 [PA PB]]]].

```

```

destruct arr1 as [v [a [b [f [P11 P12]]]]].

```

```

destruct arr2 as [w [c [d [g [P21 P22]]]]].

```

```

simpl in P.

```

```

simpl in PA.

```

```

simpl in PB.

```

```

unfold cmpV.

```

```

simpl.

```

```

assert ( < < a, d > , compose_relV a b d f g > ≡ v )
as H.

```

```

assert ( < < c, d > , g > ≡ < < y, y > , identity
y > ) as H3.

```

```

apply (traV _ _ _ (symV _ _ P21) P).

```

```

assert ( < c, d > ≡ < y, y > ) as H4.

```

```

apply (pairing_injective _ _ _ _ H3).

```

```

assert (c ≡ y) as H5.

```

```

apply (pairing_injective _ _ _ _ H4).

```

```

assert (d ≡ y) as H6.

```

```

apply (pairing_injective _ _ _ _ H4).

```

```

assert (g ≡ identity y) as H7.

```

```

apply (pairing_injective _ _ _ _ H3).

```

```

assert ( < < a, d > , compose_relV a b d f g > ≡
< < a, b > , f > ) as H2.

```

```

apply pairing_extensional.
apply pairing_extensional.
apply refV.
apply (traV _ _ _ H6 (traV _ _ _ (symV _ _ H5)
  (symV _ _ PB))).
apply ext3.
(* apply tt.
apply P12. *)
intros t Q.
assert (∃ x:V, ∃ z:V,
  t ≡ ⟨ x, z ⟩ ∧ x ∈ a ∧ z ∈ d ∧
  ∃ y:V, y ∈ b ∧ ⟨ x, y ⟩ ∈ f ∧ ⟨ y, z ⟩ ∈ g).
apply compose_relV_char_2.
apply P12.
apply (is_rel_extensional c d g b d g (symV _ _ PB)
(refV _) (refV _)).
apply P22.
apply Q.
destruct X as [x' [z' [PC [PD [PE PF]]]]].
destruct PF as [y' [PG [PH PI]]].
assert (⟨ y', z' ⟩ ∈ identity y) as H8.
apply (ext2 _ _ _ PI H7).
assert (y' ≡ z') as H9.
apply (identity_lemma _ _ y).
apply H8.
assert (⟨ x', y' ⟩ ≡ ⟨ x', z' ⟩) as HA.
apply pairing_extensional.
apply refV.
apply H9.
assert (⟨ x', y' ⟩ ≡ t) as HB.
apply (traV _ _ _ HA (symV _ _ PC)).
apply (ext1 _ _ _ (symV _ _ HB) PH).

intros t Q.
assert (∃ x':V, ∃ z':V,
  t ≡ ⟨ x', z' ⟩ ∧ x' ∈ a ∧ z' ∈ d ∧
  ∃ y':V, y' ∈ b ∧ ⟨ x', y' ⟩ ∈ f ∧ ⟨ y', z' ⟩ ∈ g)

```

```

as H8.
  assert ( $\exists x' : V, \exists z' : V, x' \in a \wedge z' \in b$ 
            $\wedge t \doteq \langle x', z' \rangle$ ) as X.
  apply (is_rel_pairs a b f t).
  apply P12.
  apply Q.
  destruct X as [x' [z' [PC [PD PE ]]]].
  exists x'.
  exists z'.
  split.
  apply PE.
  split.
  apply PC.
  split.
  assert (b  $\doteq$  d) as H8.
  apply (traV _ _ _ PB (traV _ _ _ H5 (symV _ _
H6))))).
  apply (ext2 _ _ _ PD H8).
  exists z'.
  split.
  apply PD.
  split.
  apply (ext1 _ _ _ (symV _ _ PE) Q).
  assert ( $\langle z', z' \rangle \in \text{identity } y$ ) as H8.
  apply identity_lemma_2.
  apply (ext2 _ _ _ PD (traV _ _ _ PB H5))).
  apply (ext2 _ _ _ H8 (symV _ _ H7))).
  apply compose_relV_char_2.
  apply P12.
  apply (is_rel_extensional c d g b d g).
  apply (symV _ _ PB).
  apply refV.
  apply refV.
  apply P22.
  apply H8.
  apply (traV _ _ _ H2 (symV _ _ P11))).
  apply H.

```

(* assoc *)

```
intros u u' u'' u'''.
destruct u as [x [arr1 [arr2 [PA PB]]]].
destruct arr1 as [v [c [d [g [P11 P12]]]]].
destruct arr2 as [w [i [j [h [P21 P22]]]]].
destruct u' as [x' [arr1' [arr2' [PA' PB']]]].
destruct arr1' as [v' [a [b [f [P11' P12']]]]].
destruct arr2' as [w' [c' [d' [g' [P21' P22']]]]].

destruct u'' as [x'' [arr1'' [arr2'' [PA''
PB''']]].
destruct arr1'' as [v'' [a' [b' [f' [P11''
P12''']]]].
destruct arr2'' as [w'' [c'' [j' [hg [P21''
P22''']]]].

destruct u''' as [x''' [arr1''' [arr2''' [PA'''
PB'''']]].
destruct arr1''' as [v''' [a''' [d''' [gf''
[P11''' P12'''']]]].
destruct arr2''' as [w''' [i''' [j''' [h' [P21'''
P22'''']]]].
simpl in PA.
simpl in PB.
simpl in PA'.
simpl in PB'.
simpl in PA''.
simpl in PB''.
simpl in PA'''.
simpl in PB'''.
intros H1 H2 H3.
simpl in H1.
simpl in H2.
simpl in H3.
intros H4 H5
```

```

intros H4 H5.
simpl in H4.
simpl in H5.

simpl.
assert ( < < a', j' > , compose_relV a' b' j' f' hg
> ≡ < < a''', j''' > , compose_relV a''' d''' j'''
gf'' h' > ) as G.
assert ( < < a, d' > , compose_relV a b d' f g' >
≡ v''' ) as H5'.
apply H5.
clear H5.
assert ( < < a, b > , f > ≡ < < a', b' > , f' > )
as E1.
apply (traV _ _ _ (symV _ _ P11') (traV _ _ _
(symV _ _ H1) P11'))).
assert ( < a, b > ≡ < a', b' > ) as E11.
apply (pairing_injective _ _ _ _ E1).
assert (a ≡ a') as E111.
apply (pairing_injective _ _ _ _ E11).
assert (b ≡ b') as E112.
apply (pairing_injective _ _ _ _ E11).
assert (f ≡ f') as E12.
apply (pairing_injective _ _ _ _ E1).
assert ( < < i, j > , h > ≡ < < i''', j''' > , h'
> ) as E2.
apply (traV _ _ _ (symV _ _ P21) (traV _ _ _ H3
P21''')).
assert ( < i, j > ≡ < i''', j''' > ) as E21.
apply (pairing_injective _ _ _ _ E2).
assert (i ≡ i''') as E211.
apply (pairing_injective _ _ _ _ E21).
assert (j ≡ j''') as E212.
apply (pairing_injective _ _ _ _ E21).
assert (h ≡ h') as E22.
apply (pairing_injective _ _ _ _ E2).
assert ( < < a, d' > , compose_relV a b d' f g' >
≡ < < a''', d''' > , gf'' > ) as E3.

```



```

apply (traV _ _ _ H5' P11'').
assert (⟨ a, d' ⟩ ≐ ⟨ a'', d'' ⟩) as E31.
apply (pairing_injective _ _ _ E3).
assert (a ≐ a'') as E311.
apply (pairing_injective _ _ _ E31).
assert (d' ≐ d'') as E312.
apply (pairing_injective _ _ _ E31).
assert (compose_relV a b d' f g' ≐ gf'') as E32.
apply (pairing_injective _ _ _ E3).

assert (⟨ ⟨ c, j ⟩, compose_relV c d j g h ⟩ ≐ ⟨
⟨ c'', j' ⟩, hg ⟩) as E4.
apply (traV _ _ _ (symV _ _ H4) P21'').
assert (⟨ c, j ⟩ ≐ ⟨ c'', j' ⟩) as E41.
apply (pairing_injective _ _ _ E4).
assert (c ≐ c'') as E411.
apply (pairing_injective _ _ _ E41).
assert (j ≐ j') as E412.
apply (pairing_injective _ _ _ E41).
assert (compose_relV c d j g h ≐ hg) as E42.
apply (pairing_injective _ _ _ E4).
assert (⟨ ⟨ c, d ⟩, g ⟩ ≐ ⟨ ⟨ c', d' ⟩, g' ⟩)
as E5.
apply (traV _ _ _ (symV _ _ P11) (traV _ _ _ (symV
_ _ H2) P21')).
assert (g ≐ g') as E52.
apply (pairing_injective _ _ _ E5).
assert (⟨ c, d ⟩ ≐ ⟨ c', d' ⟩) as E51.
apply (pairing_injective _ _ _ E5).
assert (c ≐ c') as E511.
apply (pairing_injective _ _ _ E51).
assert (d ≐ d') as E512.
apply (pairing_injective _ _ _ E51).

apply pairing_extensional.
apply pairing_extensional.
apply (traV _ _ _ (symV _ _ E111) E311).

```

```

apply (traV _ _ _ (symV _ _ E412) E212).

apply ext3.

intros t Q.
assert (
  (∃ x:V, ∃ z:V,
t ≡ ⟨ x, z ⟩ ∧ x ∈ a' ∧ z ∈ j' ∧
  ∃ y:V, y ∈ b' ∧ ⟨ x, y ⟩ ∈ f' ∧ ⟨ y, z ⟩ ∈ hg)
) as Q'.
apply compose_relV_char_2.
apply P12''.
assert (is_rel c'' j' hg) as Q''.
apply P22''.
apply (is_rel_extensional c'' j' hg b' j' hg
(symV _ _ PB'') (refV _) (refV _) Q'').
apply Q.
assert ((∃ x:V, ∃ z:V,
  t ≡ ⟨ x, z ⟩ ∧ x ∈ a''' ∧ z ∈ j''' ∧
  ∃ y:V, y ∈ d''' ∧ ⟨ x, y ⟩ ∈ gf'' ∧ ⟨ y, z ⟩ ∈
h')) as G.
destruct Q' as [x0 [z0 [R1 [R2 R3]]]].
exists x0.
exists z0.
split.
apply R1.
split.
apply (ext2 _ _ _ R2 (traV _ _ _ (symV _ _ E111)
E311)).
destruct R3 as [R4 R5].
split.
apply (ext2 _ _ _ R4 (traV _ _ _ (symV _ _ E412)
E212)).
destruct R5 as [y0 [R6 [R7 R8]]].
assert (⟨ y0, z0 ⟩ ∈ compose_relV c d j g h) as
R9.
apply (ext2 _ _ _ R8 (symV _ _ E42)).

```

