Category Theory

Alexander Katovsky

March 17, 2025

Abstract

This article presents a development of Category Theory in Isabelle. A Category is defined using records and locales in Isabelle/HOL. Functors and Natural Transformations are also defined. The main result that has been formalized is that the Yoneda functor is a full and faithful embedding. We also formalize the completeness of many sorted monadic equational logic. Extensive use is made of the HOLZF theory in both cases. For an informal description see [1].

Contents

1 Category

theory *Category*
imports *HOL−Library.FuncSet*
begin

record $('o,'m)$ *Category* =
  Obj :: $'o$ *set* (‹*obj*₁› *70*)
  Mor :: $'m$ *set* (‹*mor*₁› *70*)
  Dom :: $'m \Rightarrow 'o$ (‹*dom*₁ -› [*80*] *70*)
  Cod :: $'m \Rightarrow 'o$ (‹*cod*₁ -› [*80*] *70*)

Id :: $'o \Rightarrow 'm$ (‹*id*₁ -› [*80*] *75*)
Comp :: $'m \Rightarrow 'm \Rightarrow 'm$ (**infixl** ‹;;₁› *70*)

definition
MapsTo :: ($'o$,$'m$,$'a$) *Category-scheme* $\Rightarrow 'm \Rightarrow 'o \Rightarrow 'o \Rightarrow$ *bool* (‹- *maps*₁ - *to* -› [*60*, *60*, *60*] *65*) **where**
MapsTo CC f X Y $\equiv f \in$ *Mor CC* $\wedge$ *Dom CC f* $= X \wedge$ *Cod CC f* $= Y$

definition
CompDefined :: ($'o$,$'m$,$'a$) *Category-scheme* $\Rightarrow 'm \Rightarrow 'm \Rightarrow$ *bool* (**infixl** ‹≈>₁› *65*) **where**
CompDefined CC f g $\equiv f \in$ *Mor CC* $\wedge$ $g \in$ *Mor CC* $\wedge$ *Cod CC f* $=$ *Dom CC g*

locale *ExtCategory* =
fixes *C* :: ($'o$,$'m$,$'a$) *Category-scheme* (**structure**)
assumes *CdomExt*: (*Dom C*) $\in$ *extensional* (*Mor C*)
and *CcodExt*: (*Cod C*) $\in$ *extensional* (*Mor C*)
and *CidExt*: (*Id C*) $\in$ *extensional* (*Obj C*)
and *CcompExt*: (*case-prod* (*Comp C*)) $\in$ *extensional* ({(*f*,*g*) | *f g* . *f* ≈> *g*})

locale *Category* = *ExtCategory* +
assumes *Cdom* : $f \in$ *mor* $\Longrightarrow$ *dom f* $\in$ *obj*
and *Ccod* : $f \in$ *mor* $\Longrightarrow$ *cod f* $\in$ *obj*
and *Cidm* [*dest*]: $X \in$ *obj* $\Longrightarrow$ (*id X*) *maps X to X*
and *Cidl* : $f \in$ *mor* $\Longrightarrow$ *id* (*dom f*) ;; $f = f$
and *Cidr* : $f \in$ *mor* $\Longrightarrow$ f ;; *id* (*cod f*) $= f$
and *Cassoc* : $[\![f \approx> g$; $g \approx> h]\!] \Longrightarrow (f$;; $g)$;; $h = f$;; $(g$;; $h)$
and *Ccompt* : $[\![f$ *maps X to Y* ; g *maps Y to Z*$]\!] \Longrightarrow (f$;; $g)$ *maps X to Z*

definition
MakeCat :: ($'o$,$'m$,$'a$) *Category-scheme* $\Rightarrow$ ($'o$,$'m$,$'a$) *Category-scheme* **where**
MakeCat C $\equiv$ (|
Obj $=$ *Obj C* ,
Mor $=$ *Mor C* ,
Dom $=$ *restrict* (*Dom C*) (*Mor C*) ,
Cod $=$ *restrict* (*Cod C*) (*Mor C*) ,
Id $=$ *restrict* (*Id C*) (*Obj C*) ,
Comp $= \lambda f\ g$. (*restrict* (*case-prod* (*Comp C*)) ({(*f*,*g*) | *f g* . *f* ≈>$_C$ *g*})) (*f*,*g*),
... $=$ *Category.more C*
|)

lemma *MakeCatMapsTo*: *f maps*$_C$ *X to Y* $\Longrightarrow$ *f maps*$_{MakeCat\ C}$ *X to Y*
by (*auto simp add*: *MapsTo-def MakeCat-def*)

lemma *MakeCatComp*: $f \approx>_C g \Longrightarrow f$;;$_{MakeCat\ C}$ $g = f$;;$_C$ g
by (*auto simp add*: *MapsTo-def MakeCat-def*)

lemma *MakeCatId*: $X \in obj_C \Longrightarrow id_C\ X = id_{MakeCat\ C}\ X$

by (*auto simp add*: *MapsTo-def MakeCat-def*)

lemma *MakeCatObj*: $obj_{MakeCat\ C} = obj_C$
by (*simp add*: *MakeCat-def*)

lemma *MakeCatMor*: $mor_{MakeCat\ C} = mor_C$
by (*simp add*: *MakeCat-def*)

lemma *MakeCatDom*: $f \in mor_C \Longrightarrow dom_C\ f = dom_{MakeCat\ C}\ f$
by (*simp add*: *MakeCat-def*)

lemma *MakeCatCod*: $f \in mor_C \Longrightarrow cod_C\ f = cod_{MakeCat\ C}\ f$
by (*simp add*: *MakeCat-def*)

lemma *MakeCatCompDef*: $f \approx>_{MakeCat\ C} g = f \approx>_C g$
by (*auto simp add*: *CompDefined-def MakeCat-def*)

lemma *MakeCatComp2*: $f \approx>_{MakeCat\ C} g \Longrightarrow f\ ;;_{MakeCat\ C}\ g = f\ ;;_C\ g$
by (*simp add*: *MakeCatCompDef MakeCatComp*)

lemma *ExtCategoryMakeCat*: *ExtCategory* (*MakeCat C*)
by (*unfold-locales*, *simp-all add*: *MakeCat-def extensional-def CompDefined-def*)

lemma *MakeCat*: *Category-axioms C* $\Longrightarrow$ *Category* (*MakeCat C*)
apply(*intro-locales*, *simp add*: *ExtCategoryMakeCat*)
apply (*simp add*: *Category-axioms-def*)
apply (*auto simp add*: *MakeCat-def CompDefined-def MapsTo-def*)
done

lemma *MapsToE*[*elim*]: $[\![f\ maps_C\ X\ to\ Y\ ;\ [\![f \in mor_C\ ;\ dom_C\ f = X\ ;\ cod_C\ f = Y]\!] \Longrightarrow R]\!] \Longrightarrow R$
by (*auto simp add*: *MapsTo-def*)

lemma *MapsToI*[*intro*]: $[\![f \in mor_C\ ;\ dom_C\ f = X\ ;\ cod_C\ f = Y]\!] \Longrightarrow f\ maps_C\ X\ to\ Y$
by (*auto simp add*: *MapsTo-def*)

lemma *CompDefinedE*[*elim*]: $[\![f \approx>_C g\ ;\ [\![f \in mor_C\ ;\ g \in mor_C\ ;\ cod_C\ f = dom_C\ g]\!] \Longrightarrow R]\!] \Longrightarrow R$
by (*auto simp add*: *CompDefined-def*)

lemma *CompDefinedI*[*intro*]: $[\![f \in mor_C\ ;\ g \in mor_C\ ;\ cod_C\ f = dom_C\ g]\!] \Longrightarrow f \approx>_C g$
by (*auto simp add*: *CompDefined-def*)

lemma (**in** *Category*) *MapsToCompI*: **assumes** $f \approx> g$ **shows** $(f\ ;;\ g)\ maps\ (dom\ f)\ to\ (cod\ g)$

proof−
have *f maps* (*dom f*) *to* (*dom g*)
and *g maps* (*dom g*) *to* (*cod g*) **using** *assms* **by** *auto*
thus *?thesis* **by** (*simp add*: *Ccompt*[*of f dom f dom g g cod g*])
qed

lemma *MapsToCompDef*:
assumes *f maps*$_C$ *X to Y* **and** *g maps*$_C$ *Y to Z*
shows *f* ≈>$_C$ *g*
proof(*rule CompDefinedI*)
show *f* ∈ *mor*$_C$ **and** *g* ∈ *mor*$_C$ **using** *assms* **by** *auto*
have *cod*$_C$ *f* = *Y* **and** *dom*$_C$ *g* = *Y* **using** *assms* **by** *auto*
thus *cod*$_C$ *f* = *dom*$_C$ *g* **by** *simp*
qed

lemma (**in** *Category*) *MapsToMorDomCod*:
assumes *f* ≈> *g*
shows *f* ;; *g* ∈ *mor* **and** *dom* (*f* ;; *g*) = *dom f* **and** *cod* (*f* ;; *g*) = *cod g*
proof−
have (*f* ;; *g*) *maps* (*dom f*) *to* (*cod g*) **using** *assms* **by** (*simp add*: *MapsToCompI*)
thus *f* ;; *g* ∈ *mor* **and** *dom* (*f* ;; *g*) = *dom f* **and** *cod* (*f* ;; *g*) = *cod g* **by** *auto*
qed

lemma (**in** *Category*) *MapsToObj*:
assumes *f maps X to Y*
shows *X* ∈ *obj* **and** *Y* ∈ *obj*
proof−
have *dom f* = *X* **and** *cod f* = *Y* **and** *f* ∈ *mor* **using** *assms* **by** *auto*
thus *X* ∈ *obj* **and** *Y* ∈ *obj* **by** (*auto simp add*: *Cdom Ccod*)
qed

lemma (**in** *Category*) *IdInj*:
assumes *X* ∈ *obj* **and** *Y* ∈ *obj* **and** *id X* = *id Y*
shows *X* = *Y*
proof−
have *dom* (*id X*) = *dom* (*id Y*) **using** *assms* **by** *simp*
moreover have *dom* (*id X*) = *X* **and** *dom* (*id Y*) = *Y* **using** *assms* **by** (*auto simp add*: *MapsTo-def*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *Category*) *CompDefComp*:
assumes *f* ≈> *g* **and** *g* ≈> *h*
shows *f* ≈> (*g* ;; *h*) **and** (*f* ;; *g*) ≈> *h*
proof(*auto simp add*: *CompDefined-def*)
show *f* ∈ *mor* **and** *h* ∈ *mor* **using** *assms* **by** *auto*
have *1*: *g* ;; *h maps* (*dom g*) *to* (*cod h*)
and *2*: *f* ;; *g maps* (*dom f*) *to* (*cod g*) **using** *assms* **by** (*simp add*: *MapsTo-CompI*)+

thus $g \;;;\; h \in mor$ **and** $f \;;;\; g \in mor$ **by** *auto*
have $cod\ f = dom\ g$ **using** *assms* **by** *auto*
also have $... = dom\ (g \;;;\; h)$ **using** *1* **by** *auto*
finally show $cod\ f = dom\ (g \;;;\; h)$ **.**
have $dom\ h = cod\ g$ **using** *assms* **by** *auto*
also have $... = cod\ (f \;;;\; g)$ **using** *2* **by** *auto*
finally show $cod\ (f \;;;\; g) = dom\ h$ **by** *simp*
qed

lemma (**in** *Category*) *CatIdInMor*: $X \in obj \Longrightarrow id\ X \in mor$
by (*auto simp add*: *Cidm*)

lemma (**in** *Category*) *MapsToId*: **assumes** $X \in obj$ **shows** $id\ X \approx> id\ X$
proof(*rule CompDefinedI*)
have *id X maps X to X* **using** *assms* **by** (*simp add*: *Cidm*)
thus $id\ X \in mor$ **and** $id\ X \in mor$ **and** $cod\ (id\ X) = dom\ (id\ X)$ **by** *auto*
qed

lemmas (**in** *Category*) *Simps* = *Cdom Ccod Cidm Cidl Cidr MapsToCompI IdInj MapsToId*

lemma (**in** *Category*) *LeftRightInvUniq*:
assumes *0*: $h \approx> f$ **and** *z*: $f \approx> g$
assumes *1*: $f \;;;\; g = id\ (dom\ f)$
and *2*: $h \;;;\; f = id\ (cod\ f)$
shows $h = g$
proof−
have *mor*: $h \in mor \wedge g \in mor$
and $dc : dom\ f = cod\ h \wedge cod\ f = dom\ g$ **using** *0 z* **by** *auto*
then have $h = h \;;;\; id\ (dom\ f)$ **by** (*auto simp add*: *Simps*)
also have $... = h \;;;\; (f \;;;\; g)$ **using** *1* **by** *simp+*
also have $... = (h \;;;\; f) \;;;\; g$ **using** *0 z* **by** (*simp add*: *Cassoc*)
also have $... = (id\ (cod\ f)) \;;;\; g$ **using** *2* **by** *simp+*
also have $... = g$ **using** *mor dc* **by** (*auto simp add*: *Simps*)
finally show *?thesis* **.**
qed

lemma (**in** *Category*) *CatIdDomCod*:
assumes $X \in obj$
shows $dom\ (id\ X) = X$ **and** $cod\ (id\ X) = X$
proof−
have *id X maps X to X* **using** *assms*
by (*simp add*: *Simps*)
thus $dom\ (id\ X) = X$ **and** $cod\ (id\ X) = X$ **by** *auto*
qed

lemma (**in** *Category*) *CatIdCompId*:
assumes $X \in obj$
shows $id\ X \;;;\; id\ X = id\ X$

proof−
have *0*: *id X* ∈ *mor* **using** *assms* **by** (*auto simp add*: *Simps*)
moreover have *cod* (*id X*) = *X* **using** *assms* **by** *auto*
moreover have *id X* ;; *id* (*cod* (*id X*)) = *id X* **using** *0* **by** (*simp add*: *Simps*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *Category*) *CatIdUniqR*:
assumes *iota*: ι *maps X to X*
and *rid*: ∀ *f* . *f* ≈> ι ⟶ *f* ;; ι = *f*
shows *id X* = ι
proof(*rule LeftRightInvUniq* [*of id X id X* ι])
have *0*: *X* ∈ *obj* **using** *iota* **by** (*auto simp add*: *Simps*)
hence *id X maps X to X* **by** (*simp add*: *Cidm*)
thus *1*: *id X* ≈> ι **using** *iota* **by** (*auto simp add*: *Simps*)
show *id X* ≈> *id X* **using** *0* **by** (*auto simp add*: *Simps*)
show (*id X*) ;; ι = (*id* (*dom* (*id X*))) **using** *0 1 rid* **by** (*auto simp add*: *Simps CompDefined-def MapsTo-def*)
show (*id X*) ;; (*id X*) = (*id* (*cod* (*id X*))) **using** *0* **by** (*auto simp add*: *CatIdCompId CompDefined-def MapsTo-def*)
qed

definition
inverse-rel :: ('*o*,'*m*,'*a*) *Category-scheme* ⇒ '*m* ⇒ '*m* ⇒ *bool* (‹*cinv*₁ - -› *60*)
where
inverse-rel C f g ≡ (*f* ≈>$_C$ *g*) ∧ (*f* ;;$_C$ *g*) = (*id*$_C$ (*dom*$_C$ *f*)) ∧ (*g* ;;$_C$ *f*) = (*id*$_C$ (*cod*$_C$ *f*))

definition
isomorphism :: ('*o*,'*m*,'*a*) *Category-scheme* ⇒ '*m* ⇒ *bool* (‹*ciso*₁ -› [*70*]) **where**
isomorphism C f ≡ ∃ *g* . *inverse-rel C f g*

lemma (**in** *Category*) *Inverse-relI*: ⟦*f* ≈> *g* ; *f* ;; *g* = *id* (*dom f*) ; *g* ;; *f* = *id* (*cod f*)⟧ ⟹ (*cinv f g*)
by (*auto simp add*: *inverse-rel-def*)

lemma (**in** *Category*) *Inverse-relE*[*elim*]: ⟦*cinv f g* ; ⟦*f* ≈> *g* ; *f* ;; *g* = *id* (*dom f*) ; *g* ;; *f* = *id* (*cod f*)⟧ ⟹ *P*⟧ ⟹ *P*
by (*auto simp add*: *inverse-rel-def*)

lemma (**in** *Category*) *Inverse-relSym*:
assumes *cinv f g*
shows *cinv g f*
proof(*rule Inverse-relI*)
have *1*: *f* ≈> *g* **using** *assms* **by** *auto*
show *2*: *g* ≈> *f*
proof(*rule CompDefinedI*)

```
    show g ∈ mor and 0: f ∈ mor using assms by auto
    have f ;; g maps dom f to cod g using 1 by (simp add: MapsToCompI)
    hence cod g = cod (f ;; g) by auto
   also have ... = cod (id (dom f)) using assms by (auto simp add: inverse-rel-def)
    also have ... = dom f
    proof−
      have dom f ∈ obj using 0 by (simp add: Simps)
      hence id (dom f) maps (dom f) to (dom f) by (simp add: Simps)
      thus ?thesis by auto
    qed
    finally show 2: cod g = dom f by simp
  qed
  show g ;; f = id (dom g) using assms by (auto simp add: inverse-rel-def)
  show f ;; g = id (cod g)using assms 1 2 by (auto simp add: inverse-rel-def)
qed

lemma (in Category) InverseUnique:
  assumes 1: cinv f g
  and     2: cinv f h
  shows   g = h
proof(rule LeftRightInvUniq [of g f h])
  have cinv g f using 1 2 by (simp only: Inverse-relSym[of f g])
  thus g ≈> f and
    g ;; f = id (cod f) using 1 by auto
  show f ≈> h using 2 by auto
  show f ;; h = id (dom f) using 2 by auto
qed

lemma (in Category) InvId: assumes X ∈ obj shows (cinv (id X) (id X))
proof(rule Inverse-relI)
  show id X ≈> id X using assms by (simp add: Simps)
   have dom (id X) = X and dom (id X) = X using assms by (simp add:
CatIdDomCod)+
  thus id X ;; id X = id (dom (id X)) and id X ;; id X = id (cod (id X))
    using assms by (simp add: CatIdCompId CatIdDomCod)+
qed

definition
  inverse :: ('o,'m,'a) Category-scheme ⇒ 'm ⇒ 'm (‹Cinv1 -› [70]) where
  inverse C f ≡ THE g . inverse-rel C f g

lemma (in Category) inv2Inv:
  assumes cinv f g
  shows   ciso f and Cinv f = g
proof−
  show ciso f using assms by (auto simp add: isomorphism-def)
  hence ∃! g . cinv f g using assms by (auto simp add: InverseUnique)
  thus Cinv f = g using assms by (auto simp add: inverse-def)
qed
```

lemma (**in** *Category*) *iso2Inv*:
assumes *ciso f*
shows *cinv f* (*Cinv f*)
proof−
have ∃! *g* . *cinv f g* **using** *assms* **by** (*auto simp add*: *InverseUnique isomorphism-def*)
thus *?thesis* **by** (*auto simp add*: *inverse-def intro*:*theI'*)
qed

lemma (**in** *Category*) *InvInv*:
assumes *ciso f*
shows *ciso* (*Cinv f*) **and** (*Cinv* (*Cinv f*)) = *f*
proof−
have *cinv f* (*Cinv f*) **using** *assms* **by** (*simp add*: *iso2Inv*)
hence *cinv* (*Cinv f*) *f* **by** (*simp add*: *Inverse-relSym*[*of f*])
thus *ciso* (*Cinv f*) **and** *Cinv* (*Cinv f*) = *f* **by** (*auto simp add*: *inv2Inv*)
qed

lemma (**in** *Category*) *InvIsMor*: (*cinv f g*) ⟹ (*f* ∈ *mor* ∧ *g* ∈ *mor*)
by (*auto simp add*: *inverse-rel-def*)

lemma (**in** *Category*) *IsoIsMor*: *ciso f* ⟹ *f* ∈ *mor*
by (*auto simp add*: *InvIsMor dest*: *iso2Inv*)

lemma (**in** *Category*) *InvDomCod*:
assumes *ciso f*
shows *dom* (*Cinv f*) = *cod f* **and** *cod* (*Cinv f*) = *dom f* **and** *Cinv f* ∈ *mor*
proof−
have *1*: *cinv f* (*Cinv f*) **using** *assms* **by** (*auto simp add*: *iso2Inv*)
thus *dom* (*Cinv f*) = *cod f* **by** (*auto simp add*: *inverse-rel-def*)
from *1* **have** *cinv* (*Cinv f*) *f* **by** (*auto simp add*: *Inverse-relSym*[*of f*])
thus *cod* (*Cinv f*) = *dom f* **by** (*auto simp add*: *inverse-rel-def*)
show *Cinv f* ∈ *mor* **using** *1* **by** (*auto simp add*: *inverse-rel-def*)
qed

lemma (**in** *Category*) *IsoCompInv*: *ciso f* ⟹ *f* ≈> *Cinv f*
by (*auto simp add*: *IsoIsMor InvDomCod*)

lemma (**in** *Category*) *InvCompIso*: *ciso f* ⟹ *Cinv f* ≈> *f*
by (*auto simp add*: *IsoIsMor InvDomCod*)

lemma (**in** *Category*) *IsoInvId1* : *ciso f* ⟹ (*Cinv f*) ;; *f* = (*id* (*cod f*))
by (*auto dest*: *iso2Inv*)

lemma (**in** *Category*) *IsoInvId2* : *ciso f* ⟹ *f* ;; (*Cinv f*) = (*id* (*dom f*))
by (*auto dest*: *iso2Inv*)

lemma (**in** *Category*) *IsoCompDef*:

```
  assumes 1: f ≈> g and 2: ciso f and 3: ciso g
  shows (Cinv g) ≈> (Cinv f)
proof(rule CompDefinedI)
  show Cinv g ∈ mor and Cinv f ∈ mor using assms by (auto simp add: InvDomCod)
  have cod (Cinv g) = dom g using 3 by (simp add: InvDomCod)
  also have ... = cod f using 1 by auto
  also have ... = dom (Cinv f) using 2 by (simp add: InvDomCod)
  finally show cod (Cinv g) = dom (Cinv f) .
qed

lemma (in Category) IsoCompose:
  assumes 1: f ≈> g and 2: ciso f and 3: ciso g
  shows ciso (f ;; g) and Cinv (f ;; g) = (Cinv g) ;; (Cinv f)
proof−
  have  a: (Cinv g) ≈> (Cinv f) using assms by (simp add: IsoCompDef)
  hence b: (Cinv g) ;; (Cinv f) maps (dom (Cinv g)) to (cod (Cinv f)) by (simp add: MapsToCompI)
  hence c: (Cinv g) ;; (Cinv f) maps (cod g) to (dom f) using 2 3 by (simp add: InvDomCod)
  have  d: f ;; g maps (dom f) to (cod g) using 1 by (simp add: Simps)
  have cinv (f ;; g) ((Cinv g) ;; (Cinv f))
  proof(auto simp add: inverse-rel-def)
    show f ;; g ≈> (Cinv g) ;; (Cinv f)
    proof(rule CompDefinedI)
      show f ;; g ∈ mor using d by auto
      show (Cinv g) ;; (Cinv f) ∈ mor using c by auto
      have cod (f ;; g) = cod g using d by auto
      also have ... = dom (Cinv g) using assms by (simp add: InvDomCod)
      also have ... = dom ((Cinv g) ;; (Cinv f)) using b by auto
      finally show cod (f ;; g) = dom ((Cinv g) ;; (Cinv f)) .
    qed
    show f ;; g ;; ((Cinv g) ;; (Cinv f)) = id (dom (f ;; g))
    proof−
      have e: g ≈> (Cinv g) using assms by (simp add: IsoCompInv)
      have f: f ∈ mor using 1 by auto
      have (f ;; g) ;; ((Cinv g) ;; (Cinv f)) = (f ;; (g ;; (Cinv g))) ;; (Cinv f)
        using 1 e a by (auto simp add: Cassoc CompDefComp)
     also have ... = f ;; (id (dom g)) ;; (Cinv f) using 3 by (simp add: IsoInvId2)
      also have ... = f ;; id (cod f) ;; (Cinv f) using 1 by (auto simp add: Simps)
      also have ... = f ;; (Cinv f) using f by (auto simp add: Cidr)
      also have ... = id (dom f) using 2 by (simp add: IsoInvId2)
      also have ... = id (dom (f ;; g)) using d by auto
      finally show ?thesis by simp
    qed
    show ((Cinv g) ;; (Cinv f)) ;; (f ;; g) = id (cod (f ;; g))
    proof−
      have e: (Cinv f) ≈> f using assms by (simp add: InvCompIso)
      have f: g ∈ mor using 1 by auto
```

```
    have ((Cinv g) ;; (Cinv f)) ;; (f ;; g) = (Cinv g) ;; (((Cinv f) ;; f) ;; g)
      using 1 e a by (auto simp add: Cassoc CompDefComp)
   also have ... = (Cinv g) ;; ((id (cod f)) ;; g) using 2 by (simp add: IsoInvId1)
    also have ... = (Cinv g) ;; ((id (dom g)) ;; g) using 1 by (auto simp add:
Simps)
    also have ... = (Cinv g) ;; g using f by (auto simp add: Cidl)
    also have ... = id (cod g) using 3 by (simp add: IsoInvId1)
    also have ... = id (cod (f ;; g)) using d by auto
    finally show ?thesis by simp
  qed
 qed
 thus ciso (f ;; g) and Cinv (f ;; g) = (Cinv g) ;; (Cinv f) by (auto simp add:
inv2Inv)
qed

definition ObjIso C A B ≡ ∃ k . (k maps_C A to B) ∧ ciso_C k

definition
 UnitCategory :: (unit, unit) Category where
 UnitCategory = MakeCat (|
    Obj = {()} ,
    Mor = {()} ,
    Dom = (λf.()) ,
    Cod = (λf.()) ,
    Id = (λf.()) ,
    Comp = (λf g. ())
 |)

lemma [simp]: Category(UnitCategory)
apply (simp add: UnitCategory-def, rule MakeCat)
by (unfold-locales, auto simp add: UnitCategory-def)

definition
  OppositeCategory :: ('o,'m,'a) Category-scheme ⇒ ('o,'m,'a) Category-scheme
(‹Op -› [65] 65) where
 OppositeCategory C ≡ (|
    Obj = Obj C ,
    Mor = Mor C ,
    Dom = Cod C ,
    Cod = Dom C ,
    Id  = Id C ,
    Comp = (λf g. g ;;_C f),
    ... = Category.more C
 |)

lemma OpCatOpCat: Op (Op C) = C
 by (simp add: OppositeCategory-def)
```

lemma *OpCatCatAx*: *Category-axioms C* $\Longrightarrow$ *Category-axioms* (*Op C*)
by (*simp add*: *OppositeCategory-def Category-axioms-def MapsTo-def CompDefined-def*)

lemma *OpCatCatExt*: *ExtCategory C* $\Longrightarrow$ *ExtCategory* (*Op C*)
by (*auto simp add*: *OppositeCategory-def ExtCategory-def MapsTo-def CompDefined-def extensional-def*)

lemma *OpCatCat*: *Category C* $\Longrightarrow$ *Category* (*Op C*)
by (*intro-locales*, *simp-all add*: *Category-def OpCatCatAx OpCatCatExt*)

lemma *MapsToOp*: *f maps*$_C$ *X to Y* $\Longrightarrow$ *f maps*$_{Op\ C}$ *Y to X*
by (*simp add*: *MapsTo-def OppositeCategory-def*)

lemma *MapsToOpOp*: *f maps*$_{Op\ C}$ *X to Y* $\Longrightarrow$ *f maps*$_C$ *Y to X*
by (*simp add*: *MapsTo-def OppositeCategory-def*)

lemma *CompDefOp*: *f* $\approx>_C$ *g* $\Longrightarrow$ *g* $\approx>_{Op\ C}$ *f*
by (*simp add*: *CompDefined-def OppositeCategory-def*)

end

2 Universe

theory *Universe*
imports *HOL−ZF.MainZF*
begin

locale *Universe* =
fixes *U* :: *ZF* (**structure**)
assumes *Uempty* : *Elem Empty U*
and *Usubset* : *Elem u U* $\Longrightarrow$ *subset u U*
and *Usingle* : *Elem u U* $\Longrightarrow$ *Elem* (*Singleton u*) *U*
and *Upow* : *Elem u U* $\Longrightarrow$ *Elem* (*Power u*) *U*
and *Uim* : $[\![$*Elem I U* ; *Elem u* (*Fun I U*) $]\!]$ $\Longrightarrow$ *Elem* (*Sum* (*Range u*)) *U*
and *Unat* : *Elem Nat U*

lemma *ElemLambdaFun* : ($\bigwedge$ *x* .*Elem x u* $\Longrightarrow$ *Elem* (*f x*) *U*) $\Longrightarrow$ *Elem* (*Lambda u f*) (*Fun u U*)
apply (*subst Elem-Lambda-Fun*)
apply *simp*
done

lemma *RangeRepl*: *Range* (*Lambda A f*) = *Repl A f*
apply (*subst Ext*)
apply (*subst Range*)

apply (*simp add*: *Repl Opair Lambda-def*)
done

lemma (**in** *Universe*) *Utrans*: ⟦*Elem a U* ; *Elem b a*⟧ ⟹ *Elem b U*
apply (*drule Usubset*)
apply (*insert HOLZF.subset-def* [*of a U*])
apply (*auto simp add*: *Usubset*)
done

lemma *ReplId*: *Repl A id* = *A*
by (*subst Ext*, *simp add*: *Repl*)

lemma (**in** *Universe*) *UniverseSum* : *Elem u U* ⟹ *Elem* (*Sum u*) *U*
apply (*frule-tac u* = *Lambda u id* **in** *Uim*)
apply (*subst Elem-Lambda-Fun*)
apply (*frule Usubset*)
apply (*simp add*: *subset-def*)
apply (*simp only*: *RangeRepl ReplId*)
done

lemma (**in** *Universe*) *UniverseUnion*:
  assumes *Elem u U* **and** *Elem v U*
  shows *Elem* (*union u v*) *U*
proof−
  let *?f* = (% *x*. *if x* = *Empty then u else v*)
    and *?S* = (*Power* (*Power Empty*))
  have *1*: (*Upair u v*) = *Range* (*Lambda ?S ?f*)
    by (*subst RangeRepl*, *simp add*: *Upair-def*)
  have *2*: ⟦*Elem u U*; *Elem v U*⟧ ⟹ *Elem* (*Lambda ?S ?f*) (*Fun ?S U*)
    by (*rule ElemLambdaFun*, *simp*)
  show *?thesis* **using** *assms*
    apply (*subst HOLZF.union-def*)
    apply (*subst 1*)
    apply (*rule-tac I*=*?S* **in** *Uim*)
    apply (*rule Upow*)+
    apply (*rule Uempty*)
    apply (*rule 2*)
    apply *simp*+
    done
qed

lemma *UPairSingleton*: *Upair u v* = *union* (*Singleton u*) (*Singleton v*)
apply (*subst Ext*)
apply (*subst Upair*)
apply (*subst union*)
apply (*subst Singleton*)+
apply (*simp*)
done

lemma (**in** *Universe*) *UniverseUPair*: ⟦*Elem u U* ; *Elem v U*⟧ ⟹ *Elem* (*Upair u v*) *U*
apply (*subst UPairSingleton*)
apply (*rule UniverseUnion*)
apply (*rule Usingle*, *simp*)+
done

lemma (**in** *Universe*) *UniversePair*: ⟦*Elem u U* ; *Elem v U*⟧ ⟹ *Elem* (*Opair u v*) *U*
apply (*subst Opair-def*)
apply ((*rule UniverseUPair*)+, *simp*+)+
done

lemma (**in** *Universe*) ⟦*Elem u U* ; *Elem v U*⟧ ⟹ *Elem* (*Sum* (*Repl u* (%*x* . *Singleton* (*Opair x v*)))) *U*
apply (*rule RangeRepl* [*THEN subst*])
apply (*rule Uim* [*of u*], *simp*)
apply (*rule ElemLambdaFun*)
apply (*rule Usingle*)
apply (*rule UniversePair*)
apply (*rule Utrans*)
apply *simp*+
done

lemma *SumRepl*: *Sum* (*Repl A* (*Singleton o f*)) = *Repl A f*
proof−
show *?thesis*
apply (*subst Ext*)
apply (*subst Sum*)
apply (*subst Repl*)+
apply (*auto simp add*: *Singleton*)
proof−
fix *a*
show *Elem a A* ⟹ ∃ *y*. *Elem* (*f a*) *y* ∧ (∃ *a*. *Elem a A* ∧ *y* = *Singleton* (*f a*))
apply (*rule exI* [*of* - *Singleton* (*f a*)])
apply (*subst Singleton*, *simp*+)
apply (*rule exI* [*of* - *a*], *simp*+)
done
qed
qed

lemma (**in** *Universe*) *UniverseProd*:
assumes *Elem u U* **and** *Elem v U*
shows *Elem* (*CartProd u v*) *U*
proof−
show *?thesis* **using** *assms*
apply (*subst CartProd-def*)

```
  apply (rule RangeRepl [of u % x . (Repl v (Opair x)), THEN subst])
  apply (rule Uim [of u], simp)
  apply (rule ElemLambdaFun)
  proof−
   fix x
   show ⟦Elem u U; Elem v U; Elem x u⟧ ⟹ Elem (Repl v (Opair x)) U
    apply (drule Utrans [of u x], simp)
    apply (rule SumRepl [THEN subst])
    apply (rule RangeRepl [THEN subst])
    apply (rule Uim [of v], simp)
    apply (rule ElemLambdaFun,simp)
    apply (rule Usingle)
    apply (rule UniversePair)
    apply (drule Usubset, simp)
    proof−
     fix xa
     show ⟦Elem v U; Elem x u; Elem x U; Elem xa v⟧ ⟹ Elem xa U
      by (rule Utrans, simp+)
    qed
   qed
  qed

lemma (in Universe) UniverseSubset: ⟦subset u v ; Elem v U⟧ ⟹ Elem u U
 apply (drule-tac HOLZF.Power [of u v, THEN ssubst])
 apply (drule Upow)
 apply (rule Utrans, simp+)
 done

definition
 Product :: ZF ⇒ ZF where
 Product U = Sep (Fun U (Sum U)) (%f . (∀ u . Elem u U ⟶ Elem (app f u)
u))

lemma SepSubset: subset (Sep A p) A
apply (subst subset-def)
apply (subst Sep, simp)
done

lemma SubsetSmall:
 assumes subset A' A and subset A B shows subset A' B
 proof−
  have (subset A' A ∧ subset A B) ⟶ subset A' B
   by ((subst subset-def)+, simp+)
  thus ?thesis using assms by simp
 qed

lemma SubsetTrans:
 assumes (subset a b) and (subset b c)
 shows (subset a c)
```

proof−
have (*subset a b*) ∧ (*subset b c*) ⟶ (*subset a c*)
by ((*subst subset-def*)+, *simp*)
thus *?thesis* **using** *assms* **by** *simp*
qed

lemma *SubsetSepTrans*: *subset A B* ⟹ *subset* (*Sep A p*) *B*
apply (*rule SubsetSmall* [*of Sep A p A B*])
apply (*rule SepSubset*)
by *simp*

lemma *ProductSubset*: *subset* (*Product u*) (*Power* (*CartProd u* (*Sum u*)))
apply (*subst Product-def*)
apply (*subst Fun-def*)
apply (*subst PFun-def*)
apply (*rule SubsetSepTrans*)+
apply (*subst subset-def*)
by *simp*

lemma (**in** *Universe*) *UniverseProduct*: *Elem u U* ⟹ *Elem* (*Product u*) *U*
apply (*rule-tac u*=(*Product u*) **and** *v*=*Power* (*CartProd u* (*Sum u*)) **in** *UniverseSubset*)
apply (*rule ProductSubset*)
apply (*rule Upow*)
apply (*rule UniverseProd*, *simp*)
apply (*rule UniverseSum*,*simp*)
done

lemma *ZFImageRangeExplode*: *x* ∈ *range explode* ⟹ *f* ‘ *x* ∈ *range explode*
proof−
assume *x* ∈ *range explode*
from *this* **obtain** *y* **where** *x* = *explode y* **using** *range-ex1-eq* **by** *auto*
hence *f* ‘ *x* = *explode* (*Repl y f*) **using** *explode-Repl-eq* **by** *simp*
thus *f* ‘ *x* ∈ *range explode* **by** *auto*
qed

definition *subsetFn X Y* ≡ λ *x* . (*if x* ∈ *Y then x else SOME y* . *y* ∈ *Y*)

lemma *subsetFn*: ⟦*Y* ≠ {} ; *Y* ⊆ *X* ⟧ ⟹ (*subsetFn X Y*) ‘ *X* = *Y*
proof(*auto simp add*: *subsetFn-def*)
fix *x* **assume** *x* ∈ *Y*
thus (*SOME y*. *y* ∈ *Y*) ∈ *Y* **using** *someI-ex*[*of* λ *x* . *x* ∈ *Y*] **by** *auto*
qed

lemma *ZFSubsetRangeExplode*: ⟦*X* ∈ *range explode* ; *Y* ⊆ *X*⟧ ⟹ *Y* ∈ *range explode*
proof(*cases Y* = {}, *simp*)
have *explode Empty* = {} **using** *explode-Empty* **by** *simp*
thus {} ∈ *range explode* **by** (*auto simp add*: *explode-def*)

assume $Y \neq \{\}$ **and** $Y \subseteq X$ **and** $X \in$ *range explode* **thus** $Y \in$ *range explode*
using *ZFImageRangeExplode*[*of X subsetFn X Y*] *subsetFn*[*of Y X*] **by** *simp*
qed

lemma *ZFUnionRangeExplode*:
assumes $\bigwedge x . x \in A \Longrightarrow f\ x \in$ *range explode* **and** $A \in$ *range explode*
shows $(\bigcup x \in A . f\ x) \in$ *range explode*
proof−
let *?S* = *Sep* (*Sum* (*Repl* (*implode A*) (*implode o f*))) (λ *y* . $\exists$ *x* . (*Elem x* (*implode A*)) $\wedge$ (*Elem y* (*implode* (*f x*))))
have *explode ?S* = $(\bigcup x \in A . f\ x)$
proof (*auto simp add*: *UNION-eq explode-def Sep Sum Repl assms Elem-implode cong del*: *image-cong-simp*)
fix *x y* **assume** *a*: $y \in A$ **and** *b*: $x \in f\ y$
show $\exists z.$ *Elem x z* $\wedge$ ($\exists a.\ a \in A \wedge z =$ *implode* (*f a*))
apply (*rule exI* [**where** *?x* = *implode* (*f y*)])
apply (*auto simp add*: *explode-def Sep Sum Repl assms Elem-implode a b cong del*: *image-cong-simp*)
apply (*rule exI* [**where** *?x* = *y*])
apply (*simp add*: *a*)
done
qed
thus *?thesis* **by** *auto*
qed

lemma *ZFUnionNatInRangeExplode*: $(\bigwedge (n :: nat) . f\ n \in$ *range explode*$) \Longrightarrow (\bigcup n . f\ n) \in$ *range explode*
proof−
assume *a*: $(\bigwedge (n :: nat) . f\ n \in$ *range explode*$)$
have *explode Nat* $\in$ *range explode* **by** *simp*
moreover have $\bigwedge n . n \in$ (*explode Nat*) $\Longrightarrow$ (*f o Nat2nat*) *n* $\in$ *range explode* **using** *a*
by(*auto simp add*: *explode-def*)
moreover have $(\bigcup n . f\ n) = (\bigcup n \in$ (*explode Nat*) . (*f o Nat2nat*) *n*)
proof(*auto simp add*: *Nat2nat-def*)
fix *x n* **assume** *aa*: $x \in f\ n$ **show** $\exists n \in$ (*explode Nat*) . $x \in f$ (*inv nat2Nat n*)
apply(*rule bexI*[**where** *?x* = *nat2Nat n*])
by(*auto simp add*: *aa inj-nat2Nat explode-Elem*)
qed
ultimately show $(\bigcup n . f\ n) \in$ *range explode* **using** *ZFUnionRangeExplode* **by** *simp*
qed

lemma *ZFProdFnInRangeExplode*: $[\![A \in$ *range explode* ; $B \in$ *range explode*$]\!] \Longrightarrow f$ ‘ $(A \times B) \in$ *range explode*
proof−
assume *a*: $A \in$ *range explode* **and** *b*: $B \in$ *range explode*
let *?X* = (*explode* (*CartProd* (*implode A*) (*implode B*)))

have f ‘ $(A \times B) = (f \;\; o \; (\lambda \; z \; . \; (Fst \; z, \; Snd \; z)))$ ‘ *?X*
proof(*auto simp add*: *explode-def CartProd image-def Fst Snd*)
{
fix *z y* **assume** *z*: $z \in A$ **and** *y*: $y \in B$ **show** $\exists \, x. \; (\exists \, a. \; Elem \; a \; (implode \; A) \wedge$
$(\exists \, b. \; Elem \; b \; (implode \; B) \wedge x = Opair \; a \; b)) \wedge f \; (z, \; y) = f \; (Fst \; x, \; Snd \; x)$
apply(*insert z y a b*)
apply(*rule exI*[**where** *?x* = *Opair z y*])
apply(*auto simp add*: *Opair explode-Elem Fst Snd*)
done
}
{
fix *a b* **assume** *aa*: *Elem a* (*implode A*) **and** *bb*: *Elem b* (*implode B*)
show $\exists \;\; x \in A \; . \; \exists \;\; y \in B \; . \; f \; (a,b) = f \; (x,y)$
by(*rule bexI*[**where** *?x* = *a*], *rule bexI*[**where** *?x* = *b*], *simp*, *insert a b aa bb*, *auto simp add*: *explode-Elem*)
}
qed
moreover have *?X* $\in$ *range explode* **by** *simp*
ultimately show f ‘ $(A \times B) \in$ *range explode* **using** *ZFImageRangeExplode* **by** *simp*
qed

lemma *ZFUnionInRangeExplode*: $[\![A \in range \; explode \; ; \; B \in range \; explode]\!] \Longrightarrow A \cup B \in range \; explode$
proof−
assume $A \in range \; explode$ **and** $B \in range \; explode$
from *this* **obtain** A' B' **where** A': $A = explode \; A'$ **and** B': $B = explode \; B'$ **by** *auto*
have $A \cup B = explode \; (union \; (implode \; A) \; (implode \; B))$
by(*auto simp add*: *explode-union union explode-Elem* A' B')
thus $A \cup B \in range \; explode$ **by** *auto*
qed

lemma *SingletonInRangeExplode*: $\{x\} \in range \; explode$
proof−
have *explode* (*Singleton x*) = $\{x\}$ **by**(*auto simp add*: *explode-def Singleton*)
thus *?thesis* **by** *auto*
qed

definition *ZFTriple* :: [*ZF*,*ZF*,*ZF*] ⇒ *ZF* **where**
ZFTriple a b c = *Opair* (*Opair a b*) *c*
definition *ZFTFst* = *Fst o Fst*
definition *ZFTSnd* = *Snd o Fst*
definition *ZFTThd* = *Snd*

lemma *ZFTFst*: *ZFTFst* (*ZFTriple a b c*) = *a*
by(*auto simp add*: *ZFTriple-def ZFTFst-def Fst*)
lemma *ZFTSnd*: *ZFTSnd* (*ZFTriple a b c*) = *b*
by(*auto simp add*: *ZFTriple-def ZFTSnd-def Snd Fst*)

lemma *ZFTThd*: *ZFTThd* (*ZFTriple a b c*) = *c*
by(*auto simp add*: *ZFTriple-def ZFTThd-def Snd Fst*)

lemma *ZFTriple*: *ZFTriple a b c* = *ZFTriple a′ b′ c′* ⟹ (*a* = *a′* ∧ *b* = *b′* ∧ *c* = *c′*)
by(*auto simp add*: *ZFTriple-def Opair*)

lemma *ZFSucZero*: *Nat2nat* (*SucNat Empty*) = *1*
proof−
have *nat2Nat 0* = *Empty* **by** *auto*
hence (*SucNat Empty*) = *nat2Nat* (*Suc 0*) **by** *auto*
hence *Nat2nat* (*SucNat Empty*) = *Nat2nat* (*nat2Nat* (*Suc 0*)) **by** *simp*
also have ... = *Suc 0* **using** *Nat2nat-nat2Nat*[*of Suc 0*] **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *ZFZero*: *Nat2nat Empty* = *0*
proof−
have *nat2Nat 0* = *Empty* **by** *auto*
hence *Nat2nat Empty* = *Nat2nat* (*nat2Nat 0*) **by** *simp*
thus *?thesis* **using** *Nat2nat-nat2Nat*[*of 0*] **by** *simp*
qed

lemma *ZFSucNeq0*: *Elem x Nat* ⟹ *Nat2nat* (*SucNat x*) ≠ *0*
by(*auto simp add*: *Nat2nat-SucNat*)

end

3 Monadic Equational Theory

theory *MonadicEquationalTheory*
imports *Category Universe*
begin

record (*′t*,*′f*) *Signature* =
BaseTypes :: *′t set* (‹*Ty*₁›)
BaseFunctions :: *′f set* (‹*Fn*₁›)
SigDom :: *′f* ⇒ *′t* (‹*sDom*₁›)
SigCod :: *′f* ⇒ *′t* (‹*sCod*₁›)

locale *Signature* =
fixes *S* :: (*′t*,*′f*) *Signature* (**structure**)
assumes *Domt*: *f* ∈ *Fn* ⟹ *sDom f* ∈ *Ty*
and *Codt*: *f* ∈ *Fn* ⟹ *sCod f* ∈ *Ty*

definition *funsignature-abbrev* (‹- ∈ *Sig* - : - → -›) **where**
f ∈ *Sig S* : *A* → *B* ≡ *f* ∈ (*BaseFunctions S*) ∧ *A* ∈ (*BaseTypes S*) ∧ *B* ∈ (*BaseTypes S*) ∧
(*SigDom S f*) = *A* ∧ (*SigCod S f*) = *B* ∧ *Signature S*

lemma *funsignature-abbrevE*[*elim*]:
$[\![f \in Sig\ S : A \to B\ ;\ [\![f \in (BaseFunctions\ S)\ ;\ A \in (BaseTypes\ S)\ ;\ B \in (BaseTypes\ S)\ ;$
$(SigDom\ S\ f) = A\ ;\ (SigCod\ S\ f) = B\ ;\ Signature\ S]\!] \Longrightarrow R]\!] \Longrightarrow R$
by (*simp add*: *funsignature-abbrev-def*)

datatype ('*t*,'*f*) *Expression* = *ExprVar* (‹*Vx*›) | *ExprApp* '*f* ('*t*,'*f*) *Expression* (‹- *E@* ->)
datatype ('*t*,'*f*) *Language* = *Type* '*t* (‹⊢ - *Type*›) | *Term* '*t* ('*t*,'*f*) *Expression* '*t* (‹*Vx* : - ⊢ - : ->) |
Equation '*t* ('*t*,'*f*) *Expression* ('*t*,'*f*) *Expression* '*t* (‹*Vx* : - ⊢ -≡- : ->)

inductive
WellDefined :: ('*t*,'*f*) *Signature* ⇒ ('*t*,'*f*) *Language* ⇒ *bool* (‹*Sig* - ▷ ->) **where**
WellDefinedTy: $A \in BaseTypes\ S \Longrightarrow Sig\ S \triangleright \vdash A\ Type$
| *WellDefinedVar*: $Sig\ S \triangleright \vdash A\ Type \Longrightarrow Sig\ S \triangleright (Vx : A\ \vdash Vx : A)$
| *WellDefinedFn*: $[\![Sig\ S \triangleright (Vx : A\ \vdash e : B)\ ;\ f \in Sig\ S : B \to C]\!] \Longrightarrow Sig\ S \triangleright (Vx : A\ \vdash (f\ E@\ e) : C)$
| *WellDefinedEq*: $[\![Sig\ S \triangleright (Vx : A\ \vdash e1 : B)\ ;\ Sig\ S \triangleright (Vx : A\ \vdash e2 : B)]\!] \Longrightarrow Sig\ S \triangleright (Vx : A \vdash e1 \equiv e2 : B)$

lemmas *WellDefined.intros* [*intro*]
inductive-cases *WellDefinedTyE* [*elim*!]: $Sig\ S \triangleright \vdash A\ Type$
inductive-cases *WellDefinedVarE* [*elim*!]: $Sig\ S \triangleright (Vx : A\ \vdash Vx : A)$
inductive-cases *WellDefinedFnE* [*elim*!]: $Sig\ S \triangleright (Vx : A\ \vdash (f\ E@\ e) : C)$
inductive-cases *WellDefinedEqE* [*elim*!]: $Sig\ S \triangleright (Vx : A \vdash e1 \equiv e2 : B)$

lemma *SigId*: $Sig\ S \triangleright (Vx : A\ \vdash Vx : B) \Longrightarrow A = B$
apply(*rule WellDefined.cases*)
by *simp*+

lemma *SigTyId*: $Sig\ S \triangleright (Vx : A\ \vdash Vx : A) \Longrightarrow A \in BaseTypes\ S$
apply(*rule WellDefined.cases*)
by *auto*

lemma (**in** *Signature*) *SigTy*: $\bigwedge B\ .\ Sig\ S \triangleright (Vx : A\ \vdash e : B) \Longrightarrow (A \in BaseTypes\ S \wedge B \in BaseTypes\ S)$
proof(*induct e*)
{
fix *B* **assume** *a*: $Sig\ S \triangleright Vx : A \vdash Vx : B$
have $A = B$ **using** *a SigId*[*of S*] **by** *simp*
thus $A \in Ty \wedge B \in Ty$ **using** *a* **by** *auto*
}
{
fix *B f e* **assume** *ih*: $\bigwedge B'.\ Sig\ S \triangleright (Vx : A \vdash e : B') \Longrightarrow A \in Ty \wedge B' \in Ty$
and *a*: $Sig\ S \triangleright (Vx : A \vdash (f\ E@\ e) : B)$

from *a* **obtain** B' **where** *f*: $f \in Sig\ S : B' \to B$ **and** $Sig\ S \triangleright (Vx : A \vdash e : B')$ **by** *auto*
hence $A \in Ty$ **using** *ih* **by** *auto*
moreover have $B \in Ty$ **using** *f* **by** (*auto simp add*: *funsignature-abbrev-def Codt*)
ultimately show $A \in Ty \wedge B \in Ty$ **by** *simp*
}
qed

datatype ('o,'m) *IType* = *IObj* 'o | *IMor* 'm | *IBool bool*

record ('t,'f,'o,'m) *Interpretation* =
ISignature :: ('t,'f) *Signature* (‹*iS*1›)
ICategory :: ('o,'m) *Category* (‹*iC*1›)
ITypes :: 't ⇒ 'o (‹*Ty*⟦-⟧1›)
IFunctions :: 'f ⇒ 'm (‹*Fn*⟦-⟧1›)

locale *Interpretation* =
fixes *I* :: ('t,'f,'o,'m) *Interpretation* (**structure**)
assumes *ICat*: *Category iC*
and *ISig*: *Signature iS*
and *It* : $A \in BaseTypes\ iS \Longrightarrow Ty⟦A⟧ \in Obj\ iC$
and *If* : $(f \in Sig\ iS : A \to B) \Longrightarrow Fn⟦f⟧\ maps_{iC}\ Ty⟦A⟧\ to\ Ty⟦B⟧$

inductive *Interp* (‹$L⟦$-$⟧_1 \to$ -›) **where**
InterpTy: $Sig\ iS_I \triangleright \vdash A\ Type \Longrightarrow$
$L⟦\vdash A\ Type⟧_I \to (IObj\ Ty⟦A⟧_I)$
| *InterpVar*: $L⟦\vdash A\ Type⟧_I \to (IObj\ c) \Longrightarrow$
$L⟦Vx : A\ \vdash Vx : A⟧_I \to (IMor\ (Id\ iC_I\ c))$
| *InterpFn*: $⟦Sig\ iS_I \triangleright Vx : A\ \vdash e : B$;
$f \in Sig\ iS_I : B \to C$;
$L⟦Vx : A\ \vdash e : B⟧_I \to (IMor\ g)⟧ \Longrightarrow$
$L⟦Vx : A\ \vdash (f\ E@\ e) : C⟧_I \to (IMor\ (g\ ;;_{ICategory\ I}\ Fn⟦f⟧_I))$
| *InterpEq*: $⟦L⟦Vx : A\ \vdash e1 : B⟧_I \to (IMor\ g1)$;
$L⟦Vx : A\ \vdash e2 : B⟧_I \to (IMor\ g2)⟧ \Longrightarrow$
$L⟦Vx : A\ \vdash e1 \equiv e2 : B⟧_I \to (IBool\ (g1 = g2))$

lemmas *Interp.intros* [*intro*]
inductive-cases *InterpTyE* [*elim*!]: $L⟦\vdash A\ Type⟧_I \to i$
inductive-cases *InterpVarE* [*elim*!]: $L⟦Vx : A\ \vdash Vx : A⟧_I \to i$
inductive-cases *InterpFnE* [*elim*!]: $L⟦Vx : A\ \vdash (f\ E@\ e) : C⟧_I \to i$
inductive-cases *InterpEqE* [*elim*!]: $L⟦Vx : A\ \vdash e1 \equiv e2 : B⟧_I \to i$

lemma (**in** *Interpretation*) *InterpEqEq*[*intro*]:
$⟦L⟦Vx : A\ \vdash e1 : B⟧ \to (IMor\ g)$; $L⟦Vx : A\ \vdash e2 : B⟧ \to (IMor\ g)⟧ \Longrightarrow L⟦Vx : A\ \vdash e1 \equiv e2 : B⟧ \to (IBool\ True)$
proof−
assume *a*: $L⟦Vx : A\ \vdash e1 : B⟧ \to (IMor\ g)$ **and** *b*: $L⟦Vx : A\ \vdash e2 : B⟧ \to$