```

    assert (  $y_0 \in c \wedge z_0 \in j \wedge$ 
       $\exists y:V, y \in d \wedge \langle y_0, y \rangle \in g \wedge \langle y, z_0 \rangle \in h$ ) as
RA.
  apply (compose_relV_char c d j g h).
  apply P12.
  assert (is_rel d j h) as RB.
  apply (is_rel_extensional i j h d j ).
  apply (symV _ _ PB).
  apply (refV _).
  apply (refV _).
  apply P22.
  apply RB.
  apply R9.
  destruct RA as [RB [RC [y1 RD]]].
  exists y1.
  split.
  assert (d = d'') as RE.
  apply (traV _ _ _ PB (traV _ _ _ E211 (symV _ _
PB'')))).
  apply (ext2 _ _ _ (fst RD) RE).
  split.
  assert (  $\langle x_0, y_1 \rangle \in \text{compose\_relV } a \ b \ d' \ f \ g'$ ) as
RF.
  assert (  $x_0 \in a \wedge y_1 \in d' \wedge$ 
     $\exists y:V, y \in b \wedge \langle x_0, y \rangle \in f \wedge \langle y, y_1 \rangle \in g'$ ) as
RG.
  split.
  apply (ext2 _ _ _ R2 (symV _ _ E111)).
  split.
  assert (d = d') as RH.
  apply (traV _ _ _ PB (traV _ _ _ E211
(traV _ _ _ (symV _ _ PB'')) (symV _ _ E312)))).
  apply (ext2 _ _ _ (fst RD) RH).
  exists y0.
  split.
  apply (ext2 _ _ _ R6 (symV _ _ E112)).
  split.

```

```

apply (ext2 _ _ _ R7 (symV _ _ E12)).
apply (ext2 _ _ _ (fst (snd RD)) E52).
apply compose_relV_char.

apply P12'.
assert (is_rel c' d' g') as RH.
apply P22'.
apply (is_rel_extensional c' d' g' b d' g').
apply (symV _ _ PB').
apply refV.
apply refV.
apply RH.
apply RG.
apply (ext2 _ _ _ RF E32).
apply (ext2 _ _ _ (snd (snd RD)) E22).
apply compose_relV_char_2.
apply P12'''.
assert (is_rel i''' j''' h') as RJ.
apply P22'''.
apply (is_rel_extensional i''' j''' h' d''' j'''
h').
apply (symV _ _ PB''').
apply refV.
apply refV.
apply RJ.
apply G.

intros t Q.
assert (∃ x:V, ∃ z:V,
  t ≡ ⟨ x, z ⟩ ∧ x ∈ a''' ∧ z ∈ j''' ∧
  ∃ y:V, y ∈ d''' ∧ ⟨ x, y ⟩ ∈ gf''' ∧ ⟨ y, z ⟩ ∈
h') as Q'.
apply compose_relV_char_2.
apply P12'''.
assert (is_rel i''' j''' h') as R2.
apply P22'''.
apply (is_rel_extensional i''' j''' h' d''' j'''
h').

```

```

apply (symV _ _ PB'').
apply refV.
apply refV.

apply R2.
apply Q.
assert (∃ x:V, ∃ z:V,
t ≡ ⟨ x, z ⟩ ∧ x ∈ a' ∧ z ∈ j' ∧
  ∃ y:V, y ∈ b' ∧ ⟨ x, y ⟩ ∈ f' ∧ ⟨ y, z ⟩ ∈ hg) as
G.
destruct Q' as [x0 [z0 [R4 [R5 [R6 R7]]]]].
destruct R7 as [y0 [R8 [R9 RA]]].
exists x0.
exists z0.
split.
apply R4.
split.
apply (ext2 _ _ _ R5).
apply (traV _ _ _ (symV _ _ E311) E111).
split.
apply (ext2 _ _ _ R6).
apply (traV _ _ _ (symV _ _ E212) E412).
assert (⟨ x0, y0 ⟩ ∈ compose_relV a b d' f g') as
RB.
apply (ext2 _ _ _ R9).
apply (symV _ _ E32).
assert (
x0 ∈ a ∧ y0 ∈ d' ∧
  ∃ y:V, y ∈ b ∧ ⟨ x0, y ⟩ ∈ f ∧ ⟨ y, y0 ⟩ ∈ g') as
RB'.
apply compose_relV_char.
apply P12'.
assert (is_rel c' d' g') as RC.
apply P22'.
apply (is_rel_extensional c' d' g' b d' g').
apply (symV _ _ PB').
apply refV.
apply refV.

```

```

    apply RC.
    apply RB.
    destruct RB' as [RC [RD [y1 [RE [RF RG]]]]].
    exists y1.
    split.
    apply (ext2 _ _ _ RE E112).
    split.
    apply (ext2 _ _ _ RF E12).
    assert (⟨ y1, z0 ⟩ ∈ compose_relV c d j g h) as
RH.
    assert (
y1 ∈ c ∧ z0 ∈ j ∧
    ∃y:V, y ∈ d ∧ ⟨ y1, y ⟩ ∈ g ∧ ⟨ y, z0 ⟩ ∈ h) as
RB'.
    split.
    apply (ext2 _ _ _ RE (traV _ _ _ PB' (symV _ _
E511))).
    split.
    apply (ext2 _ _ _ R6 (symV _ _ E212)).
    exists y0.
    split.
    apply (ext2 _ _ _ RD (symV _ _ E512)).
    split.
    apply (ext2 _ _ _ RG (symV _ _ E52)).
    apply (ext2 _ _ _ RA (symV _ _ E22)).
    apply compose_relV_char.
    apply P12.
    assert (is_rel i j h) as RJ.
    apply P22.
    apply (is_rel_extensional i j h d j h).
    apply (symV _ _ PB).
    apply refV.
    apply refV.
    apply RJ.
    apply RB'.
    apply (ext2 _ _ _ RH E42).

```

```

apply compose_relV_char_2.
apply P12''.
assert (is_rel c'' j' hg) as R3.

apply P22''.
apply (is_rel_extensional c'' j' hg b' j' hg).
apply (symV _ _ PB'').
apply refV.
apply refV.
apply R3.
apply G.
apply G.
Defined.

```

(* Another category is built from Rbar using
the following general construction. *)

(*

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

*)

(* Ad hoc definitions Big families *)

Notation " p^{-1} " := (Typeoidsym _ _ _ p) (at level 3,
no associativity).

Notation " $p \odot q$ " := (Typeoidtra _ _ _ _ q p) (at
level 34, right associativity).

Notation " $F \rightharpoonup p$ " := (Typeoidmapextensionality _ _ F
_ _ p) (at level 100).

Notation " $F \bullet p$ " := (Typeoidfamilymap _ F _ _ p)
(at level 9, right associativity).

Lemma Typeoidfamilyrefgeneral {A: Typeoid} (F:
Typeoidfamily A):

$\forall x: A, \forall p: x \approx x, \forall y: F x, F \bullet p y \approx y.$

Proof.

intros x p y.

apply Typeoidtra with (F•(Typeoidrefl A x) y);

eauto using Typeoidtra, Typeoidfamilyirr,

Typeoidfamilyref, Typeoidrefl.

Defined.

Lemma Typeoidfamilycmpgeneral {A: Typeoid} (F:
Typeoidfamily A):

$\forall x y z: A, \forall p: x \approx y, \forall q: y \approx z, \forall r: x \approx z,$
 $\forall w: F x, F \bullet q (F \bullet p w) \approx F \bullet r w.$

Proof.

intros x y z p q r w.

assert (F • q (F • p w) $\approx$ (F • (q $\odot$ p) w)) as H.

apply Typeoidfamilycmp.

eauto using Typeoidtra, Typeoidfamilyirr,

Typeoidfamilycmp.

Defined.

Lemma Typeoidfamilycmp3general {A: Typeoid} (F:
Typeoidfamily A):

$\forall x y z, \forall u: A, \forall p: x \approx y, \forall q: y \approx z, \forall s:$
 $z \approx u,$

$\forall r: x \approx u,$

$\forall w: F x, F \bullet s (F \bullet q (F \bullet p w)) \approx F \bullet r w.$

Proof.

intros x y z u p q s r w.

assert (F • s (F • q (F • p w)) $\approx$

F • (s $\odot$ q) (F • p w)) as H1.

apply Typeoidfamilycmp.


```

    assert ( F • (s ∘ q) (F • p w)  ≈≈≈ F • r w) as H2.
    apply Typeoidfamilycmpgeneral.
      eauto using Typeoidtra, Typeoidfamilyirr,
Typeoidfamilycmp.
Defined.

```

```

Lemma Typeoidfamilycmpinvert
{A: Typeoid} (F: Typeoidfamily 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 •
(Typeoidrefl A x) w) as H1.
  apply Typeoidfamilycmpgeneral.
  assert (F • (Typeoidrefl A x) w ≈≈≈,{ F x } w) as
H2.
  apply Typeoidfamilyrefgeneral.
  apply (Typeoidtra _ _ _ _ H1 H2).
Defined.

```

```

(* Here are some tactics for Typeoids
   similar to those for setoids *)

```

```

Hint Resolve Typeoidrefl : Swesetoid.
Hint Immediate Typeoidsym : Swesetoid.
Hint Resolve Typeoidtra : Swesetoid. (* The warning
here is expected *)

```

```

Hint Resolve Typeoidmapextensionality : swesetoid.

```

```

Ltac Swesetoid := simpl; eauto with Swesetoid.

```

(* Some lemmas about proof-irrelevant families *)

Lemma Typeoidfamilyirrgeneral

```
{A: Typeoid} (F: Typeoidfamily 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) as H1.  
apply Typeoidfamilyirr.  
assert (F • q u ≈≈,{ F y } F • q v) as H2.  
apply Typeoidmapextensionality.  
apply H.  
apply (Typeoidtra _ _ _ _ H1 H2).
```

Defined.

Lemma Typeoidfamilyirrgeneraldouble

```
{A: Typeoid} (F: Typeoidfamily 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 Typeoidfamilycmp.  
assert (F • q' (F • q v) ≈≈ F • (q'⊙ q) v).  
apply Typeoidfamilycmp.  
assert (F • (p'⊙ p) u ≈≈ F • (q'⊙ q) v ).  
apply Typeoidfamilyirrgeneral.
```

exact H.

Swesetoid.

Defined.

Lemma Typeoidfamilycmpgeneral_3

```
{A: Typeoid} (F: Typeoidfamily 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 Typeoidfamilycmp.
Swesetoid.
```

Defined.

Lemma Typeoidfamilycmptactical

```
{A: Typeoid} (F: Typeoidfamily A):
  ∀ x y z: A,
  ∀ p: x ≈≈≈ z,  ∀ p': z ≈≈≈ y,
  ∀u: F x,  ∀v: F y,
    F • (p'⊙ p) u ≈≈≈ v -> F•p' (F•p u) ≈≈≈ v.
intros x y z p p' u v H.
assert (F•p' (F•p u) ≈≈≈ F • (p' ⊙ p) u) as H2.
apply Typeoidfamilycmp.
Swesetoid.
```

Defined.

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

```
Definition arr_from_family {A:Typeoid} (F:
Typeoidfamily A): Typeoid.
  apply
    (Build_Typeoid (∃a: A, (∃b: A, Typeoidmap (F a) (F
b))))
    (λ p q, match (p, q) with
      (existT a (existT b f), existT a' (existT b' f'))
=>
```

$$\exists e: a \approx\approx a', \quad \exists d: b \approx\approx b', \quad \forall x: (F a), \\ F \bullet d (f x) \approx\approx f' (F \bullet e x) \\ \text{end})) .$$