(*IMor g*)
thus *?thesis* **using** *InterpEq*[*of I A e1 B g e2 g*] **by** *simp*
qed

lemma (**in** *Interpretation*) *InterpExprWellDefined*:
$L[\![Vx : A \vdash e : B]\!] \to i \Longrightarrow Sig\ iS \triangleright Vx : A \vdash e : B$
apply (*rule Interp.cases*)
by *auto*

lemma (**in** *Interpretation*) *WellDefined*: $L[\![\varphi]\!] \to i \Longrightarrow Sig\ iS \triangleright \varphi$
apply(*rule Interp.cases*)
by (*auto simp add*: *InterpExprWellDefined*)

lemma (**in** *Interpretation*) *Bool*: $L[\![\varphi]\!] \to (IBool\ i) \Longrightarrow \exists\ A\ B\ e\ d\ .\ \varphi = (Vx : A \vdash e \equiv d : B)$
apply(*rule Interp.cases*)
by *auto*

lemma (**in** *Interpretation*) *FunctionalExpr*:
$\bigwedge i\ j\ A\ B. [\![L[\![Vx : A \vdash e : B]\!] \to i;\ L[\![Vx : A \vdash e : B]\!] \to j]\!] \Longrightarrow i = j$
apply (*induct e*)
apply (*rule Interp.cases*)
apply *auto*
apply (*auto simp add*: *funsignature-abbrev-def*)
done

lemma (**in** *Interpretation*) *Functional*: $[\![L[\![\varphi]\!] \to i1\ ;\ L[\![\varphi]\!] \to i2]\!] \Longrightarrow i1 = i2$
proof(*induct* φ)
{
fix *t* **show** $[\![L[\![\vdash t\ Type]\!] \to i1\ ;\ L[\![\vdash t\ Type]\!] \to i2]\!] \Longrightarrow i1 = i2$ **by** *auto*
}
{
fix *t1 t2 e*
show $[\![L[\![Vx : t1 \vdash e : t2]\!] \to i1;\ L[\![Vx : t1 \vdash e : t2]\!] \to i2]\!] \Longrightarrow i1 = i2$ **by** (*simp add*: *FunctionalExpr*)
}
{
fix *t1 t2 e1 e2* **show** $[\![L[\![Vx : t1 \vdash e1 \equiv e2 : t2]\!] \to i1;\ L[\![Vx : t1 \vdash e1 \equiv e2 : t2]\!] \to i2]\!] \Longrightarrow i1 = i2$
proof(*auto*)
{
fix *f g h* **assume** *f1*: $L[\![Vx : t1 \vdash e1 : t2]\!] \to (IMor\ f)$ **and** *f2*: $L[\![Vx : t1 \vdash e2 : t2]\!] \to (IMor\ f)$
and *g1*: $L[\![Vx : t1 \vdash e1 : t2]\!] \to (IMor\ g)$ **and** *g2*: $L[\![Vx : t1 \vdash e2 : t2]\!] \to (IMor\ h)$
have $f = g$ **using** *f1 g1 FunctionalExpr*[*of t1 e1 t2 IMor f IMor g*] **by** *simp*
moreover have $f = h$ **using** *f2 g2 FunctionalExpr*[*of t1 e2 t2 IMor f IMor h*] **by** *simp*
ultimately show $g = h$ **by** *simp*

```
    }
    moreover
    {
     fix f g h assume f1: L⟦Vx : t1 ⊢ e1 : t2⟧ → (IMor f) and f2: L⟦Vx : t1 ⊢
e2 : t2⟧ → (IMor g)
       and g1: L⟦Vx : t1 ⊢ e1 : t2⟧ → (IMor h) and g2: L⟦Vx : t1 ⊢ e2 : t2⟧
→ (IMor h)
     have f = h using f1 g1 FunctionalExpr[of t1 e1 t2 IMor f IMor h] by simp
     moreover have g = h using f2 g2 FunctionalExpr[of t1 e2 t2 IMor g IMor
h] by simp
     ultimately show f = g by simp
    }
   qed
  }
qed

lemma (in Interpretation) MorphismsPreserved:
⋀ B i . L⟦Vx : A ⊢ e : B⟧ → i ⟹ ∃ g . i = (IMor g) ∧ (g maps_iC Ty⟦A⟧ to
Ty⟦B⟧)
proof(induct e)
  {
   fix B i assume a: L⟦Vx : A ⊢ Vx : B⟧ → i show ∃ g. i = IMor g ∧ g maps_iC
Ty⟦A⟧ to Ty⟦B⟧
   proof(rule exI[where ?x=Id iC Ty⟦A⟧], rule conjI)
    have sig: Sig iS ▷ Vx : A ⊢ Vx : B using a by (simp add: InterpExprWellDe-
fined)
     hence aEqb: A = B by (simp add: SigId)
     have ty: A ∈ BaseTypes iS using aEqb sig SigTyId by simp
     hence L⟦Vx : A ⊢ Vx : A⟧ → (IMor (Id iC Ty⟦A⟧)) by auto
     thus i = IMor (Category.Id iC Ty⟦A⟧) using a aEqb Functional by simp
      show (Id iC Ty⟦A⟧) maps_iC Ty⟦A⟧ to Ty⟦B⟧ using aEqb ICat It[of A]
Category.Cidm[of iC Ty⟦A⟧] ty by simp
   qed
  }
  {
   fix f e B i assume a: ⋀ B i . L⟦Vx : A ⊢ e : B⟧ → i ⟹ ∃g. i = IMor g ∧ g
maps_iC Ty⟦A⟧ to Ty⟦B⟧
          and b: L⟦Vx : A ⊢ f E@ e : B⟧ → i
   show ∃ g. i = IMor g ∧ g maps_iC Ty⟦A⟧ to Ty⟦B⟧ using a b
   proof−
    from b obtain g B′ where g: (Sig iS ▷ Vx : A ⊢ e : B′) ∧ (f ∈ Sig iS : B′
→ B) ∧ (L⟦Vx : A ⊢ e : B′⟧ → (IMor g))
      by auto
    from a have gmaps: g maps_iC Ty⟦A⟧ to Ty⟦B′⟧ using g by auto
    have fmaps: Fn⟦f⟧ maps_iC Ty⟦B′⟧ to Ty⟦B⟧ using g If by simp
    have (g ;;_iC Fn⟦f⟧) maps_iC Ty⟦A⟧ to Ty⟦B⟧ using ICat fmaps gmaps Cate-
gory.Ccompt[of iC g] by simp
    moreover have L⟦Vx : A ⊢ f E@ e : B⟧ → (IMor (g ;;_iC Fn⟦f⟧)) using g
by auto
```

ultimately show *?thesis* **using** *b Functional exI*[**where** *?x* = (*g* ;;$_{iC}$ *Fn*⟦*f*⟧)]
by *simp*
qed
}
qed

lemma (**in** *Interpretation*) *Expr2Mor*: *L*⟦*Vx* : *A* ⊢ *e* : *B*⟧ → (*IMor g*) ⟹ (*g maps*$_{iC}$ *Ty*⟦*A*⟧ *to Ty*⟦*B*⟧)
proof−
assume *a*: *L*⟦*Vx* : *A* ⊢ *e* : *B*⟧ → (*IMor g*)
from *MorphismsPreserved*[*of A e B* (*IMor g*)] **obtain** *g*′ **where** (*IMor g*) = (*IMor g*′) ∧ (*g*′ *maps*$_{iC}$ *Ty*⟦*A*⟧ *to Ty*⟦*B*⟧)
using *a* **by** *auto*
thus *?thesis* **by** *simp*
qed

lemma (**in** *Interpretation*) *WellDefinedExprInterp*: ⋀ *B* . (*Sig iS* ▷ *Vx* : *A* ⊢ *e* : *B*) ⟹ (∃ *i* . *L*⟦*Vx* : *A* ⊢ *e* : *B*⟧ → *i*)
proof(*induct e*)
{
fix *B* **assume** *sig*: (*Sig iS* ▷ *Vx* : *A* ⊢ *Vx* : *B*) **show** ∃ *i*. *L*⟦*Vx* : *A* ⊢ *Vx* : *B*⟧ → *i*
proof−
have *aEqb*: *A* = *B* **using** *sig* **by** (*simp add*: *SigId*)
hence *ty*: *A* ∈ *BaseTypes iS* **using** *sig SigTyId* **by** *simp*
hence *L*⟦*Vx* : *A* ⊢ *Vx* : *A*⟧ → (*IMor* (*Id iC Ty*⟦*A*⟧)) **by** *auto*
thus *?thesis* **using** *aEqb* **by** *auto*
qed
}
{
fix *f e B* **assume** *a*: ⋀ *B* . (*Sig iS* ▷ *Vx* : *A* ⊢ *e* : *B*) ⟹ (∃ *i* . *L*⟦*Vx* : *A* ⊢ *e* : *B*⟧ → *i*)
and *b*: (*Sig iS* ▷ *Vx* : *A* ⊢ *f E@ e* : *B*)
show ∃ *i* . *L*⟦*Vx* : *A* ⊢ *f E@ e* : *B*⟧ → *i*
proof−
from *b* **obtain** *B*′ **where** *B*′: (*Sig iS* ▷ (*Vx* : *A* ⊢ *e* : *B*′)) ∧ (*f* ∈ *Sig iS* : *B*′ → *B*) **by** *auto*
from *B*′ *a* **obtain** *i* **where** *i*: *L*⟦*Vx* : *A* ⊢ *e* : *B*′⟧ → *i* **by** *auto*
from *MorphismsPreserved*[*of A e B*′ *i*] *i* **obtain** *g* **where** *g*: *i* = (*IMor g*) **by** *auto*
thus *?thesis* **using** *B*′ *i* **by** *auto*
qed
}
qed

lemma (**in** *Interpretation*) *Sig2Mor*: **assumes** (*Sig iS* ▷ *Vx* : *A* ⊢ *e* : *B*) **shows** ∃ *g* . *L*⟦*Vx* : *A* ⊢ *e* : *B*⟧ → (*IMor g*)
proof−
from *WellDefinedExprInterp*[*of A e B*] *assms* **obtain** *i* **where** *i*: *L*⟦*Vx* : *A* ⊢ *e*

: B⟧ → i **by** *auto*
thus *?thesis* **using** *MorphismsPreserved*[*of A e B i*] i **by** *auto*
qed

record ('*t*,'*f*) *Axioms* =
aAxioms :: ('*t*,'*f*) *Language set*
aSignature :: ('*t*,'*f*) *Signature* (‹*aS*₁›)

locale *Axioms* =
fixes *Ax* :: ('*t*,'*f*) *Axioms* (**structure**)
assumes *AxT*: (*aAxioms Ax*) ⊆ {(*Vx* : *A* ⊢ *e1* ≡ *e2* : *B*) | *A B e1 e2* . *Sig* (*aSignature Ax*) ▷ (*Vx* : *A* ⊢ *e1* ≡ *e2* : *B*)}
assumes *AxSig*: *Signature* (*aSignature Ax*)

primrec *Subst* :: ('*t*,'*f*) *Expression* ⇒ ('*t*,'*f*) *Expression* ⇒ ('*t*,'*f*) *Expression* (‹*sub* - *in* -› [*81*,*81*] *81*) **where**
(*sub e in Vx*) = *e* | *sub e in* (*f E@ d*) = (*f E@* (*sub e in d*))

lemma *SubstXinE*: (*sub Vx in e*) = *e*
by(*induct e*, *auto simp add*: *Subst-def*)

lemma *SubstAssoc*: *sub a in* (*sub b in c*) = *sub* (*sub a in b*) *in c*
by(*induct c*, (*simp add*: *Subst-def*)+)

lemma *SubstWellDefined*: ⋀ *C* . ⟦*Sig S* ▷ (*Vx* : *A* ⊢ *e* : *B*); *Sig S* ▷ (*Vx* : *B* ⊢ *d* : *C*)⟧
⟹ *Sig S* ▷ (*Vx* : *A* ⊢ (*sub e in d*) : *C*)
proof(*induct d*)
{
fix *C* **assume** *a*: *Sig S* ▷ *Vx* : *A* ⊢ *e* : *B* **and** *b*: *Sig S* ▷ *Vx* : *B* ⊢ *Vx* : *C*
have *BCeq*: *B* = *C* **using** *b SigId*[*of S*] **by** *simp*
thus *Sig S* ▷ *Vx* : *A* ⊢ *sub e in Vx* : *C* **using** *a* **by** *simp*
}
{
fix *f d C* **assume** *ih*: ⋀*B'*. ⟦*Sig S* ▷ *Vx* : *A* ⊢ *e* : *B*; *Sig S* ▷ *Vx* : *B* ⊢ *d* : *B'*⟧
⟹ *Sig S* ▷ *Vx* : *A* ⊢ *sub e in d* : *B'* **and** *a*: *Sig S* ▷ *Vx* : *A* ⊢ *e* : *B*
and *b*: *Sig S* ▷ *Vx* : *B* ⊢ *f E@ d* : *C*
from *b* **obtain** *B'* **where** *B'*: *f* ∈ *Sig S* : *B'* → *C* **and** *d*: *Sig S* ▷ *Vx* : *B* ⊢ *d* : *B'* **by** *auto*
hence *Sig S* ▷ *Vx* : *A* ⊢ *sub e in d* : *B'* **using** *ih a* **by** *simp*
hence *Sig S* ▷ *Vx* : *A* ⊢ *f E@* (*sub e in d*) : *C* **using** *B'* **by** *auto*
thus *Sig S* ▷ *Vx* : *A* ⊢ (*sub e in* (*f E@ d*)) : *C* **by** *auto*
}
qed

inductive-set (**in** *Axioms*) *Theory* **where**
Ax: *A* ∈ (*aAxioms Ax*) ⟹ *A* ∈ *Theory*
| *Refl*: *Sig* (*aSignature Ax*) ▷ (*Vx* : *A* ⊢ *e* : *B*) ⟹ (*Vx* : *A* ⊢ *e* ≡ *e* : *B*) ∈ *Theory*
| *Symm*: (*Vx* : *A* ⊢ *e1* ≡ *e2* : *B*) ∈ *Theory* ⟹ (*Vx* : *A* ⊢ *e2* ≡ *e1* : *B*) ∈ *Theory*

| *Trans*: $[\![(Vx : A \vdash e1 \equiv e2 : B) \in Theory\ ;\ (Vx : A \vdash e2 \equiv e3 : B) \in Theory]\!] \Longrightarrow$
$(Vx : A \vdash e1 \equiv e3 : B) \in Theory$
| *Congr*: $[\![(Vx : A \vdash e1 \equiv e2 : B) \in Theory\ ;\ f \in Sig\ (aSignature\ Ax) : B \to C]\!] \Longrightarrow$
$(Vx : A \vdash (f\ E@\ e1) \equiv (f\ E@\ e2) : C) \in Theory$
| *Subst*: $[\![Sig\ (aSignature\ Ax) \triangleright (Vx : A \vdash e1 : B)\ ;\ (Vx : B \vdash e2 \equiv e3 : C) \in Theory]\!] \Longrightarrow$
$(Vx : A \vdash (sub\ e1\ in\ e2) \equiv (sub\ e1\ in\ e3) : C) \in Theory$

lemma (**in** *Axioms*) *Equiv2WellDefined*: $\varphi \in Theory \Longrightarrow Sig\ aS \triangleright \varphi$
proof(*rule Theory.induct*,*auto simp add*: *SubstWellDefined*)
{
fix φ **assume** *ax*: $\varphi \in aAxioms\ Ax$
from *AxT* **obtain** *A e1 e2 B* **where** $Sig\ aS \triangleright Vx : A \vdash e1 \equiv e2 : B$ **and** $\varphi = Vx : A \vdash e1 \equiv e2 : B$ **using** *ax* **by** *blast*
thus $Sig\ aS \triangleright \varphi$ **by** *simp*
}
qed

lemma (**in** *Axioms*) *Subst'*:
$\bigwedge C\ .\ [\![Sig\ aS \triangleright Vx : B \vdash d : C\ ;\ (Vx : A \vdash e1 \equiv e2 : B) \in Theory]\!] \Longrightarrow$
$(Vx : A \vdash (sub\ e1\ in\ d) \equiv (sub\ e2\ in\ d) : C) \in Theory$
proof(*induct d*)
{
fix *C* **assume** *a*: $Sig\ aS \triangleright Vx : B \vdash Vx : C$ **and** *b*: $(Vx : A \vdash e1 \equiv e2 : B) \in Theory$
have *BCeq*: $B = C$ **using** *a SigId*[*of aS B C*] **by** *simp*
thus $(Vx : A \vdash (sub\ e1\ in\ Vx) \equiv (sub\ e2\ in\ Vx) : C) \in Theory$ **using** *b* **by** (*simp add*: *Subst-def*)
}
{
fix *C f d* **assume** *ih*: $\bigwedge B'\ .\ [\![Sig\ aS \triangleright Vx : B \vdash d : B'; (Vx : A \vdash e1 \equiv e2 : B) \in Theory]\!]$
$\Longrightarrow (Vx : A \vdash (sub\ e1\ in\ d) \equiv (sub\ e2\ in\ d) : B') \in Theory$ **and** *wd*: $Sig\ aS \triangleright Vx : B \vdash f\ E@\ d : C$ **and**
eq: $(Vx : A \vdash e1 \equiv e2 : B) \in Theory$
from *wd* **obtain** B' **where** B': $f \in Sig\ aS : B' \to C$ **and** *d*: $Sig\ aS \triangleright Vx : B \vdash d : B'$ **by** *auto*
hence $(Vx : A \vdash (sub\ e1\ in\ d) \equiv (sub\ e2\ in\ d) : B') \in Theory$ **using** *ih eq* **by** *simp*
hence $(Vx : A \vdash f\ E@\ (sub\ e1\ in\ d) \equiv f\ E@\ (sub\ e2\ in\ d) : C) \in Theory$ **using** B' **by** (*simp add*: *Congr*)
thus $(Vx : A \vdash (sub\ e1\ in\ (f\ E@\ d)) \equiv (sub\ e2\ in\ (f\ E@\ d)) : C) \in Theory$ **by** (*simp add*: *Subst-def*)
}
qed

locale *Model* = *Interpretation I* + *Axioms Ax*
for *I* :: (*'t*,*'f*,*'o*,*'m*) *Interpretation* (**structure**)
and *Ax* :: (*'t*,*'f*) *Axioms* +
assumes *AxSound*: $\varphi \in (aAxioms\ Ax) \Longrightarrow L[\![\varphi]\!] \to (IBool\ True)$
and *Seq*[*simp*]: (*aSignature Ax*) = *iS*

lemma (**in** *Interpretation*) *Equiv*:
assumes $L[\![Vx : A \vdash e1 \equiv e2 : B]\!] \to (IBool\ True)$
shows $\exists\ g\ .\ (L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ g)) \wedge (L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g))$
proof−
from *assms Sig2Mor*[*of A e1 B*] **obtain** *g1* **where** *g1*: $L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ g1)$ **by** *auto*
from *assms Sig2Mor*[*of A e2 B*] **obtain** *g2* **where** *g2*: $L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g2)$ **by** *auto*
have $L[\![Vx : A \vdash e1 \equiv e2 : B]\!] \to (IBool\ (g1 = g2))$ **using** *g1 g2* **by** *auto*
hence *g1* = *g2* **using** *assms Functional*[*of Vx* : *A* ⊢ *e1*≡*e2* : *B* (*IBool True*) *IBool* (*g1*=*g2*)] **by** *simp*
thus *?thesis* **using** *g1 g2* **by** *auto*
qed

lemma (**in** *Interpretation*) *SubstComp*: $\bigwedge h\ C\ .\ [\![(L[\![Vx : A \vdash e : B]\!] \to (IMor\ g)) ; (L[\![Vx : B \vdash d : C]\!] \to (IMor\ h))]\!] \Longrightarrow$
$(L[\![Vx : A \vdash (sub\ e\ in\ d) : C]\!] \to (IMor\ (g\ ;;_{iC}\ h)))$
proof(*induct d*,*auto*)
{
fix *h C* **assume** *a*: $L[\![Vx : A \vdash e : B]\!] \to IMor\ g$ **and** *b*: $L[\![Vx : B \vdash Vx : C]\!] \to (IMor\ h)$
show $L[\![Vx : A \vdash e : C]\!] \to IMor\ (g\ ;;_{iC}\ h)$
proof−
have *beqc*: *B* = *C* **using** *b SigId*[*of iS B C*] *InterpExprWellDefined* **by** *simp*
have $L[\![Vx : B \vdash Vx : B]\!] \to (IMor\ (Id\ iC\ Ty[\![B]\!]))$ **using** *beqc b* **by** *auto*
hence *h*: *h* = *Id iC* $Ty[\![B]\!]$ **using** *b beqc* **by** (*auto simp add*: *Functional*)
have $g\ maps_{iC}\ Ty[\![A]\!]\ to\ Ty[\![B]\!]$ **using** *a MorphismsPreserved* **by** *auto*
hence *g* ∈ *Mor iC* **and** $cod_{iC}\ g = Ty[\![B]\!]$ **by** *auto*
hence (*g* ;;$_{iC}$ *h*) = *g* **using** *h ICat Category.Cidr*[*of iC g*] **by** *simp*
thus *?thesis* **using** *beqc a* **by** *simp*
qed
}
{
fix *f d C′ h C* **assume**
i: $\bigwedge h\ C'\ .\ L[\![Vx : B \vdash d : C']\!] \to IMor\ h \Longrightarrow L[\![Vx : A \vdash (sub\ e\ in\ d) : C']\!] \to (IMor\ (g\ ;;_{iC}\ h))$ **and**
a: $L[\![Vx : A \vdash e : B]\!] \to IMor\ g\ f \in Sig\ iS : C' \to C\ L[\![Vx : B \vdash d : C']\!] \to IMor\ h$
show $L[\![Vx : A \vdash f\ E@\ (sub\ e\ in\ d) : C]\!] \to (IMor\ (g\ ;;_{iC}\ (h\ ;;_{iC}\ Fn[\![f]\!])))$
proof−
have $L[\![Vx : A \vdash (sub\ e\ in\ d) : C']\!] \to (IMor\ (g\ ;;_{iC}\ h))$ **using** *i a* **by** *auto*
moreover hence *Sig iS* ▷ *Vx* : *A* ⊢ (*sub e in d*) : *C′* **using** *InterpExprWellDe-*

fined **by** *simp*
ultimately have *1*: $L[\![Vx : A \vdash f\ E@\ (sub\ e\ in\ d) : C]\!] \to (IMor\ ((g\ ;;_{iC}\ h)\ ;;_{iC}\ Fn[\![f]\!]))$ **using** *a(2)* **by** *auto*
have $g\ maps_{iC}\ Ty[\![A]\!]$ *to* $Ty[\![B]\!]$ **and** $h\ maps_{iC}\ Ty[\![B]\!]$ *to* $Ty[\![C']\!]$ **and** $Fn[\![f]\!]\ maps_{iC}\ Ty[\![C']\!]$ *to* $Ty[\![C]\!]$
using *a If Expr2Mor* **by** *simp+*
hence $g \approx>_{iC} h$ **and** $h \approx>_{iC} (Fn[\![f]\!])$ **using** *MapsToCompDef[of iC]* **by** *simp+*
hence $(g\ ;;_{iC}\ h)\ ;;_{iC}\ Fn[\![f]\!] = g\ ;;_{iC}\ (h\ ;;_{iC}\ Fn[\![f]\!])$ **using** *ICat Category.Cassoc[of iC]* **by** *simp*
thus *?thesis* **using** *1* **by** *simp*
qed
}
qed

lemma (**in** *Model*) *Sound*: $\varphi \in Theory \Longrightarrow L[\![\varphi]\!] \to (IBool\ True)$
proof(*rule Theory.induct*, (*simp-all add*: *AxSound*)+)
{
fix *A e B* **assume** *sig*: $Sig\ iS \triangleright Vx : A \vdash e : B$ **show** $L[\![Vx : A \vdash e{\equiv}e : B]\!] \to (IBool\ True)$
proof−
from *sig Sig2Mor*[*of A e B*] **obtain** *g* **where** *g*: $L[\![Vx : A \vdash e : B]\!] \to (IMor\ g)$ **by** *auto*
thus *?thesis* **using** *InterpEq*[*of I A e B g e g*] **by** *simp*
qed
}
{
fix *A e1 e2 B* **assume** *a*: $L[\![Vx : A \vdash e1{\equiv}e2 : B]\!] \to (IBool\ True)$ **show** $L[\![Vx : A \vdash e2{\equiv}e1 : B]\!] \to (IBool\ True)$
proof−
from *a* **obtain** *g* **where** *g1*: $L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ g)$ **and** *g2*: $L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g)$
using *Equiv* **by** *auto*
thus *?thesis* **by** *auto*
qed
}
{
fix *A e1 e2 B e3* **assume** *a*: $L[\![Vx : A \vdash e1{\equiv}e2 : B]\!] \to (IBool\ True)$ **and** *b*: $L[\![Vx : A \vdash e2{\equiv}e3 : B]\!] \to (IBool\ True)$
show $L[\![Vx : A \vdash e1{\equiv}e3 : B]\!] \to (IBool\ True)$
proof−
from *a* **obtain** *g* **where** *g1*: $L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ g)$ **and** *g2*: $L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g)$
using *Equiv* **by** *auto*
from *b* **obtain** g' **where** $g1'$: $L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g')$ **and** $g2'$: $L[\![Vx : A \vdash e3 : B]\!] \to (IMor\ g')$
using *Equiv* **by** *auto*
have *eq*: $g = g'$ **using** *g2* $g1'$ **using** *Functional* **by** *auto*
thus *?thesis* **using** *eq g1* $g2'$ **by** *auto*

qed
}
{
fix *A e1 e2 B f C* **assume** *a*: $L[\![Vx : A \vdash e1 \equiv e2 : B]\!] \to (IBool\ True)$ **and** *b*: $f \in Sig\ iS : B \to C$
show $L[\![Vx : A \vdash (f\ E@\ e1) \equiv (f\ E@\ e2) : C]\!] \to (IBool\ True)$
proof−
from *a* **obtain** *g* **where** *g1*: $L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ g)$ **and** *g2*: $L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g)$
using *Equiv* **by** *auto*
have *s1*: $Sig\ iS \triangleright Vx : A \vdash e1 : B$ **and** *s2*: $Sig\ iS \triangleright Vx : A \vdash e2 : B$ **using** *g1 g2 InterpExprWellDefined* **by** *simp*+
have $L[\![Vx : A \vdash (f\ E@\ e1) : C]\!] \to (IMor\ (g\ ;;_{iC}\ Fn[\![f]\!]))$
and $L[\![Vx : A \vdash (f\ E@\ e2) : C]\!] \to (IMor\ (g\ ;;_{iC}\ Fn[\![f]\!]))$ **using** *b g1 s1 g2 s2* **by** *auto*
thus *?thesis* **using** *InterpEqEq*[*of A* $(f\ E@\ e1)\ C\ (g\ ;;_{iC}\ Fn[\![f]\!])$] **by** *simp*
qed
}
{
fix *A e1 B e2 e3 C* **assume** *a*: $Sig\ iS \triangleright (Vx : A \vdash e1 : B)$ **and** *b*: $L[\![Vx : B \vdash e2 \equiv e3 : C]\!] \to (IBool\ True)$
show $L[\![Vx : A \vdash (sub\ e1\ in\ e2) \equiv (sub\ e1\ in\ e3) : C]\!] \to (IBool\ True)$
proof−
from *b* **obtain** *g* **where** *g1*: $L[\![Vx : B \vdash e2 : C]\!] \to (IMor\ g)$ **and** *g2*: $L[\![Vx : B \vdash e3 : C]\!] \to (IMor\ g)$
using *Equiv* **by** *auto*
from *a Sig2Mor*[*of A e1 B*] **obtain** *f* **where** *f*: $L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ f)$ **by** *auto*
have $L[\![Vx : A \vdash (sub\ e1\ in\ e2) : C]\!] \to (IMor\ (f\ ;;_{iC}\ g))$ **using** *SubstComp g1 f* **by** *simp*
moreover have $L[\![Vx : A \vdash (sub\ e1\ in\ e3) : C]\!] \to (IMor\ (f\ ;;_{iC}\ g))$ **using** *SubstComp g2 f* **by** *simp*
ultimately show *?thesis* **using** *InterpEqEq* **by** *simp*
qed
}
qed

record $('t,'f)\ TermEquivClT =$
$TDomain :: 't$
$TExprSet :: ('t,'f)\ Expression\ set$
$TCodomain :: 't$

locale $ZFAxioms = Ax : Axioms\ Ax$ **for** $Ax :: (ZF,ZF)\ Axioms$ (**structure**) +
assumes *fnzf*: $BaseFunctions\ (aSignature\ Ax) \in range\ explode$

lemma [*simp*]: $ZFAxioms\ T \Longrightarrow Axioms\ T$ **by** (*simp add*: *ZFAxioms-def*)

primrec $Expr2ZF :: (ZF,ZF)\ Expression \Rightarrow ZF$ **where**

Expr2ZFVx: *Expr2ZF Vx* = *ZFTriple* (*nat2Nat 0*) (*nat2Nat 0*) *Empty*
| *Expr2ZFfe*: *Expr2ZF* (*f E@ e*) = *ZFTriple* (*SucNat* (*ZFTFst* (*Expr2ZF e*)))
(*nat2Nat 1*)
(*Opair f* (*Expr2ZF e*))

definition *ZF2Expr* :: *ZF* ⇒ (*ZF*,*ZF*) *Expression* **where**
ZF2Expr = *inv Expr2ZF*

definition *ZFDepth* = *Nat2nat o ZFTFst*
definition *ZFType* = *Nat2nat o ZFTSnd*
definition *ZFData* = *ZFTThd*

lemma *Expr2ZFType0*: *ZFType* (*Expr2ZF e*) = *0* ⟹ *e* = *Vx*
by(*cases e*,*auto simp add*: *ZFType-def ZFTSnd Elem-Empty-Nat Elem-SucNat-Nat ZFSucNeq0*)

lemma *ZFDepthInNat*: *Elem* (*ZFTFst* (*Expr2ZF e*)) *Nat*
by(*induct e*, *auto simp add*: *ZFTFst HOLZF.Elem-Empty-Nat Elem-SucNat-Nat*)

lemma *Expr2ZFType1*: *ZFType* (*Expr2ZF e*) = *1* ⟹
∃ *f e′* . *e* = (*f E@ e′*) ∧ (*Suc* (*ZFDepth* (*Expr2ZF e′*))) = (*ZFDepth* (*Expr2ZF e*))
by(*cases e*,*auto simp add*: *ZFType-def ZFTSnd ZFDepth-def ZFTFst ZFZero ZFDepthInNat Nat2nat-SucNat*)

lemma *Expr2ZFDepth0*: *ZFDepth* (*Expr2ZF e*) = *0* ⟹ *ZFType* (*Expr2ZF e*) = *0*
by(*cases e*, *auto simp add*: *ZFDepth-def ZFTFst ZFType-def ZFTSnd ZFSucNeq0 ZFDepthInNat*)

lemma *Expr2ZFDepthSuc*: *ZFDepth* (*Expr2ZF e*) = *Suc n* ⟹ *ZFType* (*Expr2ZF e*) = *1*
by(*cases e*, *auto simp add*: *ZFDepth-def ZFTFst ZFType-def ZFTSnd ZFSucZero ZFSucNeq0 ZFZero*)

lemma *Expr2Data*: *ZFData* (*Expr2ZF* (*f E@ e*)) = *Opair f* (*Expr2ZF e*)
by(*auto simp add*: *ZFData-def ZFTThd*)

lemma *Expr2ZFinj*: *inj Expr2ZF*
proof(*auto simp add*: *inj-on-def*)
have *a*: ⋀ *n e d* . ⟦*n* = *ZFDepth* (*Expr2ZF e*) ; *Expr2ZF e* = *Expr2ZF d*⟧ ⟹ *e* = *d*
proof−
fix *n* **show** ⋀ *e d* . ⟦*n* = *ZFDepth* (*Expr2ZF e*) ; *Expr2ZF e* = *Expr2ZF d*⟧ ⟹ *e* = *d*
proof(*induct n*)
{
fix *e d* **assume** *a*: *0* = *ZFDepth* (*Expr2ZF e*) **and** *b*: *Expr2ZF e* = *Expr2ZF d*

have *0 = ZFDepth (Expr2ZF d)* **using** *a b* **by** *simp*
hence *e = Vx* **and** *d = Vx* **using** *a* **by** (*simp add*: *Expr2ZFDepth0 Expr2ZFType0*)+
thus *e = d* **by** *simp*
}
{
fix *n e d* **assume** *ih*: $\bigwedge e'\ d'$. $[\![n = ZFDepth\ (Expr2ZF\ e')\ ;\ Expr2ZF\ e' = Expr2ZF\ d']\!] \Longrightarrow e' = d'$
and *a*: *Suc n = ZFDepth (Expr2ZF e)* **and** *b*: *Expr2ZF e = Expr2ZF d*
have *ZFType (Expr2ZF e) = 1* **and** *ZFType (Expr2ZF d) = 1* **using** *a b Expr2ZFDepthSuc*[*of e n*] **by** *simp*+
from *this Expr2ZFType1*[*of e*] *Expr2ZFType1*[*of d*] **obtain** *fe fd e' d'*
where *e*: *e = (fe E@ e') ∧ (Suc (ZFDepth (Expr2ZF e'))) = (ZFDepth (Expr2ZF e))* **and**
d: *d = (fd E@ d') ∧ (Suc (ZFDepth (Expr2ZF d'))) = (ZFDepth (Expr2ZF d))* **by** *auto*
hence *Suc (ZFDepth (Expr2ZF e')) = Suc n* **using** *a* **by** *simp*
hence *ne*: *(ZFDepth (Expr2ZF e')) = n* **by** (*rule Suc-inject*)
have *edat*: *ZFData (Expr2ZF e) = Opair fe (Expr2ZF e')* **using** *e Expr2Data* **by** *simp*
have *ddat*: *ZFData (Expr2ZF d) = Opair fd (Expr2ZF d')* **using** *d Expr2Data* **by** *simp*
have *fd-eq-fe*: *fe = fd* **and** *(Expr2ZF e') = (Expr2ZF d')* **using** *edat ddat Opair b* **by** *auto*
hence *e' = d'* **using** *ne ih* **by** *auto*
thus *e = d* **using** *e d fd-eq-fe* **by** *auto*
}
qed
qed
fix *e d* **show** *Expr2ZF e = Expr2ZF d ⟹ e = d* **using** *a* **by** *auto*
qed

definition *TermEquivClGen T A e B* ≡ {*e'* . (*Vx* : *A* ⊢ *e'* ≡ *e* : *B*) ∈ *Axioms.Theory T*}
definition *TermEquivCl' T A e B* ≡ (| *TDomain = A* , *TExprSet = TermEquivClGen T A e B* , *TCodomain = B*|)

definition *m2ZF* :: (*ZF*,*ZF*) *TermEquivClT* ⇒ *ZF* **where**
m2ZF t ≡ *ZFTriple* (*TDomain t*) (*implode* (*Expr2ZF* ‘ (*TExprSet t*))) (*TCodomain t*)

definition *ZF2m* :: (*ZF*,*ZF*) *Axioms* ⇒ *ZF* ⇒ (*ZF*,*ZF*) *TermEquivClT* **where**
ZF2m T ≡ *inv-into* {*TermEquivCl' T A e B* | *A e B* . *True*} *m2ZF*

lemma *TDomain*: *TDomain* (*TermEquivCl' T A e B*) = *A* **by** (*simp add*: *TermEquivCl'-def*)
lemma *TCodomain*: *TCodomain* (*TermEquivCl' T A e B*) = *B* **by** (*simp add*: *TermEquivCl'-def*)

primrec *WellFormedToSet* :: (*ZF*,*ZF*) *Signature* ⇒ *nat* ⇒ (*ZF*,*ZF*) *Expression set* **where**
  WFS0: *WellFormedToSet S 0* = {*Vx*}
| *WFSS*: *WellFormedToSet S* (*Suc n*) = (*WellFormedToSet S n*) ∪
        { *f E@ e* | *f e* . *f* ∈ *BaseFunctions S* ∧ *e* ∈ (*WellFormedToSet S n*)}

lemma *WellFormedToSetInRangeExplode*: *ZFAxioms T* ⟹ (*Expr2ZF* ‘ (*WellFormedToSet* aS_T *n*)) ∈ *range explode*
proof((*induct n*),(*simp add*: *WellFormedToSet-def SingletonInRangeExplode*))
 fix *n* **assume** *zfx*: *ZFAxioms T* **and** *ih*: *ZFAxioms T* ⟹ *Expr2ZF* ‘ *WellFormedToSet* aS_T *n* ∈ *range explode*
 hence *a*: *Expr2ZF* ‘ *WellFormedToSet* aS_T *n* ∈ *range explode* **by** *simp*
 have *Expr2ZF* ‘ { *f E@ e* | *f e* . *f* ∈ *BaseFunctions* aS_T ∧ *e* ∈ (*WellFormedToSet* aS_T *n*)}
    = (λ (*f*, *ze*) . *Expr2ZF* (*f E@* (*ZF2Expr ze*))) ‘ ((*BaseFunctions* aS_T) × *Expr2ZF* ‘ (*WellFormedToSet* aS_T *n*))
  proof (*auto simp add*: *image-Collect image-def ZF2Expr-def Expr2ZFinj*)
   fix *f e*
   assume *f*: *f* ∈ *BaseFunctions* aS_T **and** *e*: *e* ∈ *WellFormedToSet* aS_T *n*
    show ∃ *x* ∈ *BaseFunctions* aS_T.∃ *y*. (∃ *x* ∈ *WellFormedToSet* aS_T *n*. *y* = *Expr2ZF x*) ∧
     ZFTriple (*SucNat* (*ZFTFst* (*Expr2ZF e*))) (*SucNat Empty*) (*Opair f* (*Expr2ZF e*)) =
      ZFTriple (*SucNat* (*ZFTFst* (*Expr2ZF* (*inv Expr2ZF y*)))) (*SucNat Empty*) (*Opair x* (*Expr2ZF* (*inv Expr2ZF y*)))
     apply(*rule bexI*[**where** *?x* = *f*], *rule exI*[**where** *?x* = *Expr2ZF e*])
     apply (*auto simp add*: *Expr2ZFinj f e*)
     done
  show ∃ *x*. (∃ *f e*. *x* = *f E@ e* ∧ *f* ∈ (*BaseFunctions* aS_T) ∧ *e* ∈ *WellFormedToSet* aS_T *n*) ∧
          ZFTriple (*SucNat* (*ZFTFst* (*Expr2ZF e*))) (*SucNat Empty*) (*Opair f* (*Expr2ZF e*)) = *Expr2ZF x*
     apply(*rule exI*[**where** *?x* = *f E@ e*])
     apply (*auto simp add*: *f e*)
     done
 qed
 moreover have (*BaseFunctions* aS_T) ∈ *range explode* **using** *zfx ZFAxioms.fnzf* **by** *simp*
  ultimately have *Expr2ZF* ‘ { *f E@ e* | *f e* . *f* ∈ *BaseFunctions* aS_T ∧ *e* ∈ (*WellFormedToSet* aS_T *n*)} ∈ *range explode*
   using *a ZFProdFnInRangeExplode* **by** *simp*
 moreover have *Expr2ZF* ‘ *WellFormedToSet* aS_T (*Suc n*) = (*Expr2ZF* ‘ (*WellFormedToSet* aS_T *n*)) ∪
                (*Expr2ZF* ‘ { *f E@ e* | *f e* . *f* ∈ *BaseFunctions* aS_T ∧ *e* ∈ (*WellFormedToSet* aS_T *n*)}) **by** *auto*
  ultimately show *Expr2ZF* ‘ *WellFormedToSet* aS_T (*Suc n*) ∈ *range explode* **using** *ZFUnionInRangeExplode a* **by** *simp*
qed

lemma *WellDefinedToWellFormedSet*: $\bigwedge B\ .\ (Sig\ S \triangleright (Vx : A \vdash e : B)) \Longrightarrow \exists n.\ e \in WellFormedToSet\ S\ n$
proof(*induct e*)
{
fix *B* **assume** $Sig\ S \triangleright Vx : A \vdash Vx : B$
show $\exists n.\ Vx \in WellFormedToSet\ S\ n$ **by** (*rule exI*[*of - 0*], *auto*)
}
{
fix *f e B* **assume** *ih*: $\bigwedge B'.\ Sig\ S \triangleright Vx : A \vdash e : B' \Longrightarrow \exists n.\ e \in WellFormedToSet\ S\ n$
and $Sig\ S \triangleright Vx : A \vdash (f\ E@\ e) : B$
from *this* **obtain** B' **where** $Sig\ S \triangleright (Vx : A \vdash e : B')$ **and** *f*: $f \in Sig\ S : B' \to B$ **by** *auto*
from *this* **obtain** *n* **where** *a*: $e \in WellFormedToSet\ S\ n$ **using** *ih* **by** *auto*
have *ff*: $f \in BaseFunctions\ S$ **using** *f* **by** *auto*
show $\exists n.\ (f\ E@\ e) \in WellFormedToSet\ S\ n$ **by**(*rule exI*[*of - Suc n*], *auto simp add*: *a ff*)
}
qed

lemma *TermSetInSet*: $ZFAxioms\ T \Longrightarrow Expr2ZF\ `\ (TermEquivClGen\ T\ A\ e\ B) \in range\ explode$
proof−
assume *zfax*: *ZFAxioms T*
have $(TermEquivClGen\ T\ A\ e\ B) \subseteq \{e'\ .\ (Sig\ aS_T \triangleright (Vx : A \vdash e' : B))\}$
proof(*auto simp add*: *TermEquivClGen-def*)
fix *x* **assume** $Vx : A \vdash x \equiv e : B \in Axioms.Theory\ T$
hence $Sig\ aS_T \triangleright Vx : A \vdash x \equiv e : B$ **by** (*auto simp add*: *Axioms.Equiv2WellDefined zfax*)
thus $Sig\ aS_T \triangleright Vx : A \vdash x : B$ **by** *auto*
qed
also have ... $\subseteq (\bigcup\ n\ .\ (WellFormedToSet\ aS_T\ n))$ **by** (*auto simp add*: *WellDefinedToWellFormedSet*)
finally have $Expr2ZF\ `\ (TermEquivClGen\ T\ A\ e\ B) \subseteq Expr2ZF\ `\ (\bigcup\ n\ .\ (WellFormedToSet\ aS_T\ n))$ **by** *auto*
also have ... $= (\bigcup\ n\ .\ (Expr2ZF\ `\ (WellFormedToSet\ aS_T\ n)))$ **by** *auto*
finally have $Expr2ZF\ `\ (TermEquivClGen\ T\ A\ e\ B) \subseteq (\bigcup\ n\ .\ (Expr2ZF\ `\ (WellFormedToSet\ aS_T\ n)))$ **by** *auto*
moreover have $(\bigcup\ n\ .\ (Expr2ZF\ `\ (WellFormedToSet\ aS_T\ n))) \in range\ explode$
using *zfax ZFUnionNatInRangeExplode WellFormedToSetInRangeExplode* **by** *auto*
ultimately show *?thesis* **using** *ZFSubsetRangeExplode*[*of* $(\bigcup\ n\ .\ (Expr2ZF\ `\ (WellFormedToSet\ aS_T\ n)))\ Expr2ZF\ `\ (TermEquivClGen\ T\ A\ e\ B)$] **by** *simp*
qed

lemma *m2ZFinj-on*: $ZFAxioms\ T \Longrightarrow inj\text{-}on\ m2ZF\ \{TermEquivCl'\ T\ A\ e\ B \mid A\ e\ B\ .\ True\}$

apply(*auto simp only*: *inj-on-def m2ZF-def*)
apply(*drule ZFTriple*)
apply(*auto simp add*: *TDomain TCodomain implode-def*)
apply(*simp add*: *TermEquivCl'-def*)
apply(*drule inv-into-injective*)
apply(*auto simp add*: *TermSetInSet inj-image-eq-iff Expr2ZFinj*)
done

lemma *ZF2m*: *ZFAxioms T* $\Longrightarrow$ *ZF2m T* (*m2ZF* (*TermEquivCl' T A e B*)) = (*TermEquivCl' T A e B*)
proof−
assume *zfax*: *ZFAxioms T*
let *?A* = {*TermEquivCl' T A e B* | *A e B* . *True*} **and** *?x* = *TermEquivCl' T A e B*
have *?x* ∈ *?A* **by** *auto*
thus *?thesis* **using** *m2ZFinj-on*[*of T*] *inv-into-f-f zfax* **by** (*simp add*: *ZF2m-def*)
qed

definition *TermEquivCl* (‹[-,-,-]₁›) **where** [*A*,*e*,*B*]$_T$ ≡ *m2ZF* (*TermEquivCl' T A e B*)

definition *CLDomain T* ≡ *TDomain o ZF2m T*
definition *CLCodomain T* ≡ *TCodomain o ZF2m T*

definition *CanonicalComp T f g* ≡
THE h . ∃ *e e'* . *h* = [*CLDomain T f*,*sub e in e'*,*CLCodomain T g*]$_T$ ∧
f = [*CLDomain T f*,*e*,*CLCodomain T f*]$_T$ ∧ *g* = [*CLDomain T g*,*e'*,*CLCodomain T g*]$_T$

lemma *CLDomain*: *ZFAxioms T* $\Longrightarrow$ *CLDomain T* [*A*,*e*,*B*]$_T$ = *A* **by** (*simp add*: *TermEquivCl-def CLDomain-def ZF2m TDomain*)
lemma *CLCodomain*: *ZFAxioms T* $\Longrightarrow$ *CLCodomain T* [*A*,*e*,*B*]$_T$ = *B* **by** (*simp add*: *TermEquivCl-def CLCodomain-def ZF2m TCodomain*)

lemma *Equiv2Cl*: **assumes** *Axioms T* **and** (*Vx* : *A* ⊢ *e* ≡ *d* : *B*) ∈ *Axioms.Theory T* **shows** [*A*,*e*,*B*]$_T$ = [*A*,*d*,*B*]$_T$
proof−
have {*e'*. (*Vx* : *A* ⊢ *e'*≡*e* : *B*) ∈ *Axioms.Theory T*} = {*e'*. (*Vx* : *A* ⊢ *e'*≡*d* : *B*) ∈ *Axioms.Theory T*}
using *assms* **by**(*blast intro*: *Axioms.Trans Axioms.Symm*)
thus *?thesis* **by**(*auto simp add*: *TermEquivCl-def TermEquivCl'-def TermEquivClGen-def*)
qed

lemma *Cl2Equiv*:
assumes *axt*: *ZFAxioms T* **and** *sa*: *Sig aS*$_T$ ▷ (*Vx* : *A* ⊢ *e* : *B*) **and** *cl*: [*A*,*e*,*B*]$_T$ = [*A*,*d*,*B*]$_T$
shows (*Vx* : *A* ⊢ *e* ≡ *d* : *B*) ∈ *Axioms.Theory T*

proof−
have *ZF2m T* (*m2ZF* (*TermEquivCl′ T A e B*)) = *ZF2m T* (*m2ZF* (*TermEquivCl′ T A d B*)) **using** *cl* **by** (*simp add*: *TermEquivCl-def*)
hence *a*:*TermEquivCl′ T A e B* = *TermEquivCl′ T A d B* **using** *assms* **by** (*simp add*: *ZF2m*)
have (*Vx* : *A* ⊢ *e* ≡ *e* : *B*) ∈ *Axioms.Theory T* **using** *axt sa Axioms.Refl*[*of T*] **by** *simp*
hence *e* ∈ *TermEquivClGen T A e B* **by** (*simp add*: *TermEquivClGen-def*)
hence *e* ∈ *TermEquivClGen T A d B* **using** *a* **by** (*simp add*: *TermEquivCl′-def*)
thus *?thesis* **by** (*simp add*: *TermEquivClGen-def*)
qed

lemma *CanonicalCompWellDefined*:
assumes *zaxt*: *ZFAxioms T* **and** *Sig aS* $_T$ ▷ (*Vx* : *A* ⊢ *d* : *B*) **and** *Sig aS* $_T$ ▷ (*Vx* : *B* ⊢ *d′* : *C*)
shows *CanonicalComp T* [*A*,*d*,*B*] $_T$ [*B*,*d′*,*C*] $_T$ = [*A*,*sub d in d′*,*C*] $_T$
proof((*simp only*: *zaxt CanonicalComp-def CLDomain CLCodomain*),(*rule the-equality*),
(*rule exI*[*of* - *d*], *rule exI*[*of* - *d′*], *auto*))
fix *e e′* **assume** *de*: [*A*,*d*,*B*] $_T$ = [*A*,*e*,*B*] $_T$ **and** *de′*: [*B*,*d′*,*C*] $_T$ = [*B*,*e′*,*C*] $_T$
have *axt*: *Axioms T* **using** *assms* **by** *simp*
have *a*: (*Vx* : *A* ⊢ *d* ≡ *e* : *B*) ∈ *Axioms.Theory T* **using** *assms de Cl2Equiv* **by** *simp*
hence *sa*: (*Vx* : *A* ⊢ (*sub d in d′*) ≡ (*sub e in d′*) : *C*) ∈ *Axioms.Theory T*
using *assms Axioms.Subst′*[*of T*] **by** *simp*
have *b*: (*Vx* : *B* ⊢ *d′*≡ *e′* : *C*) ∈ *Axioms.Theory T* **using** *assms de′ Cl2Equiv* **by** *simp*
have *Sig aS* $_T$ ▷ (*Vx* : *A* ⊢ *d* ≡ *e* : *B*) **using** *a axt Axioms.Equiv2WellDefined*[*of T*] **by** *simp*
hence *Sig aS* $_T$ ▷ (*Vx* : *A* ⊢ *e* : *B*) **by** *auto*
hence (*Vx* : *A* ⊢ (*sub e in d′*) ≡ (*sub e in e′*) : *C*) ∈ *Axioms.Theory T* **using** *b Axioms.Subst*[*of T*] *axt* **by** *simp*
hence (*Vx* : *A* ⊢ (*sub e in e′*) ≡ (*sub d in d′*) : *C*) ∈ *Axioms.Theory T* **using** *sa axt*
by (*blast intro*: *Axioms.Symm*[*of T*] *Axioms.Trans*[*of T*])
thus [*A*,*sub e in e′*,*C*] $_T$ = [*A*,*sub d in d′*,*C*] $_T$ **using** *zaxt Equiv2Cl* **by** *simp*
qed

definition *CanonicalCat′ T* ≡ ⦇
Obj = *BaseTypes* (*aS* $_T$),
Mor = {[*A*,*e*,*B*] $_T$ | *A e B* . *Sig aS* $_T$ ▷ (*Vx* : *A* ⊢ *e* : *B*)},
Dom = *CLDomain T*,
Cod = *CLCodomain T*,
Id = (λ *A* . [*A*,*Vx*,*A*] $_T$),
Comp = *CanonicalComp T*
⦈

definition *CanonicalCat T* ≡ *MakeCat* (*CanonicalCat′ T*)

lemma *CanonicalCat′MapsTo*:

assumes f *maps*$_{CanonicalCat'\ T}$ X *to* Y **and** *zx*: *ZFAxioms* T
shows $\exists\ ef\ .\ f = [X,ef,Y]_T \wedge Sig\ (aSignature\ T) \triangleright (Vx : X \vdash ef : Y)$
proof−
have *fm*: $f \in Mor\ (CanonicalCat'\ T)$ **and** *fd*: $Dom\ (CanonicalCat'\ T)\ f = X$ **and** *fc*: $Cod\ (CanonicalCat'\ T)\ f = Y$
using *assms* **by** *auto*
from *fm* **obtain** *ef A B* **where** *ef*: $f = [A,ef,B]_T$ **and** *efsig*: $Sig\ (aSignature\ T) \triangleright (Vx : A \vdash ef : B)$
by (*auto simp add*: *CanonicalCat'-def*)
have $A = X$ **and** $B = Y$ **using** *fd fc ef zx CLDomain*[*of T*] *CLCodomain*[*of T*] **by** (*simp add*: *CanonicalCat'-def*)+
thus *?thesis* **using** *ef efsig* **by** *auto*
qed

lemma *CanonicalCatCat'*: *ZFAxioms* $T \Longrightarrow$ *Category-axioms* (*CanonicalCat'* T)
proof(*auto simp only*: *Category-axioms-def*)
have *obj*: $obj_{CanonicalCat'\ T} = BaseTypes\ (aSignature\ T)$ **by** (*simp add*: *CanonicalCat'-def*)
assume *zx*: *ZFAxioms* T
hence *axt*: *Axioms* T **by** *simp*
{
fix f **assume** *a*: $f \in mor_{CanonicalCat'\ T}$
from *a* **obtain** $A\ e\ B$ **where** *f*: $f = [A,e,B]_T$ **and** *fwd*: $Sig\ (aSignature\ T) \triangleright (Vx : A \vdash e : B)$
by (*auto simp add*: *CanonicalCat'-def*)
have *domf*: $dom_{CanonicalCat'\ T}\ f = CLDomain\ T\ f$ **and** *codf*: $cod_{CanonicalCat'\ T}\ f = CLCodomain\ T\ f$
by (*simp add*: *CanonicalCat'-def*)+
have *absig*: $A \in Ty_{aSignature\ T} \wedge B \in Ty_{aSignature\ T}$ **using** *fwd Signature.SigTy*[*of* (*aSignature T*)] *Axioms.AxSig axt*
by *auto*
have *Awd*: $Sig\ (aSignature\ T) \triangleright (Vx : A \vdash Vx : A)$ **and** *Bwd*: $Sig\ (aSignature\ T) \triangleright (Vx : B \vdash Vx : B)$
using *absig* **by** *auto*
show $dom_{CanonicalCat'\ T}\ f \in obj_{CanonicalCat'\ T}$ **using** *domf f obj CLDomain*[*of T A e B*] *absig zx*
by *simp*
show $cod_{CanonicalCat'\ T}\ f \in obj_{CanonicalCat'\ T}$ **using** *codf f obj CLCodomain*[*of T A e B*] *absig zx*
by *simp*
have *idA*: $Id\ (CanonicalCat'\ T)\ A = [A,Vx,A]_T$ **and** *idB*: $Id\ (CanonicalCat'\ T)\ B = [B,Vx,B]_T$
by (*simp add*: *CanonicalCat'-def*)+
have $(Id\ (CanonicalCat'\ T)\ (dom_{CanonicalCat'\ T}\ f)) ;;_{CanonicalCat'\ T} f = [A, sub\ Vx\ in\ e,\ B]_T$
using *f domf CLDomain*[*of T A e B*] *idA axt CanonicalCompWellDefined*[*of T*] *Awd fwd zx*
by (*simp add*: *CanonicalCat'-def*)
thus $(Id\ (CanonicalCat'\ T)\ (dom_{CanonicalCat'\ T}\ f)) ;;_{CanonicalCat'\ T} f = f$

using *f SubstXinE*[*of e*] **by** *simp*
have $f\ ;;_{CanonicalCat'\ T}\ (Id\ (CanonicalCat'\ T)\ (cod_{CanonicalCat'\ T}\ f)) = [A, sub\ e\ in\ Vx,\ B]_T$
using *f codf CLCodomain*[*of T A e B*] *idB axt CanonicalCompWellDefined*[*of T*] *Bwd fwd zx*
by (*simp add*: *CanonicalCat'-def*)
thus $f\ ;;_{CanonicalCat'\ T}\ (Id\ (CanonicalCat'\ T)\ (cod_{CanonicalCat'\ T}\ f)) = f$ **using** *f Subst-def* **by** *simp*
}
{
fix *X* **assume** *a*: $X \in obj_{CanonicalCat'\ T}$
have $X \in BaseTypes\ (aSignature\ T)$ **using** *a* **by** (*simp add*: *CanonicalCat'-def*)
hence *b*: $Sig\ (aSignature\ T) \triangleright (Vx : X \vdash Vx : X)$ **by** *auto*
show $Id\ (CanonicalCat'\ T)\ X\ maps_{CanonicalCat'\ T}\ X\ to\ X$
apply(*rule MapsToI*)
apply(*auto simp add*: *CanonicalCat'-def CLDomain CLCodomain zx*)
apply(*rule exI*)+
apply(*auto simp add*: *b*)
done
}
{
fix *f g h* **assume** *a*: $f \approx>_{CanonicalCat'\ T} g$ **and** *b*: $g \approx>_{CanonicalCat'\ T} h$
from *a b* **have** *fm*: $f \in Mor\ (CanonicalCat'\ T)$ **and** *gm*: $g \in Mor\ (CanonicalCat'\ T)$
and *hm*: $h \in Mor\ (CanonicalCat'\ T)$ **and** *fg*: $Cod\ (CanonicalCat'\ T)\ f = Dom\ (CanonicalCat'\ T)\ g$
and *gh*: $Cod\ (CanonicalCat'\ T)\ g = Dom\ (CanonicalCat'\ T)\ h$ **by** *auto*
from *fm* **obtain** *A ef B* **where** *f*: $f = [A,ef,B]_T$ **and** *fwd*: $Sig\ (aSignature\ T) \triangleright (Vx : A \vdash ef : B)$
by (*auto simp add*: *CanonicalCat'-def*)
from *gm* **obtain** *B' eg C* **where** *g*: $g = [B',eg,C]_T$ **and** *gwd*: $Sig\ (aSignature\ T) \triangleright (Vx : B' \vdash eg : C)$
by (*auto simp add*: *CanonicalCat'-def*)
from *hm* **obtain** *C' eh D* **where** *h*: $h = [C',eh,D]_T$ **and** *hwd*: $Sig\ (aSignature\ T) \triangleright (Vx : C' \vdash eh : D)$
by (*auto simp add*: *CanonicalCat'-def*)
from *fg* **have** *Beq*: $B = B'$ **using** *f g zx CLCodomain*[*of T A ef B*] *CLDomain*[*of T B' eg C*] **by** (*simp add*: *CanonicalCat'-def*)
from *gh* **have** *Ceq*: $C = C'$ **using** *g h zx CLCodomain*[*of T B' eg C*] *CLDomain*[*of T C' eh D*] **by** (*simp add*: *CanonicalCat'-def*)
have $CanonicalComp\ T\ (CanonicalComp\ T\ f\ g)\ h = CanonicalComp\ T\ f\ (CanonicalComp\ T\ g\ h)$
proof−
have $(CanonicalComp\ T\ f\ g) = [A,sub\ ef\ in\ eg,C]_T$
using *axt fwd gwd Beq zx CanonicalCompWellDefined*[*of T A ef B eg C*] *f g* **by** *simp*
moreover have $Sig\ aS_T \triangleright Vx : A \vdash sub\ ef\ in\ eg : C$ **using** *fwd gwd SubstWellDefined Beq* **by** *simp*
ultimately have *a*: $CanonicalComp\ T\ (CanonicalComp\ T\ f\ g)\ h = [A,(sub$