```

intros [a [b f]].
exists (Typeoidrefl _ _).
exists (Typeoidrefl _ _).
intro x.
assert ( f x  $\approx\approx$ , { F b } f (F • (Typeoidrefl A a)
x))).
apply Typeoidmapextensionality.
apply Typeoidsym.
apply Typeoidfamilyrefgeneral.
assert (F • (Typeoidrefl A b) (f x)  $\approx\approx$ , { F b } (f
x))).
apply Typeoidfamilyrefgeneral.
Swesetoid.
intros [a [b f]].
intros [a' [b' f']].
intros [e [d H2]].
exists e-1.
exists d-1.
intro x.
apply Typeoidsym.
assert (F • d (f (F • e-1 x))  $\approx\approx$ , { F b' } f'
(F • e (F • e-1 x))).
apply H2.
assert (F • d-1 (F • d (f (F • e-1 x)))  $\approx\approx$ , { F b
}
F • d-1 (f' (F • e (F • e-1
x))))).
apply Typeoidmapextensionality.
assumption.
assert (F • d-1 (F • d (f (F • e-1 x)))  $\approx\approx$ , { F b
} f (F • e-1 x)).
assert (F • d-1 (F • d (f (F • e-1 x)))  $\approx\approx$ , { F b
} F • (Typeoidrefl A b) (f (F • e-1 x))).
apply Typeoidfamilyvcmaeneral.

```

```

    assert (F • (Typeoidrefl A b) (f (F • e-1 x))) ≈≈≈,
{ F b } f (F • e-1 x)).
  apply Typeoidfamilyref.

```

```

  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 Typeoidmapextensionality.

```

```

  apply Typeoidmapextensionality.

```

```

  assert (F • e (F • e-1 x) ≈≈≈, { F a' } F •
(Typeoidrefl A a') x).

```

```

  apply Typeoidfamilycmpgeneral.

```

```

  assert (F • (Typeoidrefl A a') x ≈≈≈, { F a' } x).

```

```

  apply Typeoidfamilyref.

```

```

  Swesetoid. Swesetoid.

```

```

  intros [a [b f]].

```

```

  intros [a' [b' f']].

```

```

  intros [a'' [b'' f'']].

```

```

  intros [e [d H]].

```

```

  intros [e' [d' H']].

```

```

  exists (e' ◦ e).

```

```

  exists (d' ◦ d).

```

```

  intro x.

```

```

  apply Typeoidtra with (F • d' ((F • d) (f x))).

```

```

  apply Typeoidsym.

```

```

  apply Typeoidfamilycmp.

```

```

  apply Typeoidtra with (f'' (F • e' ((F • e) x))).

```

```

  specialize (H' ((F • e) x)).

```

```

  specialize (H x).

```

```

  assert (F • d' (f' (F • e x)) ≈≈≈ F • d' (F • d (f
x))) as H2.

```

```

  apply Typeoidmapextensionality.

```

```

  apply (Typeoidsym H).

```

```

    apply (Typeoidtra _ _ _ (Typeoidsym _ _ _ H2)
H').

```

```

    apply Typeoidmapextensionality.
    apply Typeoidfamilycmp.
Defined.

```

(* The id-map on a Typeoid *)

```

Definition Idmapp (A: Typeoid): Typeoidmap A A.
  apply (Build_Typeoidmap A A (λ x: A, x));
Swesetoid.
Defined.

```

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

```

Definition catid_from_family_helper {A:Typeoid}
(F: Typeoidfamily A)(a: A) : arr_from_family F.
  exists a.
  exists a.
  apply (Idmapp (F a)).
Defined.

```

(* The id creating map in the category *)

```

Definition catid_from_family
{A:Typeoid} (F: Typeoidfamily A):
  Typeoidmap A (arr_from_family F).
  apply (Build_Typeoidmap 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.

```

Defined.

(* The domain map in the category *)

Definition catdom_from_family_helper

{A:Typeoid}(F: Typeoidfamily A)(f:arr_from_family F):
A.

destruct f as [a s].

exact a.

Defined.

Definition catdom_from_family

{A:Typeoid} (F: Typeoidfamily A):

Typeoidmap (arr_from_family F) A.

apply (Build_Typeoidmap (arr_from_family F) A
(λ a, catdom_from_family_helper F a)).

intros [a s] [b t] H.

destruct s.

destruct t.

destruct H.

Swesetoid.

Defined.

(* The codomain map in the category *)

Definition catcod_from_family_helper

{A:Typeoid} (F: Typeoidfamily A)(f:arr_from_family
F): A.

destruct f as [a [b t]].

exact b.

Defined.

Definition catcod_from_family

{A:Typeoid} (F: Typeoidfamily A):

Typeoidmap (arr_from_family F) A.

apply (Build_Typeoidmap (arr_from_family F) A
(λ a, catcod_from_family_helper F a)).

intros [a <| Γh +1] H

```

intros [s] [t] [H] ..
destruct s.
destruct t.
destruct H as [p [q H]].

```

Swesetoid.

Defined.

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

$(g \ f) : \text{s.t. } \text{cod } g = \text{dom } f$

*)

Definition cms_from_family

```

{A:Typeoid} (F: Typeoidfamily A): Typeoid.
apply (Build_Typeoid (∃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))).

```

```

intros [g [f H]].
split.
apply Typeoidrefl.
apply Typeoidrefl.

```

```

intros [g [f H]].
intros [g1 [f1 H1]].
intros [H21 H22].
split.
apply Typeoidsym. assumption.
apply Typeoidsym. assumption.

```



```

intros [g1 [f1 H1]].
intros [g2 [f2 H2]].

intros [g3 [f3 H3]].
intros [H41 H42].
intros [H51 H52].
split.
apply Typeoidtra with g2. assumption. assumption.
apply Typeoidtra with f2. assumption. assumption.
Defined.

```

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

Definition catfst_from_family_helper

```

{A:Typeoid} (F: Typeoidfamily A)(u: cms_from_family
F):
  arr_from_family F.
destruct u as [g [f u]].
exact g.
Defined.

```

Definition catfst_from_family

```

{A:Typeoid} (F: Typeoidfamily A):
  Typeoidmap (cms_from_family F)
              (arr_from_family F).
apply (Build_Typeoidmap
      (cms_from_family F)
      (arr_from_family F)
      (λ u, catfst_from_family_helper F u)).
intros [g [f u]] [g' [f' v]] [H1 H2].
exact H1.
Defined.

```

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

Definition catsnd_from_family_helper

{A:Typeoid} (F: Typeoidfamily A)(u: cms_from_family F):

arr_from_family F.
destruct u as [g [f u]].
exact f.

Defined.

Definition catsnd_from_family

{A:Typeoid} (F: Typeoidfamily A):
Typeoidmap (cms_from_family F)
(arr_from_family F).
apply (Build_Typeoidmap
(cms_from_family F)
(arr_from_family F)
(λ u, catsnd_from_family_helper F u)).
intros [g [f u]] [g' [f' v]] [H1 H2].
exact H2.

Defined.

(* Composition for Typeoidmaps *)

Definition Typeoidmaps (A B: Typeoid): Typeoid.

apply (Build_Typeoid (Typeoidmap A B) (λ f g, ∀x:A, f x ≈≈≈ g x)); Swesetoid.

Defined.

Lemma bintypeoidmap_helper {A B C: Typeoid} (f:A → B
→ C)

(p: ∀a a', a ≈≈≈ a' → ∀b, f a b ≈≈≈ f a' b)
(q: ∀a b b', b ≈≈≈ b' → f a b ≈≈≈ f a b')
: (Typeoidmap A (Typeoidmaps B C)).

Proof.

apply (Build_Typeoidmap A (Typeoidmaps B C)
(λ a, (Build_Typeoidmap B C (f a) (q a)))) p).

Defined.

```

Lemma trintypeoidmap_helper {A B C D: Typeoid} (f: A
→ B → C → D)
  (p: ∀a a', a ≈≈≈ a' → ∀b c, f a b c ≈≈≈ f a' b c)
  (q: ∀a b b', b ≈≈≈ b' → ∀c, f a b c ≈≈≈ f a b' c)
  (r: ∀a b c c', c ≈≈≈ c' → f a b c ≈≈≈ f a b c')
  : (Typeoidmap A (Typeoidmaps B (Typeoidmaps C D))).

```

Proof.

```

  apply (Build_Typeoidmap A (Typeoidmaps B
(Typeoidmaps C D))) with (λ a, bintypeoidmap_helper
(f a) (q a) (r a));

```

assumption.

Defined.

```

Definition Comp {A B: Typeoid} (C: Typeoid):
Typeoidmap (Typeoidmaps B C)
  (Typeoidmaps (Typeoidmaps A B)
    (Typeoidmaps A C)).

```

```

apply trintypeoidmap_helper with (λ f: (Typeoidmap B
C), λ g: (Typeoidmap A B), λ a, f (g a)).

```

Swesetoid.

intros.

apply Typeoidmapextensionality.

Swesetoid.

intros.

apply Typeoidmapextensionality.

apply Typeoidmapextensionality.

Swesetoid.

Defined.

(* The composition map in the category *)

```

Definition catcmp_from_family_helper
  {A:Typeoid} (F: Typeoidfamily A) (u:
cms_from_family F):

```

```

  arr_from_family F.

```

```

  destruct u as [f1 f2 f3 f4 f5 f6 f7 f8 f9 f10 f11 f12 f13 f14 f15 f16 f17 f18 f19 f20 f21 f22 f23 f24 f25 f26 f27 f28 f29 f30 f31 f32 f33 f34 f35 f36 f37 f38 f39 f40 f41 f42 f43 f44 f45 f46 f47 f48 f49 f50 f51 f52 f53 f54 f55 f56 f57 f58 f59 f60 f61 f62 f63 f64 f65 f66 f67 f68 f69 f70 f71 f72 f73 f74 f75 f76 f77 f78 f79 f80 f81 f82 f83 f84 f85 f86 f87 f88 f89 f90 f91 f92 f93 f94 f95 f96 f97 f98 f99 f100 f101 f102 f103 f104 f105 f106 f107 f108 f109 f110 f111 f112 f113 f114 f115 f116 f117 f118 f119 f120 f121 f122 f123 f124 f125 f126 f127 f128 f129 f130 f131 f132 f133 f134 f135 f136 f137 f138 f139 f140 f141 f142 f143 f144 f145 f146 f147 f148 f149 f150 f151 f152 f153 f154 f155 f156 f157 f158 f159 f160 f161 f162 f163 f164 f165 f166 f167 f168 f169 f170 f171 f172 f173 f174 f175 f176 f177 f178 f179 f180 f181 f182 f183 f184 f185 f186 f187 f188 f189 f190 f191 f192 f193 f194 f195 f196 f197 f198 f199 f200 f201 f202 f203 f204 f205 f206 f207 f208 f209 f210 f211 f212 f213 f214 f215 f216 f217 f218 f219 f220 f221 f222 f223 f224 f225 f226 f227 f228 f229 f230 f231 f232 f233 f234 f235 f236 f237 f238 f239 f240 f241 f242 f243 f244 f245 f246 f247 f248 f249 f250 f251 f252 f253 f254 f255 f256 f257 f258 f259 f260 f261 f262 f263 f264 f265 f266 f267 f268 f269 f270 f271 f272 f273 f274 f275 f276 f277 f278 f279 f280 f281 f282 f283 f284 f285 f286 f287 f288 f289 f290 f291 f292 f293 f294 f295 f296 f297 f298 f299 f300 f301 f302 f303 f304 f305 f306 f307 f308 f309 f310 f311 f312 f313 f314 f315 f316 f317 f318 f319 f320 f321 f322 f323 f324 f325 f326 f327 f328 f329 f330 f331 f332 f333 f334 f335 f336 f337 f338 f339 f340 f341 f342 f343 f344 f345 f346 f347 f348 f349 f350 f351 f352 f353 f354 f355 f356 f357 f358 f359 f360 f361 f362 f363 f364 f365 f366 f367 f368 f369 f370 f371 f372 f373 f374 f375 f376 f377 f378 f379 f380 f381 f382 f383 f384 f385 f386 f387 f388 f389 f390 f391 f392 f393 f394 f395 f396 f397 f398 f399 f400 f401 f402 f403 f404 f405 f406 f407 f408 f409 f410 f411 f412 f413 f414 f415 f416 f417 f418 f419 f420 f421 f422 f423 f424 f425 f426 f427 f428 f429 f430 f431 f432 f433 f434 f435 f436 f437 f438 f439 f440 f441 f442 f443 f444 f445 f446 f447 f448 f449 f450 f451 f452 f453 f454 f455 f456 f457 f458 f459 f460 f461 f462 f463 f464 f465 f466 f467 f468 f469 f470 f471 f472 f473 f474 f475 f476 f477 f478 f479 f480 f481 f482 f483 f484 f485 f486 f487 f488 f489 f490 f491 f492 f493 f494 f495 f496 f497 f498 f499 f500 f501 f502 f503 f504 f505 f506 f507 f508 f509 f510 f511 f512 f513 f514 f515 f516 f517 f518 f519 f520 f521 f522 f523 f524 f525 f526 f527 f528 f529 f530 f531 f532 f533 f534 f535 f536 f537 f538 f539 f540 f541 f542 f543 f544 f545 f546 f547 f548 f549 f550 f551 f552 f553 f554 f555 f556 f557 f558 f559 f560 f561 f562 f563 f564 f565 f566 f567 f568 f569 f570 f571 f572 f573 f574 f575 f576 f577 f578 f579 f580 f581 f582 f583 f584 f585 f586 f587 f588 f589 f590 f591 f592 f593 f594 f595 f596 f597 f598 f599 f600 f601 f602 f603 f604 f605 f606 f607 f608 f609 f610 f611 f612 f613 f614 f615 f616 f617 f618 f619 f620 f621 f622 f623 f624 f625 f626 f627 f628 f629 f630 f631 f632 f633 f634 f635 f636 f637 f638 f639 f640 f641 f642 f643 f644 f645 f646 f647 f648 f649 f650 f651 f652 f653 f654 f655 f656 f657 f658 f659 f660 f661 f662 f663 f664 f665 f666 f667 f668 f669 f670 f671 f672 f673 f674 f675 f676 f677 f678 f679 f680 f681 f682 f683 f684 f685 f686 f687 f688 f689 f690 f691 f692 f693 f694 f695 f696 f697 f698 f699 f700 f701 f702 f703 f704 f705 f706 f707 f708 f709 f710 f711 f712 f713 f714 f715 f716 f717 f718 f719 f720 f721 f722 f723 f724 f725 f726 f727 f728 f729 f730 f731 f732 f733 f734 f735 f736 f737 f738 f739 f740 f741 f742 f743 f744 f745 f746 f747 f748 f749 f750 f751 f752 f753 f754 f755 f756 f757 f758 f759 f760 f761 f762 f763 f764 f765 f766 f767 f768 f769 f770 f771 f772 f773 f774 f775 f776 f777 f778 f779 f780 f781 f782 f783 f784 f785 f786 f787 f788 f789 f790 f791 f792 f793 f794 f795 f796 f797 f798 f799 f800 f801 f802 f803 f804 f805 f806 f807 f808 f809 f810 f811 f812 f813 f814 f815 f816 f817 f818 f819 f820 f821 f822 f823 f824 f825 f826 f827 f828 f829 f830 f831 f832 f833 f834 f835 f836 f837 f838 f839 f840 f841 f842 f843 f844 f845 f846 f847 f848 f849 f850 f851 f852 f853 f854 f855 f856 f857 f858 f859 f860 f861 f862 f863 f864 f865 f866 f867 f868 f869 f870 f871 f872 f873 f874 f875 f876 f877 f878 f879 f880 f881 f882 f883 f884 f885 f886 f887 f888 f889 f890 f891 f892 f893 f894 f895 f896 f897 f898 f899 f900 f901 f902 f903 f904 f905 f906 f907 f908 f909 f910 f911 f912 f913 f914 f915 f916 f917 f918 f919 f920 f921 f922 f923 f924 f925 f926 f927 f928 f929 f930 f931 f932 f933 f934 f935 f936 f937 f938 f939 f940 f941 f942 f943 f944 f945 f946 f947 f948 f949 f950 f951 f952 f953 f954 f955 f956 f957 f958 f959 f960 f961 f962 f963 f964 f965 f966 f967 f968 f969 f970 f971 f972 f973 f974 f975 f976 f977 f978 f979 f980 f981 f982 f983 f984 f985 f986 f987 f988 f989 f990 f991 f992 f993 f994 f995 f996 f997 f998 f999 f1000 f1001 f1002 f1003 f1004 f1005 f1006 f1007 f1008 f1009 f1010 f1011 f1012 f1013 f1014 f1015 f1016 f1017 f1018 f1019 f1020 f1021 f1022 f1023 f1024 f1025 f1026 f1027 f1028 f1029 f1030 f1031 f1032 f1033 f1034 f1035 f1036 f1037 f1038 f1039 f1040 f1041 f1042 f1043 f1044 f1045 f1046 f1047 f1048 f1049 f1050 f1051 f1052 f1053 f1054 f1055 f1056 f1057 f1058 f1059 f1060 f1061 f1062 f1063 f1064 f1065 f1066 f1067 f1068 f1069 f1070 f1071 f1072 f1073 f1074 f1075 f1076 f1077 f1078 f1079 f1080 f1081 f1082 f1083 f1084 f1085 f1086 f1087 f1088 f1089 f1090 f1091 f1092 f1093 f1094 f1095 f1096 f1097 f1098 f1099 f1100 f1101 f1102 f1103 f1104 f1105 f1106 f1107 f1108 f1109 f1110 f1111 f1112 f1113 f1114 f1115 f1116 f1117 f1118 f1119 f1120 f1121 f1122 f1123 f1124 f1125 f1126 f1127 f1128 f1129 f1130 f1131 f1132 f1133 f1134 f1135 f1136 f1137 f1138 f1139 f1140 f1141 f1142 f1143 f1144 f1145 f1146 f1147 f1148 f1149 f1150 f1151 f1152 f1153 f1154 f1155 f1156 f1157 f1158 f1159 f1160 f1161 f1162 f1163 f1164 f1165 f1166 f1167 f1168 f1169 f1170 f1171 f1172 f1173 f1174 f1175 f1176 f1177 f1178 f1179 f1180 f1181 f1182 f1183 f1184 f1185 f1186 f1187 f1188 f1189 f1190 f1191 f1192 f1193 f1194 f1195 f1196 f1197 f1198 f1199 f1200 f1201 f1202 f1203 f1204 f1205 f1206 f1207 f1208 f1209 f1210 f1211 f1212 f1213 f1214 f1215 f1216 f1217 f1218 f1219 f1220 f1221 f1222 f1223 f1224 f1225 f1226 f1227 f1228 f1229 f1230 f1231 f1232 f1233 f1234 f1235 f1236 f1237 f1238 f1239 f1240 f1241 f1242 f1243 f1244 f1245 f1246 f1247 f1248 f1249 f1250 f1251 f1252 f1253 f1254 f1255 f1256 f1257 f1258 f1259 f1260 f1261 f1262 f1263 f1264 f1265 f1266 f1267 f1268 f1269 f1270 f1271 f1272 f1273 f1274 f1275 f1276 f1277 f1278 f1279 f1280 f1281 f1282 f1283 f1284 f1285 f1286 f1287 f1288 f1289 f1290 f1291 f1292 f1293 f1294 f1295 f1296 f1297 f1298 f1299 f1300 f1301 f1302 f1303 f1304 f1305 f1306 f1307 f1308 f1309 f1310 f1311 f1312 f1313 f1314 f1315 f1316 f1317 f1318 f1319 f1320 f1321 f1322 f1323 f1324 f1325 f1326 f1327 f1328 f1329 f1330 f1331 f1332 f1333 f1334 f1335 f1336 f1337 f1338 f1339 f1340 f1341 f1342 f1343 f1344 f1345 f1346 f1347 f1348 f1349 f1350 f1351 f1352 f1353 f1354 f1355 f1356 f1357 f1358 f1359 f1360 f1361 f1362 f1363 f1364 f1365 f1366 f1367 f1368 f1369 f1370 f1371 f1372 f1373 f1374 f1375 f1376 f1377 f1378 f1379 f1380 f1381 f1382 f1383 f1384 f1385 f1386 f1387 f1388 f1389 f1390 f1391 f1392 f1393 f1394 f1395 f1396 f1397 f1398 f1399 f1400 f1401 f1402 f1403 f1404 f1405 f1406 f1407 f1408 f1409 f1410 f1411 f1412 f1413 f1414 f1415 f1416 f1417 f1418 f1419 f1420 f1421 f1422 f1423 f1424 f1425 f1426 f1427 f1428 f1429 f1430 f1431 f1432 f1433 f1434 f1435 f1436 f1437 f1438 f1439 f1440 f1441 f1442 f1443 f1444 f1445 f1446 f1447 f1448 f1449 f1450 f1451 f1452 f1453 f1454 f1455 f1456 f1457 f1458 f1459 f1460 f1461 f1462 f1463 f1464 f1465 f1466 f1467 f1468 f1469 f1470 f1471 f1472 f1473 f1474 f1475 f1476 f1477 f1478 f1479 f1480 f1481 f1482 f1483 f1484 f1485 f1486 f1487 f1488 f1489 f1490 f1491 f1492 f1493 f1494 f1495 f1496 f1497 f1498 f1499 f1500 f1501 f1502 f1503 f1504 f1505 f1506 f1507 f1508 f1509 f1510 f1511 f1512 f1513 f1514 f1515 f1516 f1517 f1518 f1519 f1520 f1521 f1522 f1523 f1524 f1525 f1526 f1527 f1528 f1529 f1530 f1531 f1532 f1533 f1534 f1535 f1536 f1537 f1538 f1539 f1540 f1541 f1542 f1543 f1544 f1545 f1546 f1547 f1548 f1549 f1550 f1551 f1552 f1553 f1554 f1555 f1556 f1557 f1558 f1559 f1560 f1561 f1562 f1563 f1564 f1565 f1566 f1567 f1568 f1569 f1570 f1571 f1572 f1573 f1574 f1575 f1576 f1577 f1578 f1579 f1580 f1581 f1582 f1583 f1584 f1585 f1586 f1587 f1588 f1589 f1590 f1591 f1592 f1593 f1594 f1595 f1596 f1597 f1598 f1599 f1600 f1601 f1602 f1603 f1604 f1605 f1606 f1607 f1608 f1609 f1610 f1611 f1612 f1613 f1614 f1615 f1616 f1617 f1618 f1619 f1620 f1621 f1622 f1623 f1624 f1625 f1626 f1627 f1628 f1629 f1630 f1631 f1632 f1633 f1634 f1635 f1636 f1637 f1638 f1639 f1640 f1641 f1642 f1643 f1644 f1645 f1646 f1647 f1648 f1649 f1650 f1651 f1652 f1653 f1654 f1655 f1656 f1657 f1658 f1659 f1660 f1661 f1662 f1663 f1664 f1665 f1666 f1667 f1668 f1669 f1670 f1671 f1672 f1673 f1674 f1675 f1676 f1677 f1678 f1679 f1680 f1681 f1682 f1683 f1684 f1685 f1686 f1687 f1688 f1689 f1690 f1691 f1692 f1693 f1694 f1695 f1696 f1697 f1698 f1699 f1700 f1701 f1702 f1703 f1704 f1705 f1706 f1707 f1708 f1709 f1710 f1711 f1712 f1713 f1714 f1715 f1716 f1717 f1718 f1719 f1720 f1721 f1722 f1723 f1724 f1725 f1726 f1727 f1728 f1729 f1730 f1731 f1732 f1733 f1734 f1735 f1736 f1737 f1738 f1739 f1740 f1741 f1742 f1743 f1744 f1745 f1746 f1747 f1748 f1749 f1750 f1751 f1752 f1753 f1754 f1755 f1756 f1757 f1758 f1759 f1760 f1761 f1762 f1763 f1764 f1765 f1766 f1767 f1768 f1769 f1770 f1771 f1772 f1773 f1774 f1775 f1776 f1777 f1778 f1779 f1780 f1781 f1782 f1783 f1784 f1785 f1786 f1787 f1788 f1789 f1790 f1791 f1792 f1793 f1794 f1795 f1796 f1797 f1798 f1799 f1800 f1801 f1802 f1803 f1804 f1805 f1806 f1807 f1808 f1809 f1810 f1811 f1812 f1813 f1814 f1815 f1816 f1817 f1818 f1819 f1820 f1821 f1822 f1823 f1824 f1825 f1826 f1827 f1828 f1829 f1830 f1831 f1832 f1833 f1834 f1835 f1836 f1837 f1838 f1839 f1840 f1841 f1842 f1843 f1844 f1845 f1846 f1847 f1848 f1849 f1850 f1851 f1852 f1853 f1854 f1855 f1856 f1857 f1858 f1859 f1860 f1861 f1862 f1863 f1864 f1865 f1866 f1867 f1868 f1869 f1870 f1871 f1872 f1873 f1874 f1875 f1876 f1877 f1878 f1879 f1880 f1881 f1882 f1883 f1884 f1885 f1886 f1887 f1888 f1889 f1890 f1891 f1892 f1893 f1894 f1895 f1896 f1897 f1898 f1899 f1900 f1901 f1902 f1903 f1904 f1905 f1906 f1907 f1908 f1909 f1910 f1911 f1912 f1913 f1914 f1915 f1916 f1917 f1918 f1919 f1920 f1921 f1922 f1923 f1924 f1925 f1926 f1927 f1928 f1929 f1930 f1931 f1932 f1933 f1934 f1935 f1936 f1937 f1938 f1939 f1940 f1941 f1942 f1943 f1944 f1945 f1946 f1947 f1948 f1949 f1950 f1951 f1952 f1953 f1954 f1955 f1956 f1957 f1958 f1959 f1960 f1961 f1962 f1963 f1964 f1965 f1966 f1967 f1968 f1969 f1970 f1971 f1972 f1973 f1974 f1975 f1976 f1977 f1978 f1979 f1980 f1981 f1982 f1983 f1984 f1985 f1986 f1987 f1988 f1989 f1990 f1991 f1992 f1993 f1994 f1995 f1996 f1997 f1998 f1999 f2000 f2001 f2002 f2003 f2004 f2005 f2006 f2007 f2008 f2009 f2010 f2011 f2012 f2013 f2014 f2015 f2016 f2017 f2018 f2019 f2020 f2021 f2022 f2023 f2024 f2025 f2026 f2027 f2028 f2029 f2030 f2031 f2032 f2033 f2034 f2035 f2036 f2037 f2038 f2039 f2040 f2041 f2042 f2043 f2044 f2045 f2046 f2047 f2048 f2049 f2050 f2051 f2052 f2053 f2054 f2055 f2056 f2057 f2058 f2059 f2060 f2061 f2062 f2063 f2064 f2065 f2066 f2067 f2068 f2069 f2070 f2071 f2072 f2073 f2074 f2075 f2076 f2077 f2078 f2079 f2080 f2081 f2082 f2083 f2084 f2085 f2086 f2087 f2088 f2089 f2090 f2091 f2092 f2093 f2094 f2095 f2096 f2097 f2098 f2099 f2100 f2101 f2102 f2103 f2104 f2105 f2106 f2107 f2108 f2109 f2110 f2111 f2112 f2113 f2114 f2115 f2116 f2117 f2118 f2119 f2120 f2121 f2122 f2123 f2124 f2125 f2126 f2127 f2128 f2129 f2130 f2131 f2132 f2133 f2134 f2135 f2136 f2137 f2138 f2139 f2140 f2141 f2142 f2143 f2144 f2145 f2146 f2147 f2148 f2149 f2150 f2151 f2152 f2153 f2154 f2155 f2156 f2157 f2158 f2159 f2160 f2161 f2162 f2163 f2164 f2165 f2166 f2167 f2168 f2169 f2170 f2171 f2172 f2173 f2174 f2175 f2176 f2177 f2178 f2179 f2180 f2181 f2182 f2183 f2184 f2185 f2186 f2187 f2188 f2189 f2190 f2191 f2192 f2193 f2194 f2195 f2196 f2197 f2198 f2199 f2200 f2201 f2202 f2203 f2204 f2205 f2206 f2207 f2208 f2209 f2210 f2211 f2212 f2213 f2214 f2215 f2216 f2217 f2218 f2219 f2220 f2221 f2222 f2223 f2224 f2225 f2226 f2227 f2228 f2229 f2230 f2231 f2232 f2233 f2234 f2235 f2236 f2237 f2238 f2239 f2240 f2241 f2242 f2243 f2244 f2245 f2246 f2247 f2248 f2249 f2250 f2251 f2252 f2253 f2254 f2255 f2256 f2257 f2258 f2259 f2260 f2261 f2262 f2263 f2264 f2265 f2266 f2267 f2268 f2269 f2270 f2271 f2272 f2273 f2274 f2275 f2276 f2277 f2278 f2279 f2280 f2281 f2282 f2283 f2284 f2285 f2286 f2287 f2288 f2289 f2290 f2291 f2292 f2293 f2294 f2295 f2296 f2297 f2298 f2299 f2300 f2301 f2302 f2303 f2304 f2305 f2306 f2307 f2308 f2309 f2310 f2311 f2312 f2313 f2314 f2315 f2316 f2317 f2318 f2319 f2320 f2321 f2322 f2323 f2324 f2325 f2326 f2327 f2328 f2329 f2330 f2331 f2332 f2333 f2334 f2335 f2336 f2337 f2338 f2339 f2340 f2341 f2342 f2343 f2344 f2345 f2346 f2347 f2348 f2349 f2350 f2351 f2352 f2353 f2354 f2355 f2356 f2357 f2358 f2359 f2360 f2361 f2362 f2363 f2364 f2365 f2366 f2367 f2368 f2369 f2370 f2371 f2372 f2373 f2374 f2375 f2376 f2377 f2378 f2379 f2380 f2381 f2382 f2383 f2384 f2385 f2386 f2387 f2388 f2389 f2390 f2391 f2392 f2393 f2394 f2395 f2396 f2397 f2398 f2399 f2400 f2401 f2402 f2403 f2404 f2405 f2406 f2407 f2408 f2409 f2410 f2411 f2412 f2413 f2414 f2415 f2416 f2417 f2418 f2419 f2420 f2421 f2422 f2423 f2424 f2425 f2426 f2427 f2428 f2429 f2430 f2431 f2432 f2433 f2434 f2435 f2436 f2437 f2438 f2439 f2440 f2441 f2442 f2443 f2444 f2445 f2446 f2447 f2448 f2449 f2450 f2451 f2452 f2453 f2454 f2455 f2456 f2457 f2458 f2459 f2460 f2461 f2462 f2463 f2464 f2465 f2466 f2467 f2468 f2469 f2470 f2471 f2472 f2473 f2474 f2475 f2476 f2477 f2
```