$(sub\ ef\ in\ eg)\ in\ eh),D]_T$
using *CanonicalCompWellDefined*[*of T A sub ef in eg C eh D*] *axt hwd h Beq Ceq zx* **by** *simp*
have $(CanonicalComp\ T\ g\ h) = [B,sub\ eg\ in\ eh,D]_T$
using *axt gwd hwd Ceq Beq CanonicalCompWellDefined*[*of T B eg C eh D*] *g h zx* **by** *simp*
moreover have $Sig\ aS_T \triangleright Vx : B \vdash sub\ eg\ in\ eh : D$ **using** *gwd hwd SubstWellDefined Beq Ceq* **by** *simp*
ultimately have *b*: $CanonicalComp\ T\ f\ (CanonicalComp\ T\ g\ h) = [A,(sub\ ef\ in\ (sub\ eg\ in\ eh)),D]_T$
using *CanonicalCompWellDefined*[*of T A ef B sub eg in eh D*] *axt fwd f zx* **by** *simp*
show *?thesis* **using** *a b* **by** (*simp add*: *SubstAssoc*)
qed
thus $(f\ ;;_{CanonicalCat'\ T}\ g)\ ;;_{CanonicalCat'\ T}\ h = f\ ;;_{CanonicalCat'\ T}\ (g\ ;;_{CanonicalCat'\ T}\ h)$
by(*simp add*: *CanonicalCat'-def*)
}
{
fix *f X Y g Z* **assume** *a*: $f\ maps_{CanonicalCat'\ T}\ X\ to\ Y$ **and** *b*: $g\ maps_{CanonicalCat'\ T}\ Y\ to\ Z$
from *a CanonicalCat'MapsTo*[*of T f X Y*] **obtain** *ef*
where *f*: $f = [X,ef,Y]_T$ **and** *fwd*: $Sig\ (aSignature\ T) \triangleright (Vx : X \vdash ef : Y)$ **using** *zx* **by** *auto*
from *b CanonicalCat'MapsTo*[*of T g Y Z*] **obtain** *eg*
where *g*: $g = [Y,eg,Z]_T$ **and** *gwd*: $Sig\ (aSignature\ T) \triangleright (Vx : Y \vdash eg : Z)$ **using** *zx* **by** *auto*
have *fg*: $CanonicalComp\ T\ f\ g = [X,sub\ ef\ in\ eg,Z]_T$
using *CanonicalCompWellDefined*[*of T X ef Y eg Z*] *f g fwd gwd zx* **by** *simp*
have *fgwd*: $Sig\ (aSignature\ T) \triangleright (Vx : X \vdash sub\ ef\ in\ eg : Z)$
using *fwd gwd SubstWellDefined*[*of* aS_T] **by** *simp*
have $(CanonicalComp\ T\ f\ g)\ maps_{CanonicalCat'\ T}\ X\ to\ Z$
proof(*rule MapsToI*)
show $Dom\ (CanonicalCat'\ T)\ (CanonicalComp\ T\ f\ g) = X$
using *fg CLDomain*[*of T*] *zx* **by** (*simp add*: *CanonicalCat'-def*)
show $CanonicalComp\ T\ f\ g \in Mor\ (CanonicalCat'\ T)$ **using** *fg fgwd* **by** (*auto simp add*: *CanonicalCat'-def*)
show $Cod\ (CanonicalCat'\ T)\ (CanonicalComp\ T\ f\ g) = Z$
using *fg CLCodomain*[*of T*] *zx* **by** (*simp add*: *CanonicalCat'-def*)
qed
thus $f\ ;;_{CanonicalCat'\ T}\ g\ maps_{CanonicalCat'\ T}\ X\ to\ Z$ **by** (*simp add*: *CanonicalCat'-def*)
}
qed

lemma *CanonicalCatCat*: $ZFAxioms\ T \Longrightarrow Category\ (CanonicalCat\ T)$
by (*simp add*: *CanonicalCat-def CanonicalCatCat' MakeCat*)

definition *CanonicalInterpretation* **where**

CanonicalInterpretation $T \equiv (\!|$
  ISignature $=$ *aSignature* T,
  ICategory $=$ *CanonicalCat* T,
  ITypes $= \lambda\ A\ .\ A$,
  IFunctions $= \lambda\ f\ .\ [SigDom\ (aSignature\ T)\ f, f\ E@\ Vx, SigCod\ (aSignature\ T)\ f]_T$
$|\!)$

abbreviation *CI* $T \equiv$ *CanonicalInterpretation* T

lemma *CIObj*: *Obj* (*CanonicalCat* T) $=$ *BaseTypes* (*aSignature* T)
by (*simp add*: *MakeCat-def CanonicalCat-def CanonicalCat′-def*)

lemma *CIMor*: *ZFAxioms* $T \Longrightarrow [A,e,B]_T \in Mor\ (CanonicalCat\ T) = Sig\ (aSignature\ T) \triangleright (Vx : A \vdash e : B)$
proof(*auto simp add*: *MakeCat-def CanonicalCat-def CanonicalCat′-def*)
  fix $A'\ e'\ B'$ **assume** *zx*: *ZFAxioms* T **and** *b*: $[A,e,B]_T = [A',e',B']_T$ **and** *c*: $Sig\ aS_T \triangleright Vx : A' \vdash e' : B'$
  have $CLDomain\ T\ [A,e,B]_T = CLDomain\ T\ [A',e',B']_T$
   and $CLCodomain\ T\ [A,e,B]_T = CLCodomain\ T\ [A',e',B']_T$ **using** *b* **by** *simp+*
  hence $A = A'$ **and** $B = B'$ **by** (*simp add*: *CLDomain CLCodomain zx*)+
  hence $(Vx : A \vdash e' \equiv e : B) \in Axioms.Theory\ T$ **using** *zx b c Cl2Equiv*[*of T A e′ B e*] **by** *simp*
  hence $Sig\ aS_T \triangleright (Vx : A \vdash e' \equiv e : B)$ **using** *Axioms.Equiv2WellDefined*[*of T*] *zx* **by** *simp*
  thus $Sig\ aS_T \triangleright Vx : A \vdash e : B$ **by** *auto*
qed

lemma *CIDom*: $[\![ZFAxioms\ T\ ;\ [A,e,B]_T \in Mor(CanonicalCat\ T)]\!] \Longrightarrow Dom\ (CanonicalCat\ T)\ [A,e,B]_T = A$
proof−
  assume $[A,e,B]_T \in Mor(CanonicalCat\ T)$ **and** *ZFAxioms T*
  thus $Dom\ (CanonicalCat\ T)\ [A,e,B]_T = A$
   by (*simp add*: *CLDomain MakeCatMor CanonicalCat-def MakeCat-def CanonicalCat′-def*)
qed

lemma *CICod*: $[\![ZFAxioms\ T\ ;\ [A,e,B]_T \in Mor(CanonicalCat\ T)]\!] \Longrightarrow Cod\ (CanonicalCat\ T)\ [A,e,B]_T = B$
proof−
  assume $[A,e,B]_T \in Mor(CanonicalCat\ T)$ **and** *ZFAxioms T*
  thus $Cod\ (CanonicalCat\ T)\ [A,e,B]_T = B$
   by (*simp add*: *CLCodomain MakeCatMor CanonicalCat-def MakeCat-def CanonicalCat′-def*)
qed

lemma *CIId*: $[\![A \in BaseTypes\ (aSignature\ T)]\!] \Longrightarrow Id\ (CanonicalCat\ T)\ A = [A, Vx, A]_T$
by(*simp add*: *MakeCat-def CanonicalCat-def CanonicalCat′-def*)

lemma *CIComp*:
assumes *ZFAxioms T* **and** *Sig* (*aSignature T*) ▷ (*Vx* : *A* ⊢ *e* : *B*) **and** *Sig* (*aSignature T*) ▷ (*Vx* : *B* ⊢ *d* : *C*)
shows $[A,e,B]_T$;;$_{CanonicalCat\ T}$ $[B,d,C]_T$ = $[A,sub\ e\ in\ d,C]_T$
proof−
have $[A,e,B]_T$ ≈>$_{CanonicalCat'\ T}$ $[B,d,C]_T$
proof(*rule CompDefinedI*)
show $[A,e,B]_T$ ∈ *Mor* (*CanonicalCat' T*) **and** $[B,d,C]_T$ ∈ *Mor* (*CanonicalCat' T*)
using *assms* **by** (*auto simp add*: *CanonicalCat'-def*)
have $cod_{CanonicalCat'\ T}$ $[A,e,B]_T$ = *B* **using** *assms* **by** (*simp add*: *CanonicalCat'-def CLCodomain*)
moreover have $dom_{CanonicalCat'\ T}$ $[B,d,C]_T$ = *B* **using** *assms* **by** (*simp add*: *CanonicalCat'-def CLDomain*)
ultimately show $cod_{CanonicalCat'\ T}$ $[A,e,B]_T$ = $dom_{CanonicalCat'\ T}$ $[B,d,C]_T$ **by** *simp*
qed
hence $[A,e,B]_T$;;$_{CanonicalCat\ T}$ $[B,d,C]_T$ = $[A,e,B]_T$;;$_{CanonicalCat'\ T}$ $[B,d,C]_T$
by (*auto simp add*: *MakeCatComp CanonicalCat-def*)
thus $[A,e,B]_T$;;$_{CanonicalCat\ T}$ $[B,d,C]_T$ = $[A,\ sub\ e\ in\ d,\ C]_T$
using *CanonicalCompWellDefined*[*of T*] *assms* **by** (*simp add*: *CanonicalCat'-def*)
qed

lemma [*simp*]: *ZFAxioms T* ⟹ *Category* $iC_{CI\ T}$ **by**(*simp add*: *CanonicalInterpretation-def CanonicalCatCat*[*of T*])
lemma [*simp*]: *ZFAxioms T* ⟹ *Signature* $iS_{CI\ T}$ **by**(*simp add*: *CanonicalInterpretation-def Axioms.AxSig*)

lemma *CIInterpretation*: *ZFAxioms T* ⟹ *Interpretation* (*CI T*)
proof(*auto simp only*: *Interpretation-def*)
assume *xt*: *ZFAxioms T*
show *Category* $iC_{CI\ T}$ **and** *Signature* $iS_{CI\ T}$ **using** *xt* **by** *simp*+
{
fix *A* **assume** *A* ∈ *BaseTypes* ($iS_{CI\ T}$) **thus** $Ty⟦A⟧_{CI\ T}$ ∈ *Obj* ($iC_{CI\ T}$)
by(*simp add*: *CanonicalInterpretation-def CIObj*[*of T*])
}
{
fix *f A B* **assume** *a*: *f* ∈ *Sig* $iS_{CI\ T}$: *A* → *B*
hence [$sDom_{ISignature\ (CI\ T)}$ *f*,*f E@ Vx*,$sCod_{ISignature\ (CI\ T)}$ *f*]$_T$ ∈ $mor_{CanonicalCat\ T}$
using *xt* **by**(*auto simp add*: *CanonicalInterpretation-def CIMor*)
thus $Fn⟦f⟧_{CI\ T}$ $maps_{ICategory\ (CI\ T)}$ $Ty⟦A⟧_{CI\ T}$ *to* $Ty⟦B⟧_{CI\ T}$ **using** *a xt*
by(*auto simp add*: *CanonicalInterpretation-def MapsTo-def funsignature-abbrev-def CIDom CICod*)
}
qed

lemma *CIInterp2Mor*: *ZFAxioms T* ⟹ (⋀ *B* . *Sig* $iS_{CI\ T}$ ▷ (*Vx* : *A* ⊢ *e* : *B*) ⟹ $L⟦Vx : A ⊢ e : B⟧_{CI\ T}$ → (*IMor* [*A*, *e*, *B*]$_T$))

proof(*induct e*)
assume *xt*: *ZFAxioms T*
{
fix *B* **assume** *a*: *Sig* $iS_{CI\ T} \triangleright Vx : A \vdash Vx : B$
have *aeqb*: $A = B$ **using** *a SigId*[*of* $iS_{CI\ T}$ *A B*] *Interpretation.InterpExprWellDefined*[*of CI T*] **by** *simp*
moreover have *bt*: $A \in BaseTypes\ iS_{CI\ T}$ **using** *a SigTyId aeqb* **by** *simp*
ultimately have *b*: $L[\![Vx : A \vdash Vx : B]\!]_{CI\ T} \rightarrow (IMor\ (Id\ iC_{CI\ T}\ Ty[\![A]\!]_{CI\ T}))$ **by** *auto*
have $Ty[\![A]\!]_{CI\ T} = A$ **by** (*simp add*: *CanonicalInterpretation-def*)
hence $(Id\ iC_{CI\ T}\ Ty[\![A]\!]_{CI\ T}) = [A,Vx,A]_T$ **using** *bt CIId* **by**(*simp add*: *CanonicalInterpretation-def*)
thus $L[\![Vx : A \vdash Vx : B]\!]_{CI\ T} \rightarrow (IMor\ [A,Vx,B]_T)$ **using** *aeqb b* **by** *simp*
}
{
fix *f e B* **assume** *ih*: $\bigwedge B'.\ [\![ZFAxioms\ T\ ;\ Sig\ iS_{CI\ T} \triangleright (Vx : A \vdash e : B')]\!] \Longrightarrow (L[\![Vx : A \vdash e : B']\!]_{CI\ T} \rightarrow (IMor\ [A,e,B']_T))$
and *a*: $Sig\ iS_{CI\ T} \triangleright Vx : A \vdash f\ E@\ e : B$
from *a* **obtain** B' **where** *sig*: $Sig\ iS_{CI\ T} \triangleright (Vx : A \vdash e : B')$ **and** *f*: $f \in Sig\ iS_{CI\ T} : B' \rightarrow B$ **by** *auto*
hence $L[\![Vx : A \vdash e : B']\!]_{CI\ T} \rightarrow (IMor\ [A,e,B']_T)$ **using** *ih xt* **by** *auto*
hence *b*: $L[\![Vx : A \vdash f\ E@\ e : B]\!]_{CI\ T} \rightarrow (IMor\ ([A,e,B']_T\ ;;_{ICategory\ (CI\ T)}\ Fn[\![f]\!]_{CI\ T}))$ **using** *sig f* **by** *auto*
have $[A,e,B']_T\ ;;_{ICategory\ (CI\ T)}\ Fn[\![f]\!]_{CI\ T} = [A,f\ E@\ e,B]_T$
proof−
have *aa*: $Fn[\![f]\!]_{CI\ T} = [B',\ f\ E@\ Vx,\ B]_T$ **using** *f* **by** (*simp add*: *Canonical-Interpretation-def funsignature-abbrev-def*)
have $Sig\ iS_{CI\ T} \triangleright Vx : A \vdash e : B'$ **and** $Sig\ iS_{CI\ T} \triangleright Vx : B' \vdash f\ E@\ Vx : B$ **using** *sig f* **by** *auto*
hence $[A,e,B']_T\ ;;_{ICategory\ (CI\ T)}\ [B',\ f\ E@\ Vx,\ B]_T = [A,\ sub\ e\ in\ (f\ E@\ Vx),\ B]_T$
using *CIComp*[*of T*] *xt* **by** (*simp add*: *CanonicalInterpretation-def*)
moreover have $sub\ e\ in\ (f\ E@\ Vx) = f\ E@\ e$ **by** (*auto simp add*: *Subst-def*)
ultimately show *?thesis* **using** *aa* **by** *simp*
qed
thus $L[\![Vx : A \vdash f\ E@\ e : B]\!]_{CI\ T} \rightarrow (IMor\ [A,f\ E@\ e,B]_T)$ **using** *b* **by** *simp*
}
qed

lemma *CIModel*: $ZFAxioms\ T \Longrightarrow Model\ (CI\ T)\ T$
proof(*auto simp add*: *Model-def Model-axioms-def*)
assume *xt*: *ZFAxioms T*
have *axt*: *Axioms T* **using** *xt* **by** *simp*
show *ii*: *Interpretation* (*CI T*) **using** *xt* **by** (*simp add*: *CIInterpretation*)
show *aeq*: $aS_T = iS_{CI\ T}$ **by** (*simp add*: *CanonicalInterpretation-def*)
{
fix φ **assume** *a*: $\varphi \in aAxioms\ T$
from *a Axioms.AxT* **obtain** *A B e d* **where** φ: $\varphi = Vx : A \vdash e \equiv d : B$ **and** *sig*: $Sig\ aS_T \triangleright Vx : A \vdash e \equiv d : B$

using *axt* **by** *blast*
have *Sig* $aS_T \triangleright Vx : A \vdash e : B$ **using** *sig* **by** *auto*
hence *e*: $L[\![Vx : A \vdash e : B]\!]_{CI\ T} \to (IMor\ [A,\ e,\ B]_T)$ **using** *aeq CIInterp2Mor xt* **by** *simp*
have *Sig* $aS_T \triangleright Vx : A \vdash d : B$ **using** *sig* **by** *auto*
hence *d*: $L[\![Vx : A \vdash d : B]\!]_{CI\ T} \to (IMor\ [A,\ d,\ B]_T)$ **using** *aeq CIInterp2Mor xt* **by** *simp*
have $\varphi \in Axioms.Theory\ T$ **using** *a axt* **by** (*simp add*: *Axioms.Theory.Ax*)
hence $[A,\ e,\ B]_T = [A,\ d,\ B]_T$ **using** φ *Equiv2Cl axt* **by** *simp*
thus $L[\![\varphi]\!]_{CI\ T} \to (IBool\ True)$ **using** *e d* φ *ii InterpEq*[*of CI T A e B* $[A,\ e,\ B]_T$ *d* $[A,\ d,\ B]_T$] **by** *simp*
}
qed

lemma *CIComplete*: **assumes** *ZFAxioms T* **and** $L[\![\varphi]\!]_{CI\ T} \to (IBool\ True)$ **shows** $\varphi \in Axioms.Theory\ T$
proof−
have *ii*: *Interpretation* (*CI T*) **using** *assms* **by** (*simp add*: *CIInterpretation*)
hence *aeq*: $aS_T = iS_{CI\ T}$ **by** (*simp add*: *CanonicalInterpretation-def*)
from *Interpretation.Bool*[*of CI T* φ *True*] **obtain** *A B e d* **where** φ: $\varphi = (Vx : A \vdash e \equiv d : B)$ **using** *ii assms* **by** *auto*
from *Interpretation.Equiv*[*of CI T A e d B*] **obtain** *g*
where *g1*: $L[\![Vx : A \vdash e : B]\!]_{CI\ T} \to (IMor\ g)$
and *g2*: $L[\![Vx : A \vdash d : B]\!]_{CI\ T} \to (IMor\ g)$ **using** *ii* φ *assms* **by** *auto*
have *ewd*: *Sig* $iS_{CI\ T} \triangleright (Vx : A \vdash e : B)$ **and** *dwd*: *Sig* $iS_{CI\ T} \triangleright (Vx : A \vdash d : B)$
using *g1 g2 Interpretation.InterpExprWellDefined*[*of CI T*] *ii* **by** *simp+*
have *ie*: $L[\![Vx : A \vdash e : B]\!]_{CI\ T} \to (IMor\ [A,\ e,\ B]_T)$ **using** *ewd CIInterp2Mor assms* **by** *simp*
have *id*: $L[\![Vx : A \vdash d : B]\!]_{CI\ T} \to (IMor\ [A,\ d,\ B]_T)$ **using** *dwd CIInterp2Mor assms* **by** *simp*
have $[A,\ e,\ B]_T = g$
using *g1 ie ii Interpretation.Functional*[*of CI T Vx* : $A \vdash e : B$ *IMor* $[A,\ e,\ B]_T$ *IMor g*] **by** *simp*
moreover have $[A,\ d,\ B]_T = g$
using *g2 id ii Interpretation.Functional*[*of CI T Vx* : $A \vdash d : B$ *IMor* $[A,\ d,\ B]_T$ *IMor g*] **by** *simp*
ultimately have $[A,\ e,\ B]_T = [A,\ d,\ B]_T$ **by** *simp*
thus *?thesis* **using** φ *ewd Cl2Equiv aeq assms* **by** *simp*
qed

lemma *Complete*:
assumes *ZFAxioms T*
and $\bigwedge$ (*I* :: (*ZF*,*ZF*,*ZF*,*ZF*) *Interpretation*) . *Model I T* $\Longrightarrow$ $(L[\![\varphi]\!]_I \to (IBool\ True))$
shows $\varphi \in Axioms.Theory\ T$
proof−
have *Model* (*CI T*) *T* **using** *assms CIModel* **by** *simp*
thus *?thesis* **using** *CIComplete*[*of T* φ] *assms* **by** *auto*

qed

end

4 Functor

theory *Functors*
imports *Category*
begin

record $('o1, 'o2, 'm1, 'm2, 'a, 'b)$ *Functor* =
  CatDom :: $('o1,'m1,'a)$*Category-scheme*
  CatCod :: $('o2,'m2,'b)$*Category-scheme*
  MapM :: $'m1 \Rightarrow 'm2$

abbreviation
  FunctorMorApp :: $('o1, 'o2, 'm1, 'm2, 'a1, 'a2, 'a)$ *Functor-scheme* $\Rightarrow 'm1 \Rightarrow 'm2$ (**infixr** ‹##› 70) **where**
  FunctorMorApp F m $\equiv$ *(MapM F) m*

definition
  MapO :: $('o1, 'o2, 'm1, 'm2, 'a1, 'a2, 'a)$ *Functor-scheme* $\Rightarrow 'o1 \Rightarrow 'o2$ **where**
  MapO F X $\equiv$ *THE Y . Y* $\in$ *Obj(CatCod F)* $\wedge$ *F ## (Id (CatDom F) X) = Id (CatCod F) Y*

abbreviation
  FunctorObjApp :: $('o1, 'o2, 'm1, 'm2, 'a1, 'a2, 'a)$ *Functor-scheme* $\Rightarrow 'o1 \Rightarrow 'o2$ (**infixr** ‹@@› 70) **where**
  FunctorObjApp F X $\equiv$ *(MapO F X)*

locale *PreFunctor* =
  fixes *F* :: $('o1, 'o2, 'm1, 'm2, 'a1, 'a2, 'a)$ *Functor-scheme* (**structure**)
  assumes *FunctorComp*: $f \approx>_{CatDom\ F} g \Longrightarrow F \#\# (f ;;_{CatDom\ F} g) = (F \#\# f) ;;_{CatCod\ F} (F \#\# g)$
  and *FunctorId*: $X \in obj_{CatDom\ F} \Longrightarrow \exists\ Y \in obj_{CatCod\ F} .\ F \#\# (id_{CatDom\ F}\ X) = id_{CatCod\ F}\ Y$
  and *CatDom*[*simp*]: *Category(CatDom F)*
  and *CatCod*[*simp*]: *Category(CatCod F)*

locale *FunctorM* = *PreFunctor* +
  assumes *FunctorCompM*: $f\ maps_{CatDom\ F}\ X\ to\ Y \Longrightarrow (F \#\# f)\ maps_{CatCod\ F}\ (F @@ X)\ to\ (F @@ Y)$

locale *FunctorExt* =
  fixes *F* :: $('o1, 'o2, 'm1, 'm2, 'a1, 'a2, 'a)$ *Functor-scheme* (**structure**)
  assumes *FunctorMapExt*: *(MapM F)* $\in$ *extensional (Mor (CatDom F))*

locale *Functor* = *FunctorM* + *FunctorExt*

definition
MakeFtor :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*,*'r*) *Functor-scheme* ⇒ (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*,*'r*) *Functor-scheme* **where**
MakeFtor F ≡ (|
CatDom = *CatDom F* ,
CatCod = *CatCod F* ,
MapM = *restrict* (*MapM F*) (*Mor* (*CatDom F*)) ,
... = *Functor.more F*
|)

lemma *PreFunctorFunctor*[*simp*]: *Functor F* ⟹ *PreFunctor F*
by (*simp add*: *Functor-def FunctorM-def*)

lemmas *functor-simps* = *PreFunctor.FunctorComp PreFunctor.FunctorId*

definition
functor-abbrev (‹*Ftor* - : - ⟶ -› [*81*]) **where**
Ftor F : *A* ⟶ *B* ≡ (*Functor F*) ∧ (*CatDom F* = *A*) ∧ (*CatCod F* = *B*)

lemma *functor-abbrevE*[*elim*]: ⟦*Ftor F* : *A* ⟶ *B* ; ⟦(*Functor F*) ; (*CatDom F* = *A*) ; (*CatCod F* = *B*)⟧ ⟹ *R*⟧ ⟹ *R*
by (*auto simp add*: *functor-abbrev-def*)

definition
functor-comp-def (‹- ≈>;;; -› [*81*]) **where**
functor-comp-def F G ≡ (*Functor F*) ∧ (*Functor G*) ∧ (*CatDom G* = *CatCod F*)

lemma *functor-comp-def*[*elim*]: ⟦*F* ≈>;;; *G* ; ⟦*Functor F* ; *Functor G* ; *CatDom G* = *CatCod F*⟧ ⟹ *R*⟧ ⟹ *R*
by (*auto simp add*: *functor-comp-def-def*)

lemma (**in** *Functor*) *FunctorMapsTo*:
assumes *f* ∈ $mor_{CatDom\ F}$
shows *F* ## *f* $maps_{CatCod\ F}$ (*F* @@ ($dom_{CatDom\ F}$ *f*)) *to* (*F* @@ ($cod_{CatDom\ F}$ *f*))
proof−
have *f* $maps_{CatDom\ F}$ ($dom_{CatDom\ F}$ *f*) *to* ($cod_{CatDom\ F}$ *f*) **using** *assms* **by** *auto*
thus *?thesis* **by** (*simp add*: *FunctorCompM*[*of f* $dom_{CatDom\ F}$ *f* $cod_{CatDom\ F}$ *f*])
qed

lemma (**in** *Functor*) *FunctorCodDom*:
assumes *f* ∈ $mor_{CatDom\ F}$
shows $dom_{CatCod\ F}$(*F* ## *f*) = *F* @@ ($dom_{CatDom\ F}$ *f*) **and** $cod_{CatCod\ F}$(*F* ## *f*) = *F* @@ ($cod_{CatDom\ F}$ *f*)
proof−
have *F* ## *f* $maps_{CatCod\ F}$ (*F* @@ ($dom_{CatDom\ F}$ *f*)) *to* (*F* @@ ($cod_{CatDom\ F}$

f)) **using** *assms* **by** (*simp add*: *FunctorMapsTo*)
thus $dom_{CatCod\ F}(F\ \#\#\ f) = F\ @@\ (dom_{CatDom\ F}\ f)$ **and** $cod_{CatCod\ F}(F\ \#\#\ f) = F\ @@\ (cod_{CatDom\ F}\ f)$
by *auto*
qed

lemma (**in** *Functor*) *FunctorCompPreserved*: $f \in mor_{CatDom\ F} \Longrightarrow F\ \#\#\ f \in mor_{CatCod\ F}$
by (*auto dest*:*FunctorMapsTo*)

lemma (**in** *Functor*) *FunctorCompDef*:
assumes $f \approx>_{CatDom\ F} g$ **shows** $(F\ \#\#\ f) \approx>_{CatCod\ F} (F\ \#\#\ g)$
proof(*auto simp add*: *CompDefined-def*)
show $F\ \#\#\ f \in mor_{CatCod\ F}$ **and** $F\ \#\#\ g \in mor_{CatCod\ F}$ **using** *assms* **by** (*auto simp add*: *FunctorCompPreserved*)
have $f \in mor_{CatDom\ F}$ **and** $g \in mor_{CatDom\ F}$ **using** *assms* **by** *auto*
hence *a*: $cod_{CatCod\ F}\ (F\ \#\#\ f) = F\ @@\ (cod_{CatDom\ F}\ f)$ **and** *b*: $dom_{CatCod\ F}(F\ \#\#\ g) = F\ @@\ (dom_{CatDom\ F}\ g)$
by (*simp add*: *FunctorCodDom*)+
have $cod_{CatCod\ F}\ (F\ \#\#\ f) = F\ @@\ (dom_{CatDom\ F}\ g)$ **using** *assms a* **by** *auto*
also have $... = dom_{CatCod\ F}\ (F\ \#\#\ g)$ **using** *b* **by** *simp*
finally show $cod_{CatCod\ F}\ (F\ \#\#\ f) = dom_{CatCod\ F}\ (F\ \#\#\ g)$ **.**
qed

lemma *FunctorComp*: $[\![Ftor\ F : A \longrightarrow B\ ;\ f \approx>_A g]\!] \Longrightarrow F\ \#\#\ (f\ ;;_A\ g) = (F\ \#\#\ f)\ ;;_B\ (F\ \#\#\ g)$
by (*auto simp add*: *PreFunctor.FunctorComp*)

lemma *FunctorCompDef*: $[\![Ftor\ F : A \longrightarrow B\ ;\ f \approx>_A g]\!] \Longrightarrow (F\ \#\#\ f) \approx>_B (F\ \#\#\ g)$
by (*auto simp add*: *Functor.FunctorCompDef*)

lemma *FunctorMapsTo*:
assumes $Ftor\ F : A \longrightarrow B$ **and** $f\ maps_A\ X\ to\ Y$
shows $(F\ \#\#\ f)\ maps_B\ (F\ @@\ X)\ to\ (F\ @@\ Y)$
proof−
have *b*: $CatCod\ F = B$ **and** *a*: $CatDom\ F = A$ **and** *ff*: $Functor\ F$ **using** *assms* **by** *auto*
have *df*: $(dom_{CatDom\ F}\ f) = X$ **and** *cf*: $(cod_{CatDom\ F}\ f) = Y$ **using** *a assms* **by** *auto*
have $f \in mor_{CatDom\ F}$ **using** *assms* **by** *auto*
hence $F\ \#\#\ f\ maps_{CatCod\ F}\ (F\ @@\ (dom_{CatDom\ F}\ f))\ to\ (F\ @@\ (cod_{CatDom\ F}\ f))$ **using** *ff*
by (*simp add*: *Functor.FunctorMapsTo*)
thus *?thesis* **using** *df cf b* **by** *simp*
qed