```

destruct u as [[a [b g]] [[c [d f]] p]].
exists a.
exists d.
apply (Comp _ f (Comp _ (F•p) g)).

```

Defined.

Definition catcmp_from_family

```

{A:Typeoid} (F: Typeoidfamily A):
  Typeoidmap (cms_from_family F)
              (arr_from_family F).
apply (Build_Typeoidmap
       (cms_from_family F)
       (arr_from_family F)
       (λ u, catcmp_from_family_helper F u)).
intros [[c [d g]] [[a [b f]] p]]
       [[c'[d' g']] [[a' [b' f']] p']].
simpl.
intros [[q [r H1]] [s [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 Typeoidmapextensionality.
apply Typeoidmapextensionality.
apply H1.
assert (f' (F • s (F • p (g x))) ≈≈≈, { F b' }
        f' (F • p' (F • r (g x))) ).
apply Typeoidmapextensionality.
assert ((F • s (F • p (g x))) ≈≈≈, { F a' } F •
(s•p) (g x)).
apply Typeoidfamilycmp.
assert ((F • p' (F • r (g x))) ≈≈≈, { F a' } F •
(p'•r) (g x)).
apply Typeoidfamilycmp

```

```

    apply typeoidfamilycmp.
    assert (F • (p'⊙r) (g x) ≈≈≈, { F a' } F • (s⊙p) (g
x)).
    apply Typeoidfamilyirr.

```

Swesetoid.

Swesetoid.

Defined.

Definition cms_from_family_builder

```

  {A:Typeoid} (F: Typeoidfamily 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:Typeoid} (F: Typeoidfamily 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 Typeoidrefl.

(* K2 *)
intro x. apply Typeoidrefl.

(* K3 *)
intros [g [f p]].
destruct g as [c [d g]].
destruct f as [a [b f]].
simpl.
apply Typeoidrefl.

(* K4 *)
intros [g [f p]].
destruct g as [c [d g]].
destruct f as [a [b f]].
simpl.
apply Typeoidrefl.

(* K5 *)
intros u u'.
intros H1 H2.
destruct u as [g [f p]].
destruct u' as [g' [f' p']].
Swesetoid.

(* K6 *)
intros f g.
intro H.
assert ((catcod_from_family F) g ≈≈≈, { A }
(catdom_from_family F) f) as H1.
apply Typeoidsym.
apply H.
exists (cms_from_family_builder F g f H1).
split. apply Typeoidrefl. apply Typeoidrefl.