lemma (**in** *PreFunctor*) *FunctorId2*:
assumes $X \in obj_{CatDom\ F}$

shows F @@ $X \in obj_{CatCod\ F} \wedge F$ ## $(id_{CatDom\ F}\ X) = id_{CatCod\ F}\ (F$ @@ $X)$
proof−
let *?Q* $= \lambda\ E\ Y\ .\ Y \in obj_{CatCod\ F} \wedge E = id_{CatCod\ F}\ Y$
let *?P* $=$ *?Q* $(F$ ## $(id_{CatDom\ F}\ X))$
from *assms FunctorId* **obtain** Y **where** *?P* Y **by** *auto*
moreover **{**
fix $y\ e\ z$ **have** $[\![$*?Q* $e\ y$; *?Q* $e\ z]\!] \Longrightarrow y = z$
by (*auto intro*: *Category.IdInj*[*of CatCod F y z*])
}
ultimately have $\exists!\ Z\ .$ *?P* Z **by** *auto*
hence *?P* $(THE\ Y\ .$ *?P* $Y)$ **by** (*rule theI'*)
thus *?thesis* **by** (*auto simp add*: *MapO-def*)
qed

lemma *FunctorId*:
assumes *Ftor* $F : C \longrightarrow D$ **and** $X \in Obj\ C$
shows F ## $(Id\ C\ X) = Id\ D\ (F$ @@ $X)$
proof−
have $CatDom\ F = C$ **and** $CatCod\ F = D$ **and** *PreFunctor* F **using** *assms* **by** *auto*
thus *?thesis* **using** *assms PreFunctor.FunctorId2*[*of F X*] **by** *simp*
qed

lemma (**in** *Functor*) *DomFunctor*: $f \in mor_{CatDom\ F} \Longrightarrow dom_{CatCod\ F}\ (F$ ## $f) = F$ @@ $(dom_{CatDom\ F}\ f)$
by (*simp add*: *FunctorCodDom*)

lemma (**in** *Functor*) *CodFunctor*: $f \in mor_{CatDom\ F} \Longrightarrow cod_{CatCod\ F}\ (F$ ## $f)$ $= F$ @@ $(cod_{CatDom\ F}\ f)$
by (*simp add*: *FunctorCodDom*)

lemma (**in** *Functor*) *FunctorId3Dom*:
assumes $f \in mor_{CatDom\ F}$
shows F ## $(id_{CatDom\ F}\ (dom_{CatDom\ F}\ f)) = id_{CatCod\ F}\ (dom_{CatCod\ F}\ (F$ ## $f))$
proof−
have $(dom_{CatDom\ F}\ f) \in obj_{CatDom\ F}$ **using** *assms* **by** (*simp add*: *Category.Cdom*)
hence F ## $(id_{CatDom\ F}\ (dom_{CatDom\ F}\ f)) = id_{CatCod\ F}\ (F$ @@ $(dom_{CatDom\ F}\ f))$ **by** (*simp add*: *FunctorId2*)
also have ... $= id_{CatCod\ F}\ (dom_{CatCod\ F}\ (F$ ## $f))$ **using** *assms* **by** (*simp add*: *DomFunctor*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *Functor*) *FunctorId3Cod*:
assumes $f \in mor_{CatDom\ F}$
shows F ## $(id_{CatDom\ F}\ (cod_{CatDom\ F}\ f)) = id_{CatCod\ F}\ (cod_{CatCod\ F}\ (F$ ## $f))$

proof−
have ($cod_{CatDom\ F}\ f$) ∈ $obj_{CatDom\ F}$ **using** *assms* **by** (*simp add*: *Category.Ccod*)
hence F ## ($id_{CatDom\ F}$ ($cod_{CatDom\ F}\ f$)) = $id_{CatCod\ F}$ (F @@ ($cod_{CatDom\ F}$ f)) **by** (*simp add*: *FunctorId2*)
also have ... = $id_{CatCod\ F}$ ($cod_{CatCod\ F}$ (F ## f)) **using** *assms* **by** (*simp add*: *CodFunctor*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *PreFunctor*) *FmToFo*: ⟦X ∈ $obj_{CatDom\ F}$; Y ∈ $obj_{CatCod\ F}$; F ## ($id_{CatDom\ F}$ X) = $id_{CatCod\ F}$ Y⟧ ⟹ F @@ X = Y
by (*auto simp add*: *FunctorId2 intro*: *Category.IdInj*[*of CatCod F F* @@ *X Y*])

lemma *MakeFtorPreFtor*:
assumes *PreFunctor F* **shows** *PreFunctor* (*MakeFtor F*)
proof−
{
fix X **assume** *a*: X ∈ $obj_{CatDom\ F}$ **have** $id_{CatDom\ F}$ X ∈ $mor_{CatDom\ F}$
proof−
have *Category* (*CatDom F*) **using** *assms* **by** (*simp add*: *PreFunctor-def*)
hence $id_{CatDom\ F}$ X *maps*$_{CatDom\ F}$ X *to* X **using** *a* **by** (*simp add*: *Category.Cidm*)
thus *?thesis* **using** *a* **by** (*auto*)
qed
}
thus *PreFunctor* (*MakeFtor F*) **using** *assms*
by(*auto simp add*: *PreFunctor-def MakeFtor-def Category.MapsToMorDomCod*)
qed

lemma *MakeFtorMor*: f ∈ $mor_{CatDom\ F}$ ⟹ *MakeFtor F* ## f = F ## f
by(*simp add*: *MakeFtor-def*)

lemma *MakeFtorObj*:
assumes *PreFunctor F* **and** X ∈ $obj_{CatDom\ F}$
shows *MakeFtor F* @@ X = F @@ X
proof−
have X ∈ $obj_{CatDom\ (MakeFtor\ F)}$ **using** *assms*(*2*) **by** (*simp add*: *MakeFtor-def*)
moreover have (F @@ X) ∈ $obj_{CatCod\ (MakeFtor\ F)}$ **using** *assms* **by** (*simp add*: *PreFunctor.FunctorId2 MakeFtor-def*)
moreover have *MakeFtor F* ## $id_{CatDom\ (MakeFtor\ F)}$ X = $id_{CatCod\ (MakeFtor\ F)}$ (F @@ X)
proof−
have *Category* (*CatDom F*) **using** *assms*(*1*) **by** (*simp add*: *PreFunctor-def*)
hence $id_{CatDom\ F}$ X *maps*$_{CatDom\ F}$ X *to* X **using** *assms*(*2*) **by** (*auto simp add*: *Category.Cidm*)
hence $id_{CatDom\ F}$ X ∈ $mor_{CatDom\ F}$ **by** *auto*
hence *MakeFtor F* ## $id_{CatDom\ (MakeFtor\ F)}$ X = F ## $id_{CatDom\ F}$ X **by** (*simp add*: *MakeFtor-def*)
also have ... = $id_{CatCod\ F}$ (F @@ X) **using** *assms* **by** (*simp add*: *PreFunc-*

tor.FunctorId2)
finally show *?thesis* **by** (*simp add*: *MakeFtor-def*)
qed
moreover have *PreFunctor* (*MakeFtor F*) **using** *assms*(*1*) **by** (*simp add*: *MakeFtor-PreFtor*)
ultimately show *?thesis* **by** (*simp add*: *PreFunctor.FmToFo*)
qed

lemma *MakeFtor*: **assumes** *FunctorM F* **shows** *Functor* (*MakeFtor F*)
proof(*intro-locales*)
show *PreFunctor* (*MakeFtor F*) **using** *assms* **by** (*simp add*: *MakeFtorPreFtor FunctorM-def*)
show *FunctorM-axioms* (*MakeFtor F*)
proof(*auto simp add*: *FunctorM-axioms-def*)
{
fix *f X Y* **assume** *aa*: *f maps*$_{CatDom\ (MakeFtor\ F)}$ *X to Y*
show ((*MakeFtor F*) *##* *f*) *maps*$_{CatCod\ (MakeFtor\ F)}$ ((*MakeFtor F*) *@@* *X*) *to* ((*MakeFtor F*) *@@* *Y*)
proof−
have ((*MakeFtor F*) *##* *f*) = *F ##* *f* **using** *aa* **by** (*auto simp add*: *MakeFtor-def*)
moreover have ((*MakeFtor F*) *@@* *X*) = *F @@* *X* **and** ((*MakeFtor F*) *@@* *Y*) = *F @@* *Y*
proof−
have *Category* (*CatDom F*) **using** *assms* **by** (*simp add*: *FunctorM-def PreFunctor-def*)
hence *X* ∈ *obj*$_{CatDom\ F}$ **and** *Y* ∈ *obj*$_{CatDom\ F}$
using *aa* **by** (*auto simp add*: *Category.MapsToObj MakeFtor-def*)
moreover have *PreFunctor F* **using** *assms*(*1*) **by** (*simp add*: *FunctorM-def*)
ultimately show ((*MakeFtor F*) *@@* *X*) = *F @@* *X* **and** ((*MakeFtor F*) *@@* *Y*) = *F @@* *Y*
by (*simp add*: *MakeFtorObj*)+
qed
moreover have *F ##* *f maps*$_{CatCod\ F}$ (*F @@* *X*) *to* (*F @@* *Y*) **using** *assms*(*1*) *aa*
by (*simp add*: *FunctorM.FunctorCompM MakeFtor-def*)
ultimately show *?thesis* **by** (*simp add*: *MakeFtor-def*)
qed
}
qed
show *FunctorExt* (*MakeFtor F*) **by**(*simp add*: *FunctorExt-def MakeFtor-def*)
qed

definition
IdentityFunctor′ :: ('*o*,'*m*,'*a*) *Category-scheme* ⇒ ('*o*,'*o*,'*m*,'*m*,'*a*,'*a*) *Functor* (‹*FId″* -› [*70*]) **where**
IdentityFunctor′ C ≡ (|*CatDom* = *C* , *CatCod* = *C* , *MapM* = (λ *f* . *f*) |)

definition
IdentityFunctor (‹*FId* -› [70]) **where**
IdentityFunctor C ≡ *MakeFtor*(*IdentityFunctor′ C*)

lemma *IdFtor′PreFunctor*: *Category C* ⟹ *PreFunctor* (*FId′ C*)
by(*auto simp add*: *PreFunctor-def IdentityFunctor′-def*)

lemma *IdFtor′Obj*:
assumes *Category C* **and** *X* ∈ $obj_{CatDom\ (FId'\ C)}$
shows (*FId′ C*) @@ *X* = *X*
proof−
have (*FId′ C*) ## $id_{CatDom\ (FId'\ C)}$ *X* = $id_{CatCod\ (FId'\ C)}$ *X* **by**(*simp add*: *IdentityFunctor′-def*)
moreover have *X* ∈ $obj_{CatCod\ (FId'\ C)}$ **using** *assms* **by** (*simp add*: *IdentityFunctor′-def*)
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *PreFunctor.FmToFo IdFtor′PreFunctor*)
qed

lemma *IdFtor′FtorM*:
assumes *Category C* **shows** *FunctorM* (*FId′ C*)
proof(*auto simp add*: *FunctorM-def IdFtor′PreFunctor assms FunctorM-axioms-def*)
{
fix *f X Y* **assume** *a*: *f* $maps_{CatDom\ (FId'\ C)}$ *X to Y*
show ((*FId′ C*) ## *f*) $maps_{CatCod\ (FId'\ C)}$ ((*FId′ C*) @@ *X*) *to* ((*FId′ C*) @@ *Y*)
proof−
have *X* ∈ $obj_{CatDom\ (FId'\ C)}$ **and** *Y* ∈ $obj_{CatDom\ (FId'\ C)}$
using *a assms* **by** (*simp add*: *Category.MapsToObj IdentityFunctor′-def*)+
moreover have (*FId′ C*) ## *f* = *f* **and** *CatDom* (*FId′ C*) = *CatCod* (*FId′ C*) **by** (*simp add*: *IdentityFunctor′-def*)+
ultimately show *?thesis* **using** *assms a* **by**(*simp add*: *IdFtor′Obj*)
qed
}
qed

lemma *IdFtorFtor*: *Category C* ⟹ *Functor* (*FId C*)
by (*auto simp add*: *IdentityFunctor-def IdFtor′FtorM intro*: *MakeFtor*)

definition
ConstFunctor′ :: ('*o1*,'*m1*,'*a*) *Category-scheme* ⇒
('*o2*,'*m2*,'*b*) *Category-scheme* ⇒ '*o2* ⇒ ('*o1*,'*o2*,'*m1*,'*m2*,'*a*,'*b*) *Functor* **where**
ConstFunctor′ A B b ≡ (|
CatDom = *A* ,
CatCod = *B* ,
MapM = (λ *f* . (*Id B*) *b*)
|)

definition *ConstFunctor A B b* ≡ *MakeFtor*(*ConstFunctor′ A B b*)

lemma *ConstFtor′* :
assumes *Category A Category B b* ∈ (*Obj B*)
shows *PreFunctor* (*ConstFunctor′ A B b*)
and *FunctorM* (*ConstFunctor′ A B b*)
proof−
show *PreFunctor* (*ConstFunctor′ A B b*) **using** *assms*
apply (*subst PreFunctor-def*)
apply (*rule conjI*)+
by (*auto simp add*: *ConstFunctor′-def Category.Simps Category.CatIdCompId*)
moreover
{
fix *X* **assume** *X* ∈ *obj*$_A$ *b* ∈ *obj*$_B$ *PreFunctor* (*ConstFunctor′ A B b*)
hence (*ConstFunctor′ A B b*) @@ *X* = *b*
by (*auto simp add*: *ConstFunctor′-def PreFunctor.FmToFo Category.Simps*)
}
ultimately show *FunctorM* (*ConstFunctor′ A B b*) **using** *assms*
by (*intro-locales*, *auto simp add*: *ConstFunctor′-def Category.Simps FunctorM-axioms-def*)
qed

lemma *ConstFtor*:
assumes *Category A Category B b* ∈ (*Obj B*)
shows *Functor* (*ConstFunctor A B b*)
by (*auto simp add*: *assms ConstFtor′ ConstFunctor-def MakeFtor*)

definition
UnitFunctor :: ('*o*,'*m*,'*a*) *Category-scheme* ⇒ ('*o*,*unit*,'*m*,*unit*,'*a*,*unit*) *Functor*
where
UnitFunctor C ≡ *ConstFunctor C UnitCategory* ()

lemma *UnitFtor*:
assumes *Category C*
shows *Functor*(*UnitFunctor C*)
proof−
have () ∈ *obj*$_{UnitCategory}$ **by** (*simp add*: *UnitCategory-def MakeCatObj*)
hence *Functor*(*ConstFunctor C UnitCategory* ()) **using** *assms*
by (*simp add*: *ConstFtor*)
thus *?thesis* **by** (*simp add*: *UnitFunctor-def*)
qed

definition
FunctorComp′ :: ('*o1*,'*o2*,'*m1*,'*m2*,'*a1*,'*a2*) *Functor* ⇒ ('*o2*,'*o3*,'*m2*,'*m3*,'*b1*,'*b2*) *Functor*
⇒ ('*o1*,'*o3*,'*m1*,'*m3*,'*a1*,'*b2*) *Functor* (**infixl** ‹;;:› *71*) **where**
FunctorComp′ F G ≡ ⦇
CatDom = *CatDom F* ,
CatCod = *CatCod G* ,

$MapM \quad = \lambda\, f\, .\, (MapM\ G)((MapM\ F)\ f)$

⟫

definition *FunctorComp* (**infixl** ‹;;;› *71*) **where** *FunctorComp F G* ≡ *MakeFtor* (*FunctorComp′ F G*)

lemma *FtorCompComp′*:
assumes $f \approx>_{CatDom\ F} g$
and $F \approx>;;;\ G$
shows $G\ \#\#\ (F\ \#\#\ (f\ ;;_{CatDom\ F}\ g)) = (G\ \#\#\ (F\ \#\#\ f))\ ;;_{CatCod\ G}\ (G\ \#\#\ (F\ \#\#\ g))$
proof−
have [*simp*]: *PreFunctor G* ∧ *PreFunctor F* **using** *assms* **by** *auto*
have [*simp*]: $(F\ \#\#\ f) \approx>_{CatDom\ G}\ (F\ \#\#\ g)$ **using** *assms* **by** (*auto simp add*: *Functor.FunctorCompDef*[*of F f g*])
have $F\ \#\#\ (f\ ;;_{CatDom\ F}\ g) = (F\ \#\#\ f)\ ;;_{CatCod\ F}\ (F\ \#\#\ g)$ **using** *assms*
by (*auto simp add*: *PreFunctor.FunctorComp*)
hence $G\ \#\#\ (F\ \#\#\ (f\ ;;_{CatDom\ F}\ g)) = G\ \#\#\ ((F\ \#\#\ f)\ ;;_{CatCod\ F}\ (F\ \#\#\ g))$ **by** *simp*
also have ... $= G\ \#\#\ ((F\ \#\#\ f)\ ;;_{CatDom\ G}\ (F\ \#\#\ g))$ **using** *assms* **by** *auto*
also have ... $= (G\ \#\#\ (F\ \#\#\ f))\ ;;_{CatCod\ G}\ (G\ \#\#\ (F\ \#\#\ g))$
by (*simp add*: *PreFunctor.FunctorComp*[*of G* (*F* ## *f*) (*F* ## *g*)])
finally show *?thesis* **by** *simp*
qed

lemma *FtorCompId*:
assumes *a*: $X \in (Obj\ (CatDom\ F))$
and $F \approx>;;;\ G$
shows $G\ \#\#\ (F\ \#\#\ (id_{CatDom\ F}\ X)) = id_{CatCod\ G}(G\ @@\ (F\ @@\ X)) \wedge G\ @@\ (F\ @@\ X) \in (Obj\ (CatCod\ G))$
proof−
have [*simp*]: *PreFunctor G* ∧ *PreFunctor F* **using** *assms* **by** *auto*
have *b*: $(F\ @@\ X) \in obj_{CatDom\ G}$ **using** *assms*
by (*auto simp add*: *PreFunctor.FunctorId2*)
have $G\ \#\#\ F\ \#\#\ (id_{CatDom\ F}\ X) = G\ \#\#\ (id_{CatCod\ F}(F\ @@\ X))$ **using** *b a*
by (*simp add*: *PreFunctor.FunctorId2*[*of F X*])
also have ... $= G\ \#\#\ (id_{CatDom\ G}(F\ @@\ X))$ **using** *assms* **by** *auto*
also have ... $= id_{CatCod\ G}(G\ @@\ (F\ @@\ X)) \wedge G\ @@\ (F\ @@\ X) \in (Obj\ (CatCod\ G))$ **using** *b*
by (*simp add*: *PreFunctor.FunctorId2*[*of G* (*F* @@ *X*)])
finally show *?thesis* **by** *simp*
qed

lemma *FtorCompIdDef*:
assumes *a*: $X \in (Obj\ (CatDom\ F))$ **and** *b*: *PreFunctor* (*F* ;;: *G*)
and $F \approx>;;;\ G$
shows $(F\ ;;:\ G)\ @@\ X = (G\ @@\ (F\ @@\ X))$
proof−

have $(F \;;;:\; G) \;\#\#\; (id_{CatDom\ (F\ ;;;:\ G)}(X)) = G \;\#\#\; (F \;\#\#\; (id_{CatDom\ F}(X)))$
using *assms*
by (*simp add*: *FunctorComp′-def*)
also have ... $= id_{CatCod\ G}(G\ @@\ (F\ @@\ (X)))$ **using** *assms a*
by(*auto simp add*: *FtorCompId*[*of* - *F G*])
finally have $(F \;;;:\; G) \;\#\#\; (id_{CatDom\ (F\ ;;;:\ G)}(X)) = id_{CatCod\ (F\ ;;;:\ G)}(G\ @@\ (F\ @@\ X))$ **using** *assms*
by (*simp add*: *FunctorComp′-def*)
moreover have $G\ @@\ (F\ @@\ (X)) \in (Obj\ (CatCod\ (F \;;;:\; G)))$ **using** *assms a*
by(*auto simp add*: *FtorCompId*[*of* - *F G*] *FunctorComp′-def*)
moreover have $X \in obj_{CatDom\ (F\ ;;;:\ G)}$ **using** *a* **by** (*simp add*: *Functor-Comp′-def*)
ultimately show *?thesis* **using** *b*
by (*simp add*: *PreFunctor.FmToFo*[*of F* $;;;:$ *G X G* $@@\ (F\ @@\ X)$])
qed

lemma *FunctorCompMapsTo*:
assumes $f \in mor_{CatDom\ (F\ ;;;:\ G)}$ **and** $F \approx>;;;\ G$
shows $(G \;\#\#\; (F \;\#\#\; f))\ maps_{CatCod\ G}\ (G\ @@\ (F\ @@\ (dom_{CatDom\ F}\ f)))\ to\ (G\ @@\ (F\ @@\ (cod_{CatDom\ F}\ f)))$
proof−
have $f \in mor_{CatDom\ F} \wedge Functor\ F$ **using** *assms* **by** (*auto simp add*: *Functor-Comp′-def*)
hence $(F \;\#\#\; f)\ maps_{CatDom\ G}\ (F\ @@\ (dom_{CatDom\ F}\ f))\ to\ (F\ @@\ (cod_{CatDom\ F}\ f))$ **using** *assms*
by (*auto simp add*: *Functor.FunctorMapsTo*[*of F f*])
moreover have *FunctorM G* **using** *assms* **by** (*auto simp add*: *FunctorComp-def Functor-def*)
ultimately show *?thesis* **by**(*simp add*: *FunctorM.FunctorCompM*[*of G F* $\#\#$ *f F* $@@\ (dom_{CatDom\ F}\ f)\ F\ @@\ (cod_{CatDom\ F}\ f)$])
qed

lemma *FunctorCompMapsTo2*:
assumes $f \in mor_{CatDom\ (F\ ;;;:\ G)}$
and $F \approx>;;;\ G$
and *PreFunctor* $(F \;;;:\; G)$
shows $((F \;;;:\; G) \;\#\#\; f)\ maps_{CatCod\ (F\ ;;;:\ G)}\ ((F \;;;:\; G)\ @@\ (dom_{CatDom\ (F\ ;;;:\ G)}\ f))\ to$
$((F \;;;:\; G)\ @@\ (cod_{CatDom\ (F\ ;;;:\ G)}\ f))$
proof−
have *Category* $(CatDom\ (F \;;;:\; G))$ **using** *assms* **by** (*simp add*: *PreFunctor-def*)

hence *1*: $(dom_{CatDom\ (F\ ;;;:\ G)}\ f) \in obj_{CatDom\ F} \wedge (cod_{CatDom\ (F\ ;;;:\ G)}\ f) \in obj_{CatDom\ F}$ **using** *assms*
by (*auto simp add*: *Category.Simps FunctorComp′-def*)
have $(G \;\#\#\; (F \;\#\#\; f))\ maps_{CatCod\ G}\ (G\ @@\ (F\ @@\ (dom_{CatDom\ F}\ f)))\ to\ (G\ @@\ (F\ @@\ (cod_{CatDom\ F}\ f)))$
using *assms* **by** (*auto simp add*: *FunctorCompMapsTo*[*of f F G*])

moreover have $CatDom\ F = CatDom(F\ ;;:\ G) \wedge CatCod\ G = CatCod(F\ ;;:\ G) \wedge (G\ \#\#\ (F\ \#\#\ f)) = ((F\ ;;:\ G)\ \#\#\ f)$ **using** *assms*
by (*simp add*: *FunctorComp′-def*)
moreover have $(F\ ;;:\ G)\ @@\ (dom_{CatDom\ (F\ ;;:\ G)}\ f) = (G\ @@\ (F\ @@\ (dom_{CatDom(F\ ;;:\ G)}\ f))) \wedge$
$(F\ ;;:\ G)\ @@\ (cod_{CatDom\ (F\ ;;:\ G)}\ f) = (G\ @@\ (F\ @@\ (cod_{CatDom(F\ ;;:\ G)}\ f)))$
by (*auto simp add*: *FtorCompIdDef*[*of - F G*] *1 assms*)
ultimately show *?thesis* **by** *auto*
qed

lemma *FunctorCompMapsTo3*:
assumes $f\ maps_{CatDom\ (F\ ;;:\ G)}\ X\ to\ Y$
and $F \approx>;;;\ G$
and *PreFunctor* $(F\ ;;:\ G)$
shows $F\ ;;:\ G\ \#\#\ f\ maps_{CatCod\ (F\ ;;:\ G)}\ F\ ;;:\ G\ @@\ X\ to\ F\ ;;:\ G\ @@\ Y$
proof−
have $f \in mor_{CatDom\ (F\ ;;:\ G)}$
and $dom_{CatDom\ (F\ ;;:\ G)}\ f = X$
and $cod_{CatDom\ (F\ ;;:\ G)}\ f = Y$ **using** *assms* **by** *auto*
thus *?thesis* **using** *assms* **by** (*auto intro*: *FunctorCompMapsTo2*)
qed

lemma *FtorCompPreFtor*:
assumes $F \approx>;;;\ G$
shows *PreFunctor* $(F\ ;;:\ G)$
proof−
have *1*: *PreFunctor G* $\wedge$ *PreFunctor F* **using** *assms* **by** *auto*
show *PreFunctor* $(F\ ;;:\ G)$ **using** *assms*
proof(*auto simp add*: *PreFunctor-def FunctorComp′-def Category.Simps FtorCompId*[*of - F G*] *intro*:*FtorCompComp′*)
show *Category* $(CatDom\ F)$ **and** *Category* $(CatCod\ G)$ **using** *assms 1* **by** (*auto simp add*: *PreFunctor-def*)
qed
qed

lemma *FtorCompM* :
assumes $F \approx>;;;\ G$
shows *FunctorM* $(F\ ;;:\ G)$
proof(*auto simp only*: *FunctorM-def*)
show *1*: *PreFunctor* $(F\ ;;:\ G)$ **using** *assms* **by** (*rule FtorCompPreFtor*)
{
fix *X Y f* **assume** *a*: $f\ maps_{CatDom\ (F\ ;;:\ G)}\ X\ to\ Y$
have $F\ ;;:\ G\ \#\#\ f\ maps_{CatCod\ (F\ ;;:\ G)}\ F\ ;;:\ G\ @@\ X\ to\ F\ ;;:\ G\ @@\ Y$
using *a assms 1* **by** (*rule FunctorCompMapsTo3*)
}
thus *FunctorM-axioms* $(F\ ;;:\ G)$
by(*auto simp add*: *1 FunctorM-axioms-def*)

qed

lemma *FtorComp*:
assumes $F \approx>;;;\ G$
shows *Functor* $(F\ ;;;\ G)$
proof−
have *FunctorM* $(F\ ;;:\ G)$ **using** *assms* **by** (*rule FtorCompM*)
thus *?thesis* **by** (*simp add*: *FunctorComp-def MakeFtor*)
qed

lemma (**in** *Functor*) *FunctorPreservesIso*:
assumes $ciso_{CatDom\ F}\ k$
shows $ciso_{CatCod\ F}\ (F\ \#\#\ k)$
proof−
have [*simp*]: $k \in mor_{CatDom\ F}$ **using** *assms* **by** (*simp add*: *Category.IsoIsMor*)
have $cinv_{CatCod\ F}\ (F\ \#\#\ k)\ (F\ \#\#\ (Cinv_{CatDom\ F}\ k))$
proof(*rule Category.Inverse-relI*)
show *Category* (*CatCod F*) **by** *simp*
show $(F\ \#\#\ k) \approx>_{CatCod\ F} (F\ \#\#\ (Cinv_{CatDom\ F}\ k))$
by (*rule FunctorCompDef*, *simp add*: *Category.IsoCompInv assms*)
show $(F\ \#\#\ k)\ ;;_{CatCod\ F}\ (F\ \#\#\ (Cinv_{CatDom\ F}\ k)) = id_{CatCod\ F}\ (dom_{CatCod\ F}\ (F\ \#\#\ k))$
proof−
have $(F\ \#\#\ k)\ ;;_{CatCod\ F}\ (F\ \#\#\ (Cinv_{CatDom\ F}\ k)) = F\ \#\#\ (k\ ;;_{CatDom\ F}\ (Cinv_{CatDom\ F}\ k))$ **using** *assms*
by(*auto simp add*: *FunctorComp Category.IsoCompInv*)
also have $... = F\ \#\#\ (id_{CatDom\ F}\ (dom_{CatDom\ F}\ k))$ **using** *assms* **by** (*simp add*: *Category.IsoInvId2*)
also have $... = id_{CatCod\ F}\ (dom_{CatCod\ F}\ (F\ \#\#\ k))$ **by** (*simp add*: *FunctorId3Dom*)
finally show *?thesis* **by** *simp*
qed
show $(F\ \#\#\ (Cinv_{CatDom\ F}\ k))\ ;;_{CatCod\ F}\ (F\ \#\#\ k) = id_{CatCod\ F}\ (cod_{CatCod\ F}\ (F\ \#\#\ k))$
proof−
have $(F\ \#\#\ (Cinv_{CatDom\ F}\ k))\ ;;_{CatCod\ F}\ (F\ \#\#\ k) = F\ \#\#\ ((Cinv_{CatDom\ F}\ k)\ ;;_{CatDom\ F}\ k)$ **using** *assms*
by(*auto simp add*: *FunctorComp Category.InvCompIso*)
also have $... = F\ \#\#\ (id_{CatDom\ F}\ (cod_{CatDom\ F}\ k))$ **using** *assms* **by** (*simp add*: *Category.IsoInvId1*)
also have $... = id_{CatCod\ F}\ (cod_{CatCod\ F}\ (F\ \#\#\ k))$ **using** *assms* **by** (*simp add*: *FunctorId3Cod*)
finally show *?thesis* **by** *simp*
qed
qed
thus *?thesis* **by**(*auto simp add*: *isomorphism-def*)
qed

declare *PreFunctor.CatDom*[*simp*] *PreFunctor.CatCod* [*simp*]

lemma *FunctorMFunctor*[*simp*]: *Functor F* $\Longrightarrow$ *FunctorM F*
by (*simp add*: *Functor-def*)

locale *Equivalence* = *Functor* +
  assumes *Full*: $[\![A \in Obj\ (CatDom\ F)\ ;\ B \in Obj\ (CatDom\ F)\ ;$
    $h\ maps_{CatCod\ F}\ (F\ @@\ A)\ to\ (F\ @@\ B)]\!] \Longrightarrow$
    $\exists\ f\ .\ (f\ maps_{CatDom\ F}\ A\ to\ B) \wedge (F\ \#\#\ f = h)$
  and *Faithful*: $[\![f\ maps_{CatDom\ F}\ A\ to\ B\ ;\ g\ maps_{CatDom\ F}\ A\ to\ B\ ;\ F\ \#\#\ f = F\ \#\#\ g]\!] \Longrightarrow f = g$
  and *IsoDense*: $C \in Obj\ (CatCod\ F) \Longrightarrow \exists\ A \in Obj\ (CatDom\ F)\ .\ ObjIso\ (CatCod\ F)\ (F\ @@\ A)\ C$

end

5 Natural Transformation

theory *NatTrans*
imports *Functors*
begin

record (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*) *NatTrans* =
  NTDom :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*) *Functor*
  NTCod :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*) *Functor*
  NatTransMap :: *'o1* $\Rightarrow$ *'m2*

abbreviation
  NatTransApp :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*) *NatTrans* $\Rightarrow$ *'o1* $\Rightarrow$ *'m2* (**infixr** ‹$$› *70*) **where**
  NatTransApp η *X* $\equiv$ (*NatTransMap* η) *X*

definition *NTCatDom* $\eta \equiv$ *CatDom* (*NTDom* η)
definition *NTCatCod* $\eta \equiv$ *CatCod* (*NTCod* η)

locale *NatTransExt* =
  fixes η :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*) *NatTrans* (**structure**)
  assumes *NTExt* : *NatTransMap* $\eta \in$ *extensional* (*Obj* (*NTCatDom* η))

locale *NatTransP* =
  fixes η :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a*, *'b*) *NatTrans* (**structure**)
  assumes *NatTransFtor*: *Functor* (*NTDom* η)
  and *NatTransFtor2*: *Functor* (*NTCod* η)
  and *NatTransFtorDom*: *NTCatDom* η = *CatDom* (*NTCod* η)
  and *NatTransFtorCod*: *NTCatCod* η = *CatCod* (*NTDom* η)
  and *NatTransMapsTo*: *X* $\in obj_{NTCatDom\ \eta} \Longrightarrow$
    (η $$ *X*) $maps_{NTCatCod\ \eta}$ ((*NTDom* η) @@ *X*) *to* ((*NTCod* η) @@ *X*)
  and *NatTrans*: *f* $maps_{NTCatDom\ \eta}$ *X to Y* $\Longrightarrow$