(* K7 *)
intro u.
intro y.
intro H.
```

```

destruct u as [g [↑ p]].
destruct g as [c [d g]].
destruct f as [a [b f]].
(* p: d = a *)

destruct H as [e [k H]].
simpl.
assert (c ≈≈, { A } a) as H2.
Swesetoid.
exists H2.
assert (b ≈≈, { A } b) as H3.
apply Typeoidrefl.
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 Typeoidfamilyrefgeneral.
assert (f (F • p (g x)) ≈≈, { F b } f (F • H2 x))
as H7.
apply Typeoidmapextensionality.
assert (y ≈≈, { A } a) as q.
Swesetoid.
assert (F • q (F • k (g x)) ≈≈, { F a } F • q (F •
e x)) as H8.
apply Typeoidmapextensionality.
apply H5.
clear H5.
assert (F • q (F • k (g x)) ≈≈, { F a } F • p (g
x)).
apply Typeoidfamilycmpgeneral.
assert (F • q (F • e x) ≈≈, { F a } F • H2 x).
apply Typeoidfamilycmpgeneral.
Swesetoid.
Swesetoid.

```

```

(* K8 *)
intro u.
intro y.
intro H.

destruct u as [g [f p]].
destruct g as [c [d g]].
destruct f as [a [b f]].
(* p: d = a *)
destruct H as [e [k H]].
simpl.
assert (c ≈≈, { A } c) as H2.
apply Typeoidrefl.
exists H2.
assert (b ≈≈, { A } d) as H3.
apply Typeoidtra with y.
assumption.
apply Typeoidsym.
Swesetoid.
exists H3.
intro x.
assert (g (F • H2 x) ≈≈, { F d } g x) as H0.
apply Typeoidmapextensionality.
apply Typeoidfamilyrefgeneral.
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 Typeoidsym.
apply Typeoidfamilycmpgeneral.
assert (F • H6 (F • k (f (F • p (g x)))) ≈≈, { F d
}

```



```

(F • H6 (F • e (F • p (g x)))) as H8.
apply Typeoidmapextensionality.
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 Typeoidfamilycmpgeneral.
assert (F • (H6 ⊙ e) (F • p (g x)) ≈≈≈, { F d }g x).
apply Typeoidfamilycmpinvert.
Swesetoid.
Swesetoid.
Swesetoid.
(* K9 *)
intros u v w z.
destruct u as [u1 [u2 up]].
destruct u1 as [u1d [u1c u1]].
destruct u2 as [u2d [u2c u2]].
destruct v as [v1 [v2 vp]].
destruct v1 as [v1d [v1c v1]].
destruct v2 as [v2d [v2c v2]].
destruct w as [w1 [w2 wp]].
destruct w1 as [w1d [w1c w1]].
destruct w2 as [w2d [w2c w2]].
destruct z as [z1 [z2 zp]].
destruct z1 as [z1d [z1c z1]].
destruct z2 as [z2d [z2c z2]].
simpl.
intros H1 H2 H3 H4 H5.
destruct H1 as [p1 [q1 H1]].
destruct H2 as [p2 [q2 H2]].
destruct H3 as [p3 [q3 H3]].
destruct H4 as [p4 [q4 H4]].
destruct H5 as [p5 [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 Typeoidmapextensionality.
apply Typeoidfamilycmpinvert.
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 Typeoidmapextensionality.
apply H4.
assert (w2 x0 ≈≈, { F w2c } F • q4-1 (F • q4 (w2
x0))).
apply Typeoidsym.
apply Typeoidfamilycmpinvert.
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 Typeoidmapextensionality.
    apply Typeoidfamilycmpinvert.
    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 Typeoidsym.
    apply Typeoidfamilycmpinvert.
    specialize (H1 x0).
    assert (F • q1-1 (F • q1 (w1 x0)) ≈≈≈ F • q1-1
(v1 (F • p1 x0))) as H99.
    apply Typeoidmapextensionality.
    apply H1.
    Swesetoid.
    clear H1.
    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 Typeoidmapextensionality.
    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 Typeoidsym.
    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 Typeoidmapextensionality.
apply Typeoidmapextensionality.
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 Typeoidmapextensionality.
apply Typeoidmapextensionality.
apply Typeoidmapextensionality.
apply Typeoidmapextensionality.
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))) as HX.
  apply Typeoidsym.

  apply Typeoidfamilycmpinvert.
  assert (
F • q2 -1 (F • q2 (v2 x0))
≈≈,{ F v2c }F • q2 -1 (u1 (F • p2 x0))) as HY.
  apply Typeoidmapextensionality.
  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)))))))))) as HZ.

  apply Typeoidfamilycmpgeneral_3.
  apply Typeoidfamilycmpgeneral_3.
  apply Typeoidfamilycmpgeneral_3.
  apply Typeoidfamilyirrgeneral.
  apply Typeoidmapextensionality.
  apply Typeoidfamilycmpgeneral_3.
  apply Typeoidsym.
  apply Typeoidfamilycmpgeneral_3.
  apply Typeoidfamilycmpgeneral_3.
  apply Typeoidfamilyirrgeneral.
  apply Typeoidmapextensionality.
  apply Typeoidfamilycmpgeneral.

  specialize (H2'(F • vp (v1 (F • p5 -1 (F • p6
x))))).
  assert (F • p3 -1
(F • zp
(F • q5
(F • q2 -1 (u1 (F • p2 (F • vp (v1 (F
• p5 -1 (F • p6 x)))))))))) ≈≈

```

```

(F • p3 -1
  (F • zp
    (F • q5
      (v2 (F • vp (v1 (F • p5 -1 (F • p6 x)))) )))))

```

as HW.

```

apply Typeoidmapextensionality.
apply Typeoidmapextensionality.
apply Typeoidmapextensionality.
apply Typeoidsym.
apply H2'.
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)))))) as
HU.

```

```

apply Typeoidsym.
apply Typeoidfamilycmpgeneral_3.
apply Typeoidfamilyirrgeneral.
apply Typeoidmapextensionality.
Swesetoid.
Swesetoid.

```

Defined.

Definition `isofunctor_ob` := Idmapp ObV:
Typeoidmap (Catobj (Cat_from_family Rbar))
(Catobj Vcat).

Definition `isofunctor_arr_helper`: (Catarr
(Cat_from_family Rbar)) -> ArrV.
 intros [u [v [hmap hext]]].

 simpl in hmap.

```

exists < <u, v> ,
      (sup (iV u) (fun x=> <(pV u) x, (pV v) (hmap
x)> )) > .
unfold is_arrow.

exists u.
exists v.
exists(sup (iV u) (fun x=> <(pV u) x, (pV v) (hmap
x)> )).
split.
apply refV.
apply functionspacechar.
unfold memV.
exists (existT _ hmap hext).
apply refV.
Defined.

```

Definition isofunctor_arr:

Typeoidmap (Catarr (Cat_from_family Rbar)) ArrV.
 apply (Build_Typeoidmap (Catarr (Cat_from_family
 Rbar)) ArrV isofunctor_arr_helper).

```

intros [u [v [hmap hext]]]
[u' [v' [hmap' hext']]] H.
destruct H as [phi [psi eq]].
simpl in eq.
destruct u as [a f].
destruct v as [b g].
destruct u' as [a' f'].
destruct v' as [b' g'].
simpl in eq.
simpl.
assert ( < < sup a f, sup b g > , sup a (λ x : a, <
f x, g (hmap x) > ) > ) =
< < sup a' f', sup b' g' > , sup a' (λ x : a',
< f' x, g' (hmap' x) > ) > ) as L.
apply pairing_extensional.
annlv pairing_extensional

```

```

apply pairing_extensional.
apply phi.
apply psi.
apply Extensionality.
intro z.
split.
intro Q.
destruct Q as [x xeq].
simpl in x.
simpl in xeq.
exists (push phi x).
simpl.
assert ( < f x, g (hmap x) > ≡ < f' (push phi x),
g' (hmap' (push phi x)) > ) as H.
apply pairing_extensional.
apply pushprop.
assert (g (hmap x) ≡ g' (push psi (hmap x))) as H2.
apply pushprop.
apply (traV _ _ _ H2 (eq x)).
apply (traV _ _ _ xeq H).
intro Q2.
destruct Q2 as [x xeq].
simpl in x.
simpl in xeq.
exists (pull phi x).
simpl.
assert ( < f' x, g' (hmap' x) > ≡
< f (pull phi x), g (hmap (pull phi x)) > ) as H2.
apply pairing_extensional.
apply symV.
apply pullprop.
simpl in hmap.
simpl in hmap'.
simpl in hext.
simpl in hext'.
assert (forall t: b, g t ≡ g' (push psi t)) as L1.
intro t.
annlv nushnon.

```



```

    apply pushprop.
    assert (forall t: b', g (pull psi t) ≐ g' t) as L2.
    intro t.
    apply pullprop.
    specialize (L1 (hmap (pull phi x))).
    specialize (eq (pull phi x)).
    specialize (L2 (hmap' x)).
    assert ( g' (hmap' x) ≐ g' (hmap' (push phi (pull
phi x)))) as L3.
    apply hext'.
    assert (f' x ≐ f (pull phi x)) as L4.
    apply symV.
    apply pullprop.
    assert (f (pull phi x) ≐ f' (push phi (pull phi
x))) as L5.
    apply pushprop.
    apply (traV _ _ _ L4 L5).
    apply (traV _ _ _ L3 (traV _ _ _ (symV _ _ eq)
(symV _ _ L1))).
    apply (traV _ _ _ xeq H2).
    apply L.
Defined.

```

Theorem isofunctor_arr_lemma (u v: VTypeoid)(h: Typeoidmap (Rbar u) (Rbar v)):

$$\text{projT1 (isofunctor_arr (existT _ u (existT _ v h)))}$$

$$\doteq$$

$$\langle \langle u, v \rangle ,$$

$$(\text{sup (iV u) (fun x => } \langle (\text{pV u) x, (pV v) (h$$

$$\text{x}) \rangle \rangle) \rangle .$$

Proof.

```

    destruct h.
    simpl.
    assert (
    < < u, v > , sup u (λ x : u, < u x, v
(Typeoidmapfunction0 x) > ) > ≐
    < < u, v > , sup u (λ x : u, < u x, v
(Typeoidmapfunction0 v) > ) > > ) as G

```

```

(Typeoidmap A B) (x : A) (y : A) (h : A → B) (u : B)
  apply refV.
apply G.
Defined.

```

Definition Injective (A B:Typeoid)

```

(f: Typeoidmap A B):=
(∀ x y: A, f x ≈≈ f y → x ≈≈ y).

```

Definition Surjective (A B:Typeoid)

```

(f: Typeoidmap A B):=
(∀ u: B, ∃ x:A, f x ≈≈ u).

```

Theorem isofunctor_arr_is_injective:

```

Injective (Catarr (Cat_from_family Rbar)) ArrV
isofunctor_arr.

```

Proof.