$((NTDom\ \eta)\ \#\#\ f)\ ;;_{NTCatCod\ \eta}\ (\eta\ \$\$\ Y) = (\eta\ \$\$\ X)\ ;;_{NTCatCod\ \eta}$
$((NTCod\ \eta)\ \#\#\ f)$

locale *NatTrans* = *NatTransP* + *NatTransExt*

lemma [*simp*]: *NatTrans* $\eta \Longrightarrow$ *NatTransP* η
by(*simp add*: *NatTrans-def*)

definition *MakeNT* :: ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *NatTrans* $\Rightarrow$ ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *NatTrans* **where**
MakeNT $\eta \equiv$ (|
NTDom = *NTDom* η ,
NTCod = *NTCod* η ,
NatTransMap = *restrict* (*NatTransMap* η) (*Obj* (*NTCatDom* η))
|)

definition
nt-abbrev (‹*NT* - : - $\Longrightarrow$ -› [*81*]) **where**
NT f : $F \Longrightarrow G \equiv$ (*NatTrans* f) $\wedge$ (*NTDom* $f = F$) $\wedge$ (*NTCod* $f = G$)

lemma *nt-abbrevE*[*elim*]: $[\![NT\ f : F \Longrightarrow G\ ;\ [\![(NatTrans\ f)\ ;\ (NTDom\ f = F)\ ;\ (NTCod\ f = G)]\!] \Longrightarrow R]\!] \Longrightarrow R$
by (*auto simp add*: *nt-abbrev-def*)

lemma *MakeNT*: *NatTransP* $\eta \Longrightarrow$ *NatTrans* (*MakeNT* η)
by(*auto simp add*: *NatTransP-def NatTrans-def MakeNT-def NTCatDom-def NTCatCod-def Category.MapsToObj*
NatTransExt-def)

lemma *MakeNT-comp*: $X \in$ *Obj* (*NTCatDom* f) $\Longrightarrow$ (*MakeNT* f) \$\$ $X = f$ \$\$ X
by (*simp add*: *MakeNT-def*)

lemma *MakeNT-dom*: *NTCatDom* f = *NTCatDom* (*MakeNT* f)
by (*simp add*: *NTCatDom-def MakeNT-def*)

lemma *MakeNT-cod*: *NTCatCod* f = *NTCatCod* (*MakeNT* f)
by (*simp add*: *NTCatCod-def MakeNT-def*)

lemma *MakeNTApp*: $X \in$ *Obj* (*NTCatDom* (*MakeNT* f)) $\Longrightarrow f$ \$\$ X = (*MakeNT* f) \$\$ X
by(*simp add*: *MakeNT-def NTCatDom-def*)

lemma *NatTransMapsTo*:
assumes *NT* η : $F \Longrightarrow G$ **and** $X \in$ *Obj* (*CatDom* F)
shows η \$\$ X *maps*$_{CatCod\ G}$ (F @@ X) *to* (G @@ X)
proof−
have *NTP*: *NatTransP* η **using** *assms* **by** *auto*
have *NTC*: *NTCatCod* η = *CatCod* G **using** *assms* **by** (*auto simp add*: *NTCatCod-def*)

have *NTD*: *NTCatDom* η = *CatDom F* **using** *assms* **by** (*auto simp add*: *NTCatDom-def*)
hence *Obj*: *X* ∈ *Obj* (*NTCatDom* η) **using** *assms* **by** *simp*
have *DF*: *NTDom* η = *F* **and** *CG*: *NTCod* η = *G* **using** *assms* **by** *auto*
have *NTmapsTo*: η $$ *X maps*$_{NTCatCod\ \eta}$ ((*NTDom* η) @@ *X*) *to* ((*NTCod* η) @@ *X*)
using *NTP Obj* **by** (*simp add*: *NatTransP.NatTransMapsTo*)
thus *?thesis* **using** *NTC NTD DF CG* **by** *simp*
qed

definition
NTCompDefined :: ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *NatTrans*
⇒ ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *NatTrans* ⇒ *bool* (**infixl** ‹≈>·› *65*) **where**
NTCompDefined η*1* η*2* ≡ *NatTrans* η*1* ∧ *NatTrans* η*2* ∧ *NTCatDom* η*2* = *NTCatDom* η*1* ∧
NTCatCod η*2* = *NTCatCod* η*1* ∧ *NTCod* η*1* = *NTDom* η*2*

lemma *NTCompDefinedE*[*elim*]: ⟦η*1* ≈>· η*2* ; ⟦*NatTrans* η*1* ; *NatTrans* η*2* ; *NTCatDom* η*2* = *NTCatDom* η*1* ;
NTCatCod η*2* = *NTCatCod* η*1* ; *NTCod* η*1* = *NTDom* η*2*⟧ ⟹ *R*⟧ ⟹ *R*
by (*simp add*: *NTCompDefined-def*)

lemma *NTCompDefinedI*: ⟦*NatTrans* η*1* ; *NatTrans* η*2* ; *NTCatDom* η*2* = *NTCatDom* η*1* ;
NTCatCod η*2* = *NTCatCod* η*1* ; *NTCod* η*1* = *NTDom* η*2*⟧ ⟹ η*1* ≈>· η*2*
by (*simp add*: *NTCompDefined-def*)

lemma *NatTransExt0*:
assumes *NTDom* η*1* = *NTDom* η*2* **and** *NTCod* η*1* = *NTCod* η*2*
and ⋀*X* . *X* ∈ *Obj* (*NTCatDom* η*1*) ⟹ η*1* $$ *X* = η*2* $$ *X*
and *NatTransMap* η*1* ∈ *extensional* (*Obj* (*NTCatDom* η*1*))
and *NatTransMap* η*2* ∈ *extensional* (*Obj* (*NTCatDom* η*2*))
shows η*1* = η*2*
proof−
have *NatTransMap* η*1* = *NatTransMap* η*2*
proof(*rule extensionalityI* [*of NatTransMap* η*1 Obj* (*NTCatDom* η*1*)])
show *NatTransMap* η*1* ∈ *extensional* (*Obj* (*NTCatDom* η*1*)) **using** *assms* **by** *simp*
have *NTCatDom* η*1* = *NTCatDom* η*2* **using** *assms* **by** (*simp add*: *NTCatDom-def*)
moreover have *NatTransMap* η*2* ∈ *extensional* (*Obj* (*NTCatDom* η*2*)) **using** *assms* **by** *simp*
ultimately show *NatTransMap* η*2* ∈ *extensional* (*Obj* (*NTCatDom* η*1*)) **by** *simp*
{fix *X* **assume** *X* ∈ *Obj* (*NTCatDom* η*1*) **thus** η*1* $$ *X* = η*2* $$ *X* **using** *assms* **by** *simp***}**

```
  qed
  thus ?thesis using assms by (simp)
qed

lemma NatTransExt':
  assumes NTDom η1' = NTDom η2' and NTCod η1' = NTCod η2'
  and     ⋀X . X ∈ Obj (NTCatDom η1') ⟹ η1' $$ X = η2' $$ X
  shows   MakeNT η1' = MakeNT η2'
proof(rule NatTransExt0)
  show NatTransMap (MakeNT η1') ∈ extensional (Obj (NTCatDom (MakeNT
η1'))) and
    NatTransMap (MakeNT η2') ∈ extensional (Obj (NTCatDom (MakeNT η2')))
using assms
    by(simp add: MakeNT-def NTCatDom-def NTCatCod-def NatTransExt-def)+
  show NTDom (MakeNT η1') = NTDom (MakeNT η2') and
    NTCod (MakeNT η1') = NTCod (MakeNT η2') using assms by (simp add:
MakeNT-def)+
  {
    fix X assume 1: X ∈ Obj (NTCatDom (MakeNT η1'))
    show (MakeNT η1') $$ X = (MakeNT η2') $$ X
    proof−
      have NTCatDom (MakeNT η1') = NTCatDom (MakeNT η2') using assms
by(simp add: NTCatDom-def MakeNT-def)
      hence 2: X ∈ Obj (NTCatDom (MakeNT η2')) using 1 by simp
        have (NTCatDom η1') = (NTCatDom (MakeNT η1')) by (rule Mak-
eNT-dom)
      hence X ∈ Obj (NTCatDom η1') using 1 assms by simp
      hence η1' $$ X = η2' $$ X using assms by simp
      moreover have η1' $$ X = (MakeNT η1') $$ X using 1 assms by (simp
add: MakeNTApp)
      moreover have η2' $$ X = (MakeNT η2') $$ X using 2 assms by (simp
add: MakeNTApp)
      ultimately have (MakeNT η1') $$ X = (MakeNT η2') $$ X by simp
      thus ?thesis using assms by simp
    qed
  }
qed

lemma NatTransExt:
  assumes NatTrans η1 and NatTrans η2 and NTDom η1 = NTDom η2 and
NTCod η1 = NTCod η2
  and     ⋀X . X ∈ Obj (NTCatDom η1) ⟹ η1 $$ X = η2 $$ X
  shows   η1 = η2
proof−
  have NatTransMap η1 ∈ extensional (Obj (NTCatDom η1)) and
      NatTransMap η2 ∈ extensional (Obj (NTCatDom η2)) using assms
  by(simp only: NatTransExt-def NatTrans-def)+
  thus ?thesis using assms by (simp add: NatTransExt0)
qed
```

definition
IdNatTrans' :: (*'o1*, *'o2*, *'m1*, *'m2*, *'a1*, *'a2*) *Functor* ⇒ (*'o1*, *'o2*, *'m1*, *'m2*, *'a1*, *'a2*) *NatTrans* **where**
IdNatTrans' F ≡ (|
NTDom = *F* ,
NTCod = *F* ,
NatTransMap = λ *X* . $id_{CatCod\ F}$ (*F* @@ *X*)
|)

definition *IdNatTrans F* ≡ *MakeNT*(*IdNatTrans' F*)

lemma *IdNatTrans-map*: *X* ∈ $obj_{CatDom\ F}$ ⟹ (*IdNatTrans F*) $$ *X* = $id_{CatCod\ F}$ (*F* @@ *X*)
by(*auto simp add*: *IdNatTrans-def IdNatTrans'-def MakeNT-comp MakeNT-def NTCatDom-def*)

lemmas *IdNatTrans-defs* = *IdNatTrans-def IdNatTrans'-def MakeNT-def IdNatTrans-map NTCatCod-def NTCatDom-def*

lemma *IdNatTransNatTrans'*: *Functor F* ⟹ *NatTransP*(*IdNatTrans' F*)
proof(*auto simp add*:*NatTransP-def IdNatTrans'-def NTCatDom-def NTCatCod-def Category.Simps*
PreFunctor.FunctorId2 functor-simps Functor.FunctorMapsTo)
{
fix *f X Y*
assume *a*: *Functor F* **and** *b*: *f* $maps_{CatDom\ F}$ *X to Y*
show (*F* ## *f*) ;;$_{CatCod\ F}$ ($id_{CatCod\ F}$ (*F* @@ *Y*)) = ($id_{CatCod\ F}$ (*F* @@ *X*)) ;;$_{CatCod\ F}$ (*F* ## *f*)
proof−
have *1*: *Category* (*CatCod F*) **using** *a* **by** *simp*
have *F* ## *f* $maps_{CatCod\ F}$ (*F* @@ *X*) *to* (*F* @@ *Y*) **using** *a b* **by** (*auto simp add*: *Functor.FunctorMapsTo*)
hence *2*: *F* ## *f* ∈ $mor_{CatCod\ F}$ **and** *3*: $cod_{CatCod\ F}$ (*F* ## *f*) = (*F* @@ *Y*)
and *4*: $dom_{CatCod\ F}$ (*F* ## *f*) = (*F* @@ *X*) **by** *auto*
have (*F* ## *f*) ;;$_{CatCod\ F}$ ($id_{CatCod\ F}$ (*F* @@ *Y*)) = (*F* ## *f*) ;;$_{CatCod\ F}$ ($id_{CatCod\ F}$ ($cod_{CatCod\ F}$ (*F* ## *f*)))
using *3* **by** *simp*
also have ... = *F* ## *f* **using** *1 2* **by** (*auto simp add*: *Category.Cidr*)
also have ... = ($id_{CatCod\ F}$ ($dom_{CatCod\ F}$ (*F* ## *f*))) ;;$_{CatCod\ F}$ (*F* ## *f*)
using *1 2* **by** (*auto simp add*: *Category.Cidl*)
also have ... = ($id_{CatCod\ F}$ (*F* @@ *X*)) ;;$_{CatCod\ F}$ (*F* ## *f*) **using** *4* **by** *simp*
finally show *?thesis* .
qed
}
qed

lemma *IdNatTransNatTrans*: *Functor F* $\Longrightarrow$ *NatTrans* (*IdNatTrans F*)
by (*simp add*: *IdNatTransNatTrans′ IdNatTrans-def MakeNT*)

definition
NatTransComp′ :: (*′o1*, *′o2*, *′m1*, *′m2*, *′a*, *′b*) *NatTrans* $\Rightarrow$
(*′o1*, *′o2*, *′m1*, *′m2*, *′a*, *′b*) *NatTrans* $\Rightarrow$
(*′o1*, *′o2*, *′m1*, *′m2*, *′a*, *′b*) *NatTrans* (**infixl** ‹·1› *75*) **where**
NatTransComp′ η1 η2 = ⦇
NTDom = *NTDom η1* ,
NTCod = *NTCod η2* ,
NatTransMap = λ *X* . (*η1* \$\$ *X*) ;;$_{NTCatCod\ \eta1}$ (*η2* \$\$ *X*)
⦈

definition *NatTransComp* (**infixl** ‹·› *75*) **where** *η1* · *η2* ≡ *MakeNT*(*η1* ·*1* *η2*)

lemma *NatTransComp-Comp1*: ⟦*x* ∈ *Obj* (*NTCatDom f*) ; *f* ≈>· *g*⟧ $\Longrightarrow$ (*f* · *g*) \$\$ *x* = (*f* \$\$ *x*) ;;$_{NTCatCod\ g}$ (*g* \$\$ *x*)
by(*auto simp add*: *NatTransComp-def NatTransComp′-def MakeNT-def NTCatCod-def NTCatDom-def*)

lemma *NatTransComp-Comp2*: ⟦*x* ∈ *Obj* (*NTCatDom f*) ; *f* ≈>· *g*⟧ $\Longrightarrow$ (*f* · *g*) \$\$ *x* = (*f* \$\$ *x*) ;;$_{NTCatCod\ f}$ (*g* \$\$ *x*)
by(*auto simp add*: *NatTransComp-def NatTransComp′-def MakeNT-def NTCatCod-def NTCatDom-def*)

lemmas *NatTransComp-defs* = *NatTransComp-def NatTransComp′-def MakeNT-def*

NatTransComp-Comp1 NTCatCod-def NTCatDom-def

lemma [*simp*]: *η1* ≈>· *η2* $\Longrightarrow$ *NatTrans η1* **by** *auto*
lemma [*simp*]: *η1* ≈>· *η2* $\Longrightarrow$ *NatTrans η2* **by** *auto*
lemma *NTCatDom*: *η1* ≈>· *η2* $\Longrightarrow$ *NTCatDom η1* = *NTCatDom η2* **by** *auto*
lemma *NTCatCod*: *η1* ≈>· *η2* $\Longrightarrow$ *NTCatCod η1* = *NTCatCod η2* **by** *auto*
lemma [*simp*]: *η1* ≈>· *η2* $\Longrightarrow$ *NTCatDom* (*η1* ·*1* *η2*) = *NTCatDom η1* **by** (*auto simp add*: *NatTransComp′-def NTCatDom-def*)
lemma [*simp*]: *η1* ≈>· *η2* $\Longrightarrow$ *NTCatCod* (*η1* ·*1* *η2*) = *NTCatCod η1* **by** (*auto simp add*: *NatTransComp′-def NTCatCod-def*)
lemma [*simp*]: *η1* ≈>· *η2* $\Longrightarrow$ *NTCatDom* (*η1* · *η2*) = *NTCatDom η1* **by** (*auto simp add*: *NatTransComp-defs*)
lemma [*simp*]: *η1* ≈>· *η2* $\Longrightarrow$ *NTCatCod* (*η1* · *η2*) = *NTCatCod η1* **by** (*auto simp add*: *NatTransComp-defs*)
lemma [*simp*]: *NatTrans η* $\Longrightarrow$ *Category*(*NTCatDom η*) **by** (*simp add*: *NatTransP.NatTransFtor NTCatDom-def*)
lemma [*simp*]: *NatTrans η* $\Longrightarrow$ *Category*(*NTCatCod η*) **by** (*simp add*: *NatTransP.NatTransFtor2 NTCatCod-def*)
lemma *DDDC*: **assumes** *NatTrans f* **shows** *CatDom* (*NTDom f*) = *CatDom*

(*NTCod f*)
proof−
have *CatDom* (*NTDom f*) = *NTCatDom f* **by** (*simp add*: *NTCatDom-def*)
thus *?thesis* **using** *assms* **by** (*simp add*: *NatTransP.NatTransFtorDom*)
qed
lemma *CCCD*: **assumes** *NatTrans f* **shows** *CatCod* (*NTCod f*) = *CatCod* (*NTDom f*)
proof−
have *CatCod* (*NTCod f*) = *NTCatCod f* **by** (*simp add*: *NTCatCod-def*)
thus *?thesis* **using** *assms* **by** (*simp add*: *NatTransP.NatTransFtorCod*)
qed

lemma *IdNatTransCompDefDom*: *NatTrans f* ⟹ (*IdNatTrans* (*NTDom f*)) ≈>· *f*
apply(*rule NTCompDefinedI*)
apply(*simp-all add*: *IdNatTransNatTrans NatTransP.NatTransFtor*)
apply(*simp-all add*: *IdNatTrans-defs CCCD*)
done

lemma *IdNatTransCompDefCod*: *NatTrans f* ⟹ *f* ≈>· (*IdNatTrans* (*NTCod f*))
apply(*rule NTCompDefinedI*)
apply(*simp-all add*: *IdNatTransNatTrans NatTransP.NatTransFtor2*)
apply(*simp-all add*: *IdNatTrans-defs DDDC*)
done

lemma *NatTransCompDefCod*:
assumes *NatTrans η* **and** *f maps*$_{NTCatDom\ \eta}$ *X to Y*
shows (*η* $$ *X*) ≈>$_{NTCatCod\ \eta}$ (*NTCod η* ## *f*)
proof(*rule CompDefinedI*)
have *b*: *X* ∈ *obj*$_{NTCatDom\ \eta}$ **and** *c*: *Y* ∈ *obj*$_{NTCatDom\ \eta}$ **using** *assms* **by** (*auto simp add*: *Category.MapsToObj*)
have *d*: (*η* $$ *X*) *maps*$_{NTCatCod\ \eta}$ ((*NTDom η*) @@ *X*) *to* ((*NTCod η*) @@ *X*) **using** *assms b*
by (*simp add*: *NatTransP.NatTransMapsTo*)
thus *η* $$ *X* ∈ *mor*$_{NTCatCod\ \eta}$ **by** *auto*
have *f maps*$_{CatDom\ (NTCod\ \eta)}$ *X to Y* **using** *assms* **by** (*simp add*: *NatTransP.NatTransFtorDom*)
hence *e*: *NTCod η* ## *f maps*$_{CatCod\ (NTCod\ \eta)}$ (*NTCod η* @@ *X*) *to* (*NTCod η* @@ *Y*) **using** *assms*
by (*simp add*: *FunctorM.FunctorCompM NatTransP.NatTransFtor2*)
thus *NTCod η* ## *f* ∈ *mor*$_{NTCatCod\ \eta}$ **by** (*auto simp add*: *NTCatCod-def*)
have *cod*$_{NTCatCod\ \eta}$ (*η* $$ *X*) = (*NTCod η* @@ *X*) **using** *d* **by** *auto*
also have ... = *dom*$_{CatCod\ (NTCod\ \eta)}$ (*NTCod η* ## *f*) **using** *e* **by** *auto*
finally show *cod*$_{NTCatCod\ \eta}$ (*η* $$ *X*) = *dom*$_{NTCatCod\ \eta}$ (*NTCod η* ## *f*) **by** (*auto simp add*: *NTCatCod-def*)
qed

lemma *NatTransCompDefDom*:
assumes *NatTrans η* **and** *f maps*$_{NTCatDom\ \eta}$ *X to Y*

shows $(NTDom\ \eta\ \#\#\ f)\ \approx>_{NTCatCod\ \eta}\ (\eta\ \$\$\ Y)$
proof(*rule CompDefinedI*)
have *b*: $X \in obj_{NTCatDom\ \eta}$ **and** *c*: $Y \in obj_{NTCatDom\ \eta}$ **using** *assms* **by** (*auto simp add*: *Category.MapsToObj*)
have *d*: $(\eta\ \$\$\ Y)\ maps_{NTCatCod\ \eta}\ ((NTDom\ \eta)\ @@\ Y)\ to\ ((NTCod\ \eta)\ @@\ Y)$ **using** *assms c*
by (*simp add*: *NatTransP.NatTransMapsTo*)
thus $\eta\ \$\$\ Y \in mor_{NTCatCod\ \eta}$ **by** *auto*
have $f\ maps_{CatDom\ (NTDom\ \eta)}\ X\ to\ Y$ **using** *assms* **by** (*simp add*: *NTCat-Dom-def*)
hence *e*: $NTDom\ \eta\ \#\#\ f\ maps_{CatCod\ (NTDom\ \eta)}\ (NTDom\ \eta\ @@\ X)\ to\ (NTDom\ \eta\ @@\ Y)$ **using** *assms*
by (*simp add*: *FunctorM.FunctorCompM NatTransP.NatTransFtor*)
thus $NTDom\ \eta\ \#\#\ f \in mor_{NTCatCod\ \eta}$ **using** *assms* **by** (*auto simp add*: *NatTransP.NatTransFtorCod*)
have $dom_{NTCatCod\ \eta}\ (\eta\ \$\$\ Y) = (NTDom\ \eta\ @@\ Y)$ **using** *d* **by** *auto*
also have $... = cod_{CatCod\ (NTDom\ \eta)}\ (NTDom\ \eta\ \#\#\ f)$ **using** *e* **by** *auto*
finally show $cod_{NTCatCod\ \eta}\ (NTDom\ \eta\ \#\#\ f) = dom_{NTCatCod\ \eta}\ (\eta\ \$\$\ Y)$
using *assms* **by** (*auto simp add*: *NatTransP.NatTransFtorCod*)
qed

lemma *NatTransCompCompDef*:
assumes $\eta 1 \approx>\cdot \eta 2$ **and** $X \in obj_{NTCatDom\ \eta 1}$
shows $(\eta 1\ \$\$\ X) \approx>_{NTCatCod\ \eta 1}\ (\eta 2\ \$\$\ X)$
proof(*rule CompDefinedI*)
have *1*: $(\eta 1\ \$\$\ X)\ maps_{NTCatCod\ \eta 1}\ ((NTDom\ \eta 1)\ @@\ X)\ to\ ((NTCod\ \eta 1)\ @@\ X)$ **using** *assms*
by (*simp add*: *NatTransP.NatTransMapsTo*)
have $NTCatCod\ \eta 1 = NTCatCod\ \eta 2$ **using** *assms* **by** *auto*
hence *2*: $(\eta 2\ \$\$\ X)\ maps_{NTCatCod\ \eta 1}\ ((NTDom\ \eta 2)\ @@\ X)\ to\ ((NTCod\ \eta 2)\ @@\ X)$ **using** *assms*
by (*simp add*: *NatTransP.NatTransMapsTo NTCatDom*)
show $\eta 1\ \$\$\ X \in mor_{NTCatCod\ \eta 1}$
and $\eta 2\ \$\$\ X \in mor_{NTCatCod\ \eta 1}$ **using** *1 2* **by** *auto*
have $cod_{NTCatCod\ \eta 1}\ (\eta 1\ \$\$\ X) = ((NTCod\ \eta 1)\ @@\ X)$ **using** *1* **by** *auto*
also have $... = ((NTDom\ \eta 2)\ @@\ X)$ **using** *assms* **by** *auto*
finally show $cod_{NTCatCod\ \eta 1}\ (\eta 1\ \$\$\ X) = dom_{NTCatCod\ \eta 1}\ (\eta 2\ \$\$\ X)$ **using** *2* **by** *auto*
qed

lemma *NatTransCompNatTrans'*:
assumes $\eta 1 \approx>\cdot \eta 2$
shows $NatTransP\ (\eta 1\ \cdot 1\ \eta 2)$
proof(*auto simp add*: *NatTransP-def*)
show $Functor\ (NTDom\ (\eta 1\ \cdot 1\ \eta 2))$ **and** $Functor\ (NTCod\ (\eta 1\ \cdot 1\ \eta 2))$ **using** *assms*
by (*auto simp add*: *NatTransComp'-def NatTransP.NatTransFtor NatTransP.NatTransFtor2*)
show $NTCatDom\ (\eta 1\ \cdot 1\ \eta 2) = CatDom\ (NTCod\ (\eta 1\ \cdot 1\ \eta 2))$ **and**

```
    NTCatCod (η1 ·1 η2) = CatCod (NTDom (η1 ·1 η2))
  proof (auto simp add: NatTransComp'-def NTCatCod-def NTCatDom-def)
   have CatDom (NTDom η1) = NTCatDom η1 by (simp add: NTCatDom-def)
    thus CatDom (NTDom η1) = CatDom (NTCod η2) using assms by (auto
simp add: NatTransP.NatTransFtorDom)
   have CatCod (NTCod η2) = NTCatCod η2 by (simp add: NTCatCod-def)
   thus CatCod (NTCod η2) = CatCod (NTDom η1) using assms by (auto simp
add: NatTransP.NatTransFtorCod)
  qed
  {
   fix X assume aa: X ∈ obj_{NTCatDom (η1 ·1 η2)}
   show (η1 ·1 η2) $$ X maps_{NTCatCod (η1 ·1 η2)} NTDom (η1 ·1 η2) @@ X
to NTCod (η1 ·1 η2) @@ X
   proof−
    have X ∈ obj_{NTCatDom η1} and NatTrans η1 using assms aa by simp+
    hence (η1 $$ X) maps_{NTCatCod η1} ((NTDom η1) @@ X) to ((NTCod η1)
@@ X)
     by (simp add: NatTransP.NatTransMapsTo)
     moreover have (η2 $$ X) maps_{NTCatCod η1} ((NTCod η1) @@ X) to
((NTCod η2) @@ X)
    proof−
     have X ∈ obj_{NTCatDom η2} and NatTrans η2 using assms aa by auto
     hence (η2 $$ X) maps_{NTCatCod η2} ((NTDom η2) @@ X) to ((NTCod η2)
@@ X)
      by (simp add: NatTransP.NatTransMapsTo)
     thus ?thesis using assms by auto
    qed
     ultimately have (η1 $$ X) ;;_{NTCatCod η1} (η2 $$ X) maps_{NTCatCod η1}
((NTDom η1) @@ X) to ((NTCod η2) @@ X)
     using assms by (simp add: Category.Ccompt)
    thus ?thesis using assms by (auto simp add: NatTransComp'-def NTCat-
Cod-def)
   qed
  }
  {
   fix f X Y assume a: f maps_{(NTCatDom (η1 ·1 η2))} X to Y
   show (NTDom (η1 ·1 η2) ## f) ;;_{NTCatCod (η1 ·1 η2)} (η1 ·1 η2 $$ Y) =
     ((η1 ·1 η2) $$ X) ;;_{NTCatCod (η1 ·1 η2)} (NTCod (η1 ·1 η2) ## f)
   proof−
    have b: X ∈ obj_{NTCatDom η1} and c: Y ∈ obj_{NTCatDom η1} using assms a
by (auto simp add: Category.MapsToObj)
    have ((NTDom η1) ## f) ;;_{NTCatCod η1} ((η1 $$ Y) ;;_{NTCatCod η1} (η2 $$
Y)) =
        (((NTDom η1) ## f) ;;_{NTCatCod η1} (η1 $$ Y)) ;;_{NTCatCod η1} (η2 $$
Y)
    proof−
     have ((NTDom η1) ## f) ≈>_{NTCatCod η1} (η1 $$ Y) using assms a by
```