```

intros [u [v [hmap hext]]] [u' [v' [hmap'
hext']]].
destruct u as [a f].
destruct v as [b g].
destruct u' as [a' f'].
destruct v' as [b' g'].
intro H.
simpl.
assert (⟨ ⟨ sup a f, sup b g ⟩ , sup a (λ x : a, ⟨
f x, g (hmap x) ⟩) ⟩ ≡
⟨ ⟨ sup a' f', sup b' g' ⟩ , sup a' (λ x : a',
⟨ f' x, g' (hmap' x) ⟩) ⟩) as L.
apply H.
clear H.
assert (⟨ sup a f, sup b g ⟩ ≡ ⟨ sup a' f', sup
b' g' ⟩) as L1.
apply (pairing_injective _ _ _ _ L).
assert (sup a f ≡ sup a' f') as L11.
apply (pairing_injective _ _ _ _ L11)

```

```

apply (pairing_injective _ _ _ _ L1).
assert (sup b g  $\doteq$  sup b' g') as L12.
apply (pairing_injective _ _ _ _ L1).
simpl in hmap.
simpl in hmap'.
simpl in hext.
simpl in hext'.

exists L11.
exists L12.
assert (sup a ( $\lambda$  x : a,  $\langle$  f x, g (hmap x)  $\rangle$ )  $\doteq$ 
      sup a' ( $\lambda$  x : a',  $\langle$  f' x, g' (hmap' x)  $\rangle$ ))
as L2.
apply (pairing_injective _ _ _ _ L).
intro x.
assert (  $\langle$  f x, g (hmap x)  $\rangle$   $\doteq$ 
 $\langle$  f' (push L2 x), g' (hmap' (push L2 x))  $\rangle$  ) as L5.
apply (pushprop L2).
assert ( f x  $\doteq$  f' (push L2 x)) as H1.
apply (pairing_injective _ _ _ _ L5).
assert (g (hmap x)  $\doteq$  g' (hmap' (push L2 x))) as H2.
apply (pairing_injective _ _ _ _ L5).
clear L.
clear L1.

assert (g (hmap x)  $\doteq$  g' (push L12 (hmap x))) as L3.
apply pushprop.
assert (g' (hmap' (push L2 x))  $\doteq$  g' (hmap' (push
L11 x)) ) as L4.
apply hext'.
assert (f x  $\doteq$  f' (push L11 x)) as L6.
apply pushprop.
apply (traV _ _ _ (symV _ _ H1) L6).
apply (traV _ _ _ (symV _ _ L3)
      (traV _ _ _ H2 L4)).
Qed.

```

```

Definition arFN (x y: V): (x  $\Rightarrow$  y)  $\rightarrow$ 
Typeoidmap (Rbar_ob x) (Rbar_ob y).
  intros [f fext].
  apply (Build_Typeoidmap (Rbar_ob x) (Rbar_ob y) f).
  simpl. assumption.
Defined.

```

```

Theorem function_extract (x y z:V)
(p:is_function x y z): x  $\Rightarrow$  y.

```

```

Proof.
  assert (z  $\in$  x  $\Rightarrow$  y) as H.
  destruct x.
  destruct y.
  destruct z.
  apply functionspacechar.
  assumption.
  apply (projT1 H).
Defined.

```

```

Definition make_an_arr (a b phi: V)(p: is_function a
b phi) :  $\exists$ a : V,  $\exists$ b : V, Typeoidmap (Rbar_ob a)
(Rbar_ob b).

```

```

  exists a.
  exists b.
  apply arFN.
  apply (function_extract a b phi p).
Defined.

```

```

Theorem isofunctor_arr_is_surjective:
Surjective (Catarr (Cat_from_family Rbar)) ArrV
isofunctor_arr.

```

```

Proof.
  intros [u [v [w [phi [q1 q2]]]]].
  simn1

```

```

    simpl.
    exists (make_an_arr v w phi q2).
    assert ( projT1 (isofunctor_arr_helper (make_an_arr
v w phi q2)) = < < v, w > , phi > ) as H.
    unfold make_an_arr.

    unfold arFN.

    unfold function_extract.

    unfold isofunctor_arr_helper.
    simpl.
    destruct u as [a f].
    destruct v as [a0 f0].
    destruct w as [a1 f1].
    destruct phi as [a2 f2].

    simpl.

    unfold functionspacechar.
    unfold pairing_extensional.
    unfold pairing_injective.
    simpl.
    destruct q2 as [i1 i0].
    destruct i0 as [i0 i2].

    assert (
< < sup a0 f0, sup a1 f1 > ,
      sup a0 (λ x : a0, < f0 x, f1 (projT1 (i0 x))
> ) >
≡ < < sup a0 f0, sup a1 f1 > , sup a2 f2 > ) as Q.
    apply pairing_extensional.
    apply refV.

    apply Extensionality.
    intro z.
    split.
    intro P.
    unfold memV in P

```

```

unfold memv in P.
destruct P as [idx P].
simpl in idx.
simpl in P.
unfold is_total in i0.

assert (⟨ f0 idx, f1 (projT1 (i0 idx)) ⟩ ∈ sup a2
f2) as P1.
apply (projT2 (i0 idx)).
apply (ext1 _ _ _ P P1).
intro P.
unfold is_rel in i1.
assert (z ∈ sup a0 f0 × sup a1 f1) as P2.
apply (i1 z).
apply P.
assert (∃ x:a0, ∃ y:a1, z ≡ ⟨f0 x, f1 y⟩) as P3.
assert ((z ∈ sup a0 f0 × sup a1 f1
  -> (∃α : sup a0 f0, ∃β : sup a1 f1, z ≡ ⟨ (sup a0
f0) α, (sup a1 f1) β ⟩))) as P4.
apply productchar.
apply P4.
apply P2.
destruct P3 as [x [y P5]].
exists x.
simpl.
assert (⟨ f0 x, f1 (projT1 (i0 x)) ⟩ ∈ sup a2 f2)
as P1.
apply (projT2 (i0 x)).
assert (⟨ f0 x, f1 y ⟩ ∈ sup a2 f2) as P3.
apply (ext1 _ _ _ (symV _ _ P5) P).
assert (⟨ f0 x, f1 (projT1 (i0 x)) ⟩ ≡
⟨ f0 x, f1 y ⟩) as P6.
apply pairing_extensional.
apply refV.
unfold is_functional in i2.
apply (i2 (f0 x)).
split.
assumption.

```

```

assumption.
apply (traV _ _ _ P5 (symV _ _ P6)).
apply Q.
apply (traV _ _ _ H (symV _ _ q1)).
Defined.

```

```

Definition isofunctor_cms_helper:
(Catcms (Cat_from_family Rbar)) -> CmsV.
  intros [arr1 [arr2 P]].
  exists ⟨projT1 (isofunctor_arr arr1),
projT1 (isofunctor_arr arr2)⟩ .
  exists (isofunctor_arr arr1).
  exists (isofunctor_arr arr2).
  split.
  apply refV.
  destruct arr1 as [d1 [c1 f1]].
  destruct arr2 as [d2 [c2 f2]].
  simpl in P.
  destruct f1.
  destruct f2.
  simpl.
  apply P.
Defined.

```

```

Definition isofunctor_cms:
Typeoidmap (Catcms (Cat_from_family Rbar)) CmsV.
apply (Build_Typeoidmap (Catcms (Cat_from_family
Rbar)) CmsV isofunctor_cms_helper).
  intros [arr1 [arr2 P]] [arr1' [arr2' P']].
  intro Q.
  assert (arr1 ≈≈ arr1' ∧ arr2 ≈≈ arr2') as Q'.
  apply Q.
  clear Q.

  assert ( / projT1 (isofunctor_arr helper arr1)

```

```

    assert ( < projT1 (isofunctor_arr_helper arr1),
             projT1 (isofunctor_arr_helper arr2) >  =
    < projT1 (isofunctor_arr_helper arr1'),
             projT1 (isofunctor_arr_helper arr2') > ) as
G1.
    apply pairing_extensional.
    assert ( isofunctor_arr_helper arr1 ≈≈≈, { ArrV }
isofunctor_arr_helper arr1') as G11.
    apply (Typeoidmapextensionality
(arr_from_family Rbar) ArrV isofunctor_arr).
    apply Q'.
    apply G11.
    assert ( isofunctor_arr_helper arr2 ≈≈≈, { ArrV }
isofunctor_arr_helper arr2') as G12.
    apply (Typeoidmapextensionality
(arr_from_family Rbar) ArrV isofunctor_arr).
    apply Q'.
    apply G12.
    apply G1.
Defined.

```

Theorem isofunctor_cms_is_injective:

Injective (Catcms (Cat_from_family Rbar)) CmsV
isofunctor_cms.

Proof.

```

    unfold Injective.
    intros [arr1 [arr2 P]] [arr1' [arr2' P']].
intro Q.
    assert ( < projT1 (isofunctor_arr_helper arr1),
             projT1 (isofunctor_arr_helper arr2) >  =
    < projT1 (isofunctor_arr_helper arr1'),
             projT1 (isofunctor_arr_helper arr2') > ) as
Q'.
    apply Q.
    clear Q.
    assert (arr1 ≈≈≈ arr1' ∧ arr2 ≈≈≈ arr2') as G.
    ...

```



```

split.
assert (projT1 (isofunctor_arr_helper arr1)
≡ projT1 (isofunctor_arr_helper arr1')) as Q1.
apply (pairing_injective _ _ _ Q').
apply isofunctor_arr_is_injective.

apply Q1.
assert (projT1 (isofunctor_arr_helper arr2)
≡ projT1 (isofunctor_arr_helper arr2')) as Q2.
apply (pairing_injective _ _ _ Q').
apply isofunctor_arr_is_injective.
apply Q2.
apply G.
Defined.

```

(* The functor that gives the isomorphism of
the two categories *)

Theorem isofunctor: Functor (Cat_from_family Rbar)
Vcat.

```

apply (Build_Functor (Cat_from_family Rbar) Vcat
isofunctor_ob isofunctor_arr isofunctor_cms).

```

Proof.

(* Prop1 *)

```

intro a.
assert (⟨ ⟨ a, a ⟩ , sup a (λ x : a, ⟨ a x, a x
⟩ ) ⟩ ≡
⟨ ⟨ a, a ⟩ , identity a ⟩ ) as G.
apply pairing_extensional.
apply refV.
unfold identity.
apply Extensionality.
intro z.
assert (z ∈ { { p ∈ a × a | (∃ y : a, p ≡ ⟨ a y, a y
⟩ ) } } ↔ z ∈ a × a ∧ (∃ y : a, z ≡ ⟨ a y, a y ⟩ )) as
L.

```

```

apply (separationchar _ (expr_extproof_1 a)).
split.
intro P.
apply L.
unfold memV in P.
destruct P as [x P].
simpl in x.
simpl in P.
split.
assert (⟨ a x, a x ⟩ ∈ a × a).
unfold memV.
exists (x,x).
apply refV.
apply (ext1 _ _ _ P H).
exists x.
apply P.
intro P.
unfold memV.
simpl.
apply L.
apply P.
apply G.

```

```

(* Prop2 *)
intros [a [b f]].
unfold isofunctor_arr.
simpl.
unfold pairing_injective.
simpl.
destruct f.
unfold is_arrow.
simpl.
apply refV.

```

```

(* Prop3 *)
intros [a [b f]].
unfold isofunctor_arr.
simpl.

```

```

    simpl.
    unfold pairing_injective.
    simpl.
    destruct f.
    unfold is_arrow.

    simpl.
    apply refV.

    (* Prop4 *)
    intros [[a [b f]] [[a' [b' f']] P]].
    destruct f.
    destruct f'.
    simpl.
    assert ( < < a, b > , sup a (λ x : a, < a x, b
(Typeoidmapfunction0 x) > ) > )
≡ < < a, b > , sup a (λ x : a, < a x, b
(Typeoidmapfunction0 x) > ) > ) as G.
    apply refV.
    apply G.

    (* Prop5 *)

    intros [[a [b f]] [[a' [b' f']] P]].
    destruct f.
    destruct f'.
    simpl.
    assert (
< < a', b' > , sup a' (λ x : a', < a' x, b'
(Typeoidmapfunction1 x) > ) > )
≡
< < a', b' > , sup a' (λ x : a', < a' x, b'
(Typeoidmapfunction1 x) > ) > )
) as G.
    apply refV.
    apply G.

    (* Prop6 *)

```

```

intros [[a [b f]] [[a' [b' f']] P]].
destruct f.
destruct f'.
simpl in P.
simpl.
assert (
< < a, b' > ,
  sup a
    (λ x : a,
      < a x,
        b'
          (Typeoidmapfunction1
            (Rbar_transp_helper b a' P
              (Typeoidmapfunction0 x))) > )) ≡
  < < a, b' > ,
    compose_relV a b b'
      (sup a (λ x : a, < a x, b
        (Typeoidmapfunction0 x) > ))
        (sup a' (λ x : a', < a' x, b'
          (Typeoidmapfunction1 x) > )) > ) as G.
  apply pairing_extensional.
  apply refV.
  apply Extensionality.
  intro z.
  split.
  intro H.
  apply compose_relV_char_2.

  unfold is_rel.
  intro s.
  intro H2.
  unfold memV in H2.
  destruct H2 as [ix H3].
  simpl in ix.
  simpl in H3.
  exists (ix, (Typeoidmapfunction0 ix)).
  simpl.

```

apply H3.

unfold is_rel.

intro s.

intro H2.

unfold memV in H2.

destruct H2 as [ix H3].

simpl in ix.

simpl in H3.

assert ($s \in a' \times b'$) as H4.

exists (ix, (Typeoidmapfunction1 ix)).

simpl.

apply H3.

apply (prod_extensional_half a' b' b b').

apply (symV _ _ P).

apply (refV b').

apply H4.

unfold memV in H.

destruct H as [ix H2].

simpl in ix.

simpl in H2.

exists (a ix).

exists (b'

(Typeoidmapfunction1

(Rbar_transp_helper b a' P

(Typeoidmapfunction0 ix)))).

split.

apply H2.

split.

exists ix.

apply refV.

split.

exists (Typeoidmapfunction1

(Rbar_transp_helper b a' P

(Typeoidmapfunction0 ix)))).

apply refV.

```

exists (b (Typeoidmapfunction0 ix)).
split.
exists (Typeoidmapfunction0 ix).
apply refV.

split.
exists ix.
apply refV.
unfold memV.
destruct a.
destruct b.
destruct a'.
destruct b'.
exists (push P (Typeoidmapfunction0 ix)).
simpl.
assert ( <f0 (Typeoidmapfunction0 ix),
        f2 (Typeoidmapfunction1 (push P
(Typeoidmapfunction0 ix))) >
≡ < f1 (push P (Typeoidmapfunction0 ix)),
    f2 (Typeoidmapfunction1 (push P
(Typeoidmapfunction0 ix))) > )
as G.
apply pairing_extensional.
apply pushprop.
apply refV.
apply G.

intro Q.
assert ((∃ x:V, ∃ z':V,
z ≡ < x, z' > ∧ x ∈ a ∧ z' ∈ b' ∧
  ∃y:V, y ∈ b ∧ < x, y > ∈ (sup a (λ x : a, < a x,
b (Typeoidmapfunction0 x) > ))
  ∧ < y, z' > ∈ (sup a' (λ x : a', < a' x, b'
(Typeoidmapfunction1 x) > )))) as Q'.
  apply compose_relV_char_2.

unfold is_rel.

```

```

intro t.
intro H.
unfold memV in H.
destruct H as [ix H1].
simpl in ix.
simpl in H1.
exists (ix, (Typeoidmapfunction0 ix)).
simpl.
apply H1.

```

```

unfold is_rel.
intro s.
intro H2.
unfold memV in H2.
destruct H2 as [ix H3].
simpl in ix.
simpl in H3.
assert (s  $\in$  a'  $\times$  b') as G.
exists (ix, (Typeoidmapfunction1 ix)).
simpl.
apply H3.
apply (prod_extensional_half a' b' b b').
apply (symV _ _ P).
apply (refV b').
apply G.
apply Q.

```

```

destruct Q' as [x [z' [Q2 [Q3 [Q4 [u [Q6 [Q7
Q8]]]]]]]].
unfold memV in Q3.
destruct Q3 as [ix Q3].
unfold memV in Q4.
destruct Q4 as [ix' Q4].
unfold memV in Q6.
destruct Q6 as [ix' Q6].
unfold memV in Q7.
destruct Q7 as [ix4 Q7].

```

```

    simpl in ix4.
    unfold memV in Q8.
    destruct Q8 as [ix5 Q8].
    simpl in ix5.
    assert ( < x, u > ≐ < a ix4, b
(Typeoidmapfunction0 ix4) > ) as Q7'.
    apply Q7.
    clear Q7.
    assert ( < u, z' >
      ≐ < a' ix5, b' (Typeoidmapfunction1 ix5) > )
as Q8'.
    apply Q8.
    clear Q8.
    unfold memV.
    assert ( ∃idx
      : a,
      z
      ≐ < a idx, b'
        (Typeoidmapfunction1
          (Rbar_transp_helper b a' P
            (Typeoidmapfunction0 idx))) > ) as G.

    exists ix4.
    assert (x ≐ a ix4) as H1.
    apply (pairing_injective _ _ _ Q7').
    assert (z' ≐ b' (Typeoidmapfunction1 ix5)) as H2.
    apply (pairing_injective _ _ _ Q8').
    assert (b' (Typeoidmapfunction1 ix5) ≐ b'
      (Typeoidmapfunction1
        (Rbar_transp_helper b a' P
          (Typeoidmapfunction0 ix4)))) as H3.
    apply Typeoidmapextensionality1.
    assert (u ≐ b (Typeoidmapfunction0 ix4)) as Q9.
    apply (pairing_injective _ _ _ Q7').
    assert (u ≐ a' ix5) as QA.
    apply (pairing_injective _ _ _ Q8').
    assert (a' ix5 ≐ b (Typeoidmapfunction0 ix4)) as

```


QB.

```
  apply (traV _ _ _ (symV _ _ QA) Q9).
  assert (a' (Rbar_transp_helper b a' P
(Typeoidmapfunction0 ix4)))  $\doteq$  b (Typeoidmapfunction0
ix4)) as QC.
```

```
  apply symV.
  destruct a'.
  destruct b.
  apply pushprop.
  apply (traV _ _ _ QB (symV _ _ QC)).
  assert (⟨ x, z' ⟩  $\doteq$  ⟨ a ix4,
```

b'

```
      (Typeoidmapfunction1
      (Rbar_transp_helper b a' P
(Typeoidmapfunction0 ix4))) > _ ) as QD.
```

```
  apply pairing_extensional.
  apply H1.
  apply (traV _ _ _ H2 H3).
  apply (traV _ _ _ Q2 QD).
  apply G.
  apply G.
```

Defined.

(* Finally we establish that isofunctor is surjective also for the Cms component. This concludes the proof that isofunctor is an isomorphism of categories as was to be proved *)

Definition isofunctor_arr_inverse (u:ArrV):
(projT1 (isofunctor_arr_is_surjective u)):
(Catarr (Cat_from_family Rbar)).

Theorem isofunctor_arr_inverse_prop (u:ArrV):
isofunctor_arr (isofunctor_arr_inverse u) $\approx\approx\approx$, { ArrV
} u.

Proof.

```
    apply (projT2 (isofunctor_arr_is_surjective u)).  
Defined.
```

Theorem `isofunctor_cms_is_surjective`:

Surjective (Catcms (Cat_from_family Rbar)) CmsV
isofunctor_cms.

Proof.

```
    unfold Surjective.  
    intros [w [arr1 [arr2 P]]].  
    assert (isofunctor_arr (isofunctor_arr_inverse  
arr1) ≈≈≈, { ArrV } arr1) as L1.  
    apply isofunctor_arr_inverse_prop.  
    assert (isofunctor_arr (isofunctor_arr_inverse  
arr2) ≈≈≈, { ArrV } arr2) as L2.  
    apply isofunctor_arr_inverse_prop.  
    assert (codV (isofunctor_arr  
(isofunctor_arr_inverse arr1)) ≈≈≈, { ObV } codV arr1)  
as L3.  
    apply Typeoidmapextensionality.  
    apply L1.  
    assert (domV (isofunctor_arr  
(isofunctor_arr_inverse arr2)) ≈≈≈, { ObV } domV arr2)  
as L4.  
    apply Typeoidmapextensionality.  
    apply L2.  
    assert (codV (isofunctor_arr  
(isofunctor_arr_inverse arr1)) ≈≈≈, { ObV } (Catcod  
(Cat_from_family Rbar)) (isofunctor_arr_inverse  
arr1)) as L5.  
    apply Typeoidsym.  
    simpl.  
    apply (Fun_cod (Cat_from_family Rbar)  
Vcat isofunctor).  
    assert (domV (isofunctor_arr  
(isofunctor_arr_inverse arr2)) ≈≈≈, { ObV } (Catdom  
(Cat_from_family Rbar)) (isofunctor_arr_inverse
```

```

arr2)) as L6.
  apply Typeoidsym.
  simpl.
  apply (Fun_dom (Cat_from_family Rbar)
Vcat isofunctor).

  assert ((Catcod (Cat_from_family Rbar))
(isofunctor_arr_inverse arr1) ≈≈≈, { ObV }
(Catdom (Cat_from_family Rbar))
(isofunctor_arr_inverse arr2)
) as L7.
  destruct P as [P1 P2].
  apply (traV _ _ _ (symV _ _ L5) (traV _ _ _ L3
    (traV _ _ _ P2 (traV _ _ _ (symV _ _ L4) L6))))).
  exists (cms_from_family_builder Rbar
(isofunctor_arr_inverse arr1)
(isofunctor_arr_inverse arr2)
L7).
  assert ( ⟨projT1 (isofunctor_arr_helper
(isofunctor_arr_inverse arr1)),
    projT1 (isofunctor_arr_helper
(isofunctor_arr_inverse arr2)) ⟩ ≐ w) as G.
  assert ( ⟨ projT1 arr1, projT1 arr2 ⟩ ≐
    ⟨ projT1 (isofunctor_arr_helper
(isofunctor_arr_inverse arr1)),
    projT1 (isofunctor_arr_helper
(isofunctor_arr_inverse arr2)) ⟩ ) as L8.
  apply pairing_extensional.
  apply symV.
  apply L1.
  apply symV.
  apply L2.
  apply symV.
  destruct P as [P1 P2].
  apply (traV _ _ _ P1 L8).
  simpl.
  apply G.
Defined.

```