(*auto simp add*: *NatTransCompDefDom*)
moreover have (*η1* $$ *Y*) ≈>$_{NTCatCod\ \eta1}$ (*η2* $$ *Y*) **using** *assms* **by** (*simp add*: *NatTransCompCompDef c*)
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *Category.Cassoc*)
qed
also have ... = ((*η1* $$ *X*) ;;$_{NTCatCod\ \eta1}$ ((*NTDom η2*) ## *f*)) ;;$_{NTCatCod\ \eta1}$ (*η2* $$ *Y*)
using *assms a* **by** (*auto simp add*: *NatTransP.NatTrans*)
also have ... = (*η1* $$ *X*) ;;$_{NTCatCod\ \eta1}$ (((*NTDom η2*) ## *f*) ;;$_{NTCatCod\ \eta1}$ (*η2* $$ *Y*))
proof−
have (*η1* $$ *X*) ≈>$_{NTCatCod\ \eta1}$ ((*NTCod η1*) ## *f*) **using** *assms a* **by** (*simp add*: *NatTransCompDefCod*)
moreover have ((*NTDom η2*) ## *f*) ≈>$_{NTCatCod\ \eta1}$ (*η2* $$ *Y*) **using** *assms a*
by (*simp add*: *NatTransCompDefDom NTCatDom NTCatCod*)
ultimately show *?thesis* **using** *assms* **by** (*auto simp add*: *Category.Cassoc*)
qed
also have ... = (*η1* $$ *X*) ;;$_{NTCatCod\ \eta1}$ ((*η2* $$ *X*) ;;$_{NTCatCod\ \eta1}$ ((*NTCod η2*) ## *f*))
using *assms a* **by** (*simp add*: *NatTransP.NatTrans NTCatDom NTCatCod*)
also have ... = (*η1* $$ *X*) ;;$_{NTCatCod\ \eta1}$ (*η2* $$ *X*) ;;$_{NTCatCod\ \eta1}$ ((*NTCod η2*) ## *f*)
proof−
have (*η1* $$ *X*) ≈>$_{NTCatCod\ \eta1}$ (*η2* $$ *X*) **using** *assms* **by** (*simp add*: *NatTransCompCompDef b*)
moreover have (*η2* $$ *X*) ≈>$_{NTCatCod\ \eta1}$ ((*NTCod η2*) ## *f*) **using** *assms a*
by (*simp add*: *NatTransCompDefCod NTCatCod NTCatDom*)
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *Category.Cassoc*)
qed
finally show *?thesis* **using** *assms* **by** (*auto simp add*: *NatTransComp'-def NTCatCod-def*)
qed
}
qed

lemma *NatTransCompNatTrans*: *η1* ≈>$_{\cdot}$ *η2* ⟹ *NatTrans* (*η1* · *η2*)
by (*simp add*: *NatTransCompNatTrans' NatTransComp-def MakeNT*)

definition
CatExp' :: ('*o1*,'*m1*,'*a*) *Category-scheme* ⇒ ('*o2*,'*m2*,'*b*) *Category-scheme* ⇒
(('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *Functor*,
('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *NatTrans*) *Category* **where**
CatExp' A B ≡ ⦇
Category.Obj = {*F* . *Ftor F* : *A* ⟶ *B*} ,
Category.Mor = {*η* . *NatTrans η* ∧ *NTCatDom η* = *A* ∧ *NTCatCod η* = *B*}
,
Category.Dom = *NTDom* ,

```
    Category.Cod = NTCod ,
    Category.Id  = IdNatTrans ,
    Category.Comp = λf g. (f · g)
  ⟫

definition CatExp A B ≡ MakeCat(CatExp′ A B)

lemma IdNatTransMapL:
  assumes NT: NatTrans f
  shows IdNatTrans (NTDom f) · f = f
proof(rule NatTransExt)
  show NatTrans f using assms .
  show NatTrans (IdNatTrans (NTDom f) · f) using NT
   by (simp add: NatTransP.NatTransFtor IdNatTransNatTrans IdNatTransCompDefDom NatTransCompNatTrans)
  show NTDom (IdNatTrans (NTDom f) · f) = NTDom f and
   NTCod (IdNatTrans (NTDom f) · f) = NTCod f by (simp add: IdNatTrans-defs NatTransComp-defs)+
  {
   fix x assume aa: x ∈ Obj (NTCatDom (IdNatTrans (NTDom f) · f))
   show (IdNatTrans (NTDom f) · f) $$ x = f $$ x
   proof−
    have XObj: x ∈ Obj(NTCatDom f) using aa by (simp add: IdNatTrans-defs NatTransComp-defs)
    have fMap: f $$ x maps_NTCatCod f NTDom f @@ x to NTCod f @@ x using NT XObj
     by (simp add: NatTransP.NatTransMapsTo)
    have (IdNatTrans (NTDom f) · f) $$ x = (IdNatTrans (NTDom f) $$ x) ;;_NTCatCod f (f $$ x)
    proof(rule NatTransComp-Comp1)
     show x ∈ obj_NTCatDom (IdNatTrans (NTDom f)) using XObj by (simp add: IdNatTrans-defs)
      show IdNatTrans (NTDom f) ≈>· f using NT by (simp add: IdNatTransCompDefDom)
    qed
    also have ... = id_NTCatCod f (dom_NTCatCod f (f $$ x)) ;;_NTCatCod f (f $$ x)
     using XObj NT fMap by (auto simp add: IdNatTrans-map NTCatDom-def CCCD NTCatCod-def)
    also have ... = f $$ x
    proof−
     have f $$ x ∈ mor_NTCatCod f using fMap by (auto)
     thus ?thesis using NT by (simp add: Category.Cidl)
    qed
    finally show ?thesis .
   qed
  }
qed
```

lemma *IdNatTransMapR*:
assumes *NT*: *NatTrans f*
shows *f* ⋆ *IdNatTrans* (*NTCod f*) = *f*
proof(*rule NatTransExt*)
show *NatTrans f* **using** *assms* **.**
show *NatTrans* (*f* ⋆ *IdNatTrans* (*NTCod f*)) **using** *NT*
by (*simp add*: *NatTransP.NatTransFtor IdNatTransNatTrans IdNatTransCompDefCod NatTransCompNatTrans*)
show *NTDom* (*f* ⋆ *IdNatTrans* (*NTCod f*)) = *NTDom f* **and**
NTCod (*f* ⋆ *IdNatTrans* (*NTCod f*)) = *NTCod f* **by** (*simp add*: *IdNatTrans-defs NatTransComp-defs*)+
{
fix *x* **assume** *aa*: *x* ∈ *Obj* (*NTCatDom* (*f* ⋆ *IdNatTrans* (*NTCod f*)))
show (*f* ⋆ *IdNatTrans* (*NTCod f*)) $$ *x* = *f* $$ *x*
proof−
have *XObj*: *x* ∈ *Obj*(*NTCatDom f*) **using** *aa* **by** (*simp add*: *NatTransComp-defs*)
have *fMap*: *f* $$ *x maps*$_{NTCatCod\ f}$ *NTDom f* @@ *x to NTCod f* @@ *x* **using** *NT XObj*
by (*simp add*: *NatTransP.NatTransMapsTo*)
have (*f* ⋆ *IdNatTrans* (*NTCod f*)) $$ *x* = (*f* $$ *x*) ;;$_{NTCatCod\ f}$ (*IdNatTrans* (*NTCod f*) $$ *x*)
using *XObj NT* **by** (*auto simp add*: *NatTransComp-Comp2 IdNatTransCompDefCod*)
also have ... = (*f* $$ *x*) ;;$_{NTCatCod\ f}$ (*id*$_{NTCatCod\ f}$ (*cod*$_{NTCatCod\ f}$ (*f* $$ *x*)))
proof−
have *x* ∈ *obj*$_{CatDom\ (NTCod\ f)}$ **using** *XObj NT* **by** (*simp add*: *IdNatTrans-defs DDDC*)
moreover have (*cod*$_{NTCatCod\ f}$ (*f* $$ *x*)) = (*NTCod f*) @@ *x* **using** *fMap* **by** *auto*
ultimately have (*IdNatTrans* (*NTCod f*) $$ *x*) = (*id*$_{NTCatCod\ f}$ (*cod*$_{NTCatCod\ f}$ (*f* $$ *x*)))
by (*simp add*: *IdNatTrans-map NTCatCod-def*)
thus *?thesis* **by** *simp*
qed
also have ... = *f* $$ *x*
proof−
have *f* $$ *x* ∈ *mor*$_{NTCatCod\ f}$ **using** *fMap* **by** (*auto*)
thus *?thesis* **using** *NT* **by** (*simp add*: *Category.Cidr*)
qed
finally show *?thesis* **.**
qed
}
qed

lemma *NatTransCompDefined*:
assumes *f* ≈>⋆ *g* **and** *g* ≈>⋆ *h*
shows (*f* ⋆ *g*) ≈>⋆ *h* **and** *f* ≈>⋆ (*g* ⋆ *h*)
proof−

show $(f \cdot g) \approx\!>\cdot h$
proof(*rule NTCompDefinedI*)
show *NatTrans* $(f \cdot g)$ **and** *NatTrans* h **using** *assms* **by** (*auto simp add*: *NatTransCompNatTrans*)
have $NTCatDom\ f = NTCatDom\ h$ **using** *assms* **by** (*simp add*: *NTCatDom*)
thus $NTCatDom\ h = NTCatDom\ (f \cdot g)$ **by** (*simp add*: *NatTransComp-defs*)
have $NTCatCod\ h = NTCatCod\ g$ **using** *assms* **by** (*simp add*: *NTCatCod*)
thus $NTCatCod\ h = NTCatCod\ (f \cdot g)$ **by** (*simp add*: *NatTransComp-defs*)
show $NTCod\ (f \cdot g) = NTDom\ h$ **using** *assms* **by** (*auto simp add*: *NatTransComp-defs*)
qed
show $f \approx\!>\cdot (g \cdot h)$
proof(*rule NTCompDefinedI*)
show *NatTrans* f **and** *NatTrans* $(g \cdot h)$ **using** *assms* **by** (*auto simp add*: *NatTransCompNatTrans*)
have $NTCatDom\ f = NTCatDom\ g$ **using** *assms* **by** (*simp add*: *NTCatDom*)
thus $NTCatDom\ (g \cdot h) = NTCatDom\ f$ **by** (*simp add*: *NatTransComp-defs*)
have $NTCatCod\ h = NTCatCod\ f$ **using** *assms* **by** (*simp add*: *NTCatCod*)
thus $NTCatCod\ (g \cdot h) = NTCatCod\ f$ **by** (*simp add*: *NatTransComp-defs*)
show $NTCod\ f = NTDom\ (g \cdot h)$ **using** *assms* **by** (*auto simp add*: *NatTransComp-defs*)
qed
qed

lemma *NatTransCompAssoc*:
assumes $f \approx\!>\cdot g$ **and** $g \approx\!>\cdot h$
shows $(f \cdot g) \cdot h = f \cdot (g \cdot h)$
proof(*rule NatTransExt*)
show *NatTrans* $((f \cdot g) \cdot h)$ **using** *assms* **by** (*simp add*: *NatTransCompNatTrans NatTransCompDefined*)
show *NatTrans* $(f \cdot (g \cdot h))$ **using** *assms* **by** (*simp add*: *NatTransCompNatTrans NatTransCompDefined*)
show $NTDom\ (f \cdot g \cdot h) = NTDom\ (f \cdot (g \cdot h))$ **and** $NTCod\ (f \cdot g \cdot h) = NTCod\ (f \cdot (g \cdot h))$
by(*simp add*: *NatTransComp-defs*)+
{
fix x **assume** *aa*: $x \in obj_{NTCatDom\ (f \cdot g \cdot h)}$ **show** $((f \cdot g) \cdot h)\ \$\$\ x = (f \cdot (g \cdot h))\ \$\$\ x$
proof−
have *ntd1*: $NTCatDom\ (f \cdot g) = NTCatDom\ (f \cdot g \cdot h)$ **and** *ntd2*: $NTCatDom\ f = NTCatDom\ (f \cdot g \cdot h)$
using *assms* **by** (*simp add*: *NatTransCompDefined*)+
have *obj1*: $x \in Obj\ (NTCatDom\ f)$ **using** *aa ntd2* **by** *simp*
have *1*: $(f \cdot g)\ \$\$\ x = (f\ \$\$\ x)\ ;;_{NTCatCod\ h}\ (g\ \$\$\ x)$ **and**
2: $(g \cdot h)\ \$\$\ x = (g\ \$\$\ x)\ ;;_{NTCatCod\ h}\ (h\ \$\$\ x)$ **using** *obj1*
using *assms* **by** (*auto simp add*: *NatTransComp-Comp1*)
have $((f \cdot g) \cdot h)\ \$\$\ x = ((f \cdot g)\ \$\$\ x)\ ;;_{NTCatCod\ h}\ (h\ \$\$\ x)$
proof(*rule NatTransComp-Comp1*)
show $x \in obj_{NTCatDom\ (f \cdot g)}$ **using** *aa ntd1* **by** *simp*

```
      show f · g ≈>· h using assms by (simp add: NatTransCompDefined)
     qed
    also have ... = ((f $$ x) ;;_NTCatCod h (g $$ x)) ;;_NTCatCod h (h $$ x) using
1 by simp
     also have ... = (f $$ x) ;;_NTCatCod h ((g $$ x) ;;_NTCatCod h (h $$ x))
     proof−
      have 1: NTCatCod h = NTCatCod f and 2: NTCatCod h = NTCatCod g
using assms by (simp add: NTCatCod)+
       hence (f $$ x) ≈>_NTCatCod h (g $$ x) using obj1 assms by (simp add:
NatTransCompCompDef)
       moreover have (g $$ x) ≈>_NTCatCod h (h $$ x) using obj1 assms 2 by
(simp add: NatTransCompCompDef NTCatDom)
       moreover have Category (NTCatCod h) using assms by auto
       ultimately show ?thesis by (simp add: Category.Cassoc)
     qed
     also have ... = (f $$ x) ;;_NTCatCod h ((g · h) $$ x) using 2 by simp
     also have ... = (f · (g · h)) $$ x
     proof−
      have NTCatCod f = NTCatCod h using assms by (simp add: NTCatCod)
      moreover have (f · (g · h)) $$ x = (f $$ x) ;;_NTCatCod f ((g · h) $$ x)
      proof(rule NatTransComp-Comp2)
       show x ∈ obj_NTCatDom f using obj1 assms by (simp add: NTCatDom)
       show f ≈>· g · h using assms by (simp add: NatTransCompDefined)
      qed
      ultimately show ?thesis by simp
     qed
     finally show ?thesis .
    qed
  }
qed

lemma CatExpCatAx:
  assumes Category A and Category B
  shows Category-axioms (CatExp' A B)
proof(auto simp add: Category-axioms-def)
  {
   fix f assume f ∈ mor_CatExp' A B
   thus dom_CatExp' A B f ∈ obj_CatExp' A B and
        cod_CatExp' A B f ∈ obj_CatExp' A B
    by(auto simp add: CatExp'-def NatTransP.NatTransFtor
    NatTransP.NatTransFtor2 NatTransP.NatTransFtorDom NatTransP.NatTransFtorCod
DDDC CCCD functor-abbrev-def)
  }
  {
   fix F assume a: F ∈ obj_CatExp' A B
   show id_CatExp' A B F maps_CatExp' A B F to F
   proof(rule MapsToI)
    have Ftor F : A ⟶ B using a by (simp add: CatExp'-def)
```

thus $id_{CatExp'\ A\ B}\ F \in mor_{CatExp'\ A\ B}$
apply(*simp add*: *CatExp'-def NTCatDom-def NTCatCod-def IdNatTransNatTrans functor-abbrev-def*)
apply(*simp add*: *IdNatTrans-defs*)
done
show $dom_{CatExp'\ A\ B}\ (id_{CatExp'\ A\ B}\ F) = F$ **by** (*simp add*: *CatExp'-def IdNatTrans-defs*)
show $cod_{CatExp'\ A\ B}\ (id_{CatExp'\ A\ B}\ F) = F$ **by** (*simp add*: *CatExp'-def IdNatTrans-defs*)
qed
}
{
fix *f* **assume** *a*: $f \in mor_{CatExp'\ A\ B}$
show $(id_{CatExp'\ A\ B}\ (dom_{CatExp'\ A\ B}\ f)) ;;_{CatExp'\ A\ B}\ f = f$ **and**
$f ;;_{CatExp'\ A\ B}\ (id_{CatExp'\ A\ B}\ (cod_{CatExp'\ A\ B}\ f)) = f$
proof(*simp-all add*: *CatExp'-def*)
have *NT*: *NatTrans f* **using** *a* **by** (*simp add*: *CatExp'-def*)
show *IdNatTrans* (*NTDom f*) $\cdot$ *f* = *f* **using** *NT* **by** (*simp add*:*IdNatTransMapL*)
show *f* $\cdot$ *IdNatTrans* (*NTCod f*) = *f* **using** *NT* **by** (*simp add*:*IdNatTransMapR*)
qed
}
{
fix *f g h* **assume** *aa*: $f \approx>_{CatExp'\ A\ B}\ g$ **and** *bb*: $g \approx>_{CatExp'\ A\ B}\ h$
{
fix *f g* **assume** $f \approx>_{CatExp'\ A\ B}\ g$ **hence** $f \approx>\cdot\ g$
apply(*simp only*: *NTCompDefined-def*)
by (*auto simp add*: *CatExp'-def*)
}
hence $f \approx>\cdot\ g$ **and** $g \approx>\cdot\ h$ **using** *aa bb* **by** *auto*
thus $f ;;_{CatExp'\ A\ B}\ g ;;_{CatExp'\ A\ B}\ h = f ;;_{CatExp'\ A\ B}\ (g ;;_{CatExp'\ A\ B}\ h)$
by(*simp add*: *CatExp'-def NatTransCompAssoc*)
}
{
fix *f g X Y Z* **assume** *a*: $f\ maps_{CatExp'\ A\ B}\ X\ to\ Y$ **and** *b*: $g\ maps_{CatExp'\ A\ B}\ Y\ to\ Z$
show $f ;;_{CatExp'\ A\ B}\ g\ maps_{CatExp'\ A\ B}\ X\ to\ Z$
proof(*rule MapsToI*, *auto simp add*: *CatExp'-def*)
have *nt1*: *NatTrans f* **and** *cd1*: *NTCatDom f* = *A*
and *cc1*: *NTCatCod f* = *B* **and** *d1*: *NTDom f* = *X* **and** *c1*: *NTCod f* = *Y*
using *a* **by** (*auto simp add*: *CatExp'-def*)
moreover have *nt2*: *NatTrans g* **and** *cd2*: *NTCatDom g* = *A*
and *cc2*: *NTCatCod g* = *B* **and** *d2*: *NTDom g* = *Y* **and** *c2*: *NTCod g* = *Z*
using *b* **by** (*auto simp add*: *CatExp'-def*)
ultimately have *Comp*: $f \approx>\cdot\ g$ **by**(*auto intro*: *NTCompDefinedI*)
thus *NatTrans* (*f* $\cdot$ *g*) **by** (*simp add*: *NatTransCompNatTrans*)
show *NTCatDom* (*f* $\cdot$ *g*) = *A* **using** *Comp cd1* **by** (*simp add*: *NTCatDom*)
show *NTCatCod* (*f* $\cdot$ *g*) = *B* **using** *Comp cc2* **by** (*simp add*: *NTCatCod*)

show *NTDom* $(f \cdot g) = X$ **using** *d1* **by** (*simp add*: *NatTransComp-defs*)
show *NTCod* $(f \cdot g) = Z$ **using** *c2* **by** (*simp add*: *NatTransComp-defs*)
qed
}
qed

lemma *CatExpCat*: $[\![Category\ A\ ;\ Category\ B]\!] \Longrightarrow Category\ (CatExp\ A\ B)$
by(*simp add*: *CatExpCatAx CatExp-def MakeCat*)

lemmas *CatExp-defs* = *CatExp-def CatExp'-def MakeCat-def*

lemma *CatExpDom*: $f \in Mor\ (CatExp\ A\ B) \Longrightarrow dom_{CatExp\ A\ B}\ f = NTDom\ f$
by (*simp add*: *CatExp-defs*)

lemma *CatExpCod*: $f \in Mor\ (CatExp\ A\ B) \Longrightarrow cod_{CatExp\ A\ B}\ f = NTCod\ f$
by (*simp add*: *CatExp-defs*)

lemma *CatExpId*: $X \in Obj\ (CatExp\ A\ B) \Longrightarrow Id\ (CatExp\ A\ B)\ X = IdNatTrans\ X$
by (*simp add*: *CatExp-defs*)

lemma *CatExpNatTransCompDef*: **assumes** $f \approx>_{CatExp\ A\ B} g$ **shows** $f \approx>\cdot g$
proof−
have *1*: $f \approx>_{CatExp'\ A\ B} g$ **using** *assms* **by** (*simp add*: *CatExp-def MakeCat-CompDef*)
show $f \approx>\cdot g$
proof(*rule NTCompDefinedI*)
show *NatTrans f* **using** *1* **by** (*auto simp add*: *CatExp'-def*)
show *NatTrans g* **using** *1* **by** (*auto simp add*: *CatExp'-def*)
show *NTCatDom g* = *NTCatDom f* **using** *1* **by** (*auto simp add*: *CatExp'-def*)
show *NTCatCod g* = *NTCatCod f* **using** *1* **by** (*auto simp add*: *CatExp'-def*)
show *NTCod f* = *NTDom g* **using** *1* **by** (*auto simp add*: *CatExp'-def*)
qed
qed

lemma *CatExpDist*:
assumes $X \in Obj\ A$ **and** $f \approx>_{CatExp\ A\ B} g$
shows $(f\ ;;_{CatExp\ A\ B}\ g)\ \$\$\ X = (f\ \$\$\ X)\ ;;_B\ (g\ \$\$\ X)$
proof−
have $f \in Mor\ (CatExp'\ A\ B)$ **using** *assms* **by** (*auto simp add*: *CatExp-def MakeCatMor*)
hence *1*: *NTCatDom f* = *A* **and** *2*: *NTCatCod f* = *B* **by** (*simp add*: *CatExp'-def*)+
hence *4*: $X \in Obj\ (NTCatDom\ f)$ **using** *assms* **by** *simp*
have *3*: $f \approx>\cdot g$ **using** *assms*(*2*) **by** (*simp add*: *CatExpNatTransCompDef*)
have $(f\ ;;_{CatExp\ A\ B}\ g)\ \$\$\ X = (f\ ;;_{CatExp'\ A\ B}\ g)\ \$\$\ X$ **using** *assms*(*2*) **by** (*simp add*: *CatExp-def MakeCatComp2*)
also have ... $= (f \cdot g)\ \$\$\ X$ **by** (*simp add*: *CatExp'-def*)
also have ... $= (f\ \$\$\ X)\ ;;_B\ (g\ \$\$\ X)$ **using** *4 2 3* **by** (*simp add*: *Nat-*

TransComp-Comp2[*of X f g*])
finally show *?thesis* **.**
qed

lemma *CatExpMorNT*: $f \in Mor\ (CatExp\ A\ B) \Longrightarrow NatTrans\ f$
by (*simp add*: *CatExp-defs*)

end

6 The Category of Sets

theory *SetCat*
imports *Functors Universe*
begin

notation *Elem* (**infixl** ‹|∈|› *70*)
notation *HOLZF.subset* (**infixl** ‹|⊆|› *71*)
notation *CartProd* (**infixl** ‹|×|› *75*)

definition
ZFfun :: *ZF* ⇒ *ZF* ⇒ (*ZF* ⇒ *ZF*) ⇒ *ZF* **where**
ZFfun d r f ≡ *Opair* (*Opair d r*) (*Lambda d f*)

definition
ZFfunDom :: *ZF* ⇒ *ZF* (‹|*dom*|-› [*72*] *72*) **where**
ZFfunDom f ≡ *Fst* (*Fst f*)

definition
ZFfunCod :: *ZF* ⇒ *ZF* (‹|*cod*|-› [*72*] *72*) **where**
ZFfunCod f ≡ *Snd* (*Fst f*)

definition
ZFfunApp :: *ZF* ⇒ *ZF* ⇒ *ZF* (**infixl** ‹|@|› *73*) **where**
ZFfunApp f x ≡ *app* (*Snd f*) *x*

definition
ZFfunComp :: *ZF* ⇒ *ZF* ⇒ *ZF* (**infixl** ‹|*o*|› *72*) **where**
ZFfunComp f g ≡ *ZFfun* (|*dom*| *f*) (|*cod*| *g*) (λ*x*. *g* |@| (*f* |@| *x*))

definition
isZFfun :: *ZF* ⇒ *bool* **where**
isZFfun drf ≡ *let f* = *Snd drf in*
isOpair drf ∧ *isOpair* (*Fst drf*) ∧ *isFun f* ∧ (*f* |⊆| (*Domain f*) |×| (*Range f*))
∧ (*Domain f* = |*dom*| *drf*) ∧ (*Range f* |⊆| |*cod*| *drf*)

lemma *isZFfunE*[*elim*]: ⟦*isZFfun f* ;
⟦*isOpair f* ; *isOpair* (*Fst f*) ; *isFun* (*Snd f*) ;
((*Snd f*) |⊆| (*Domain* (*Snd f*)) |×| (*Range* (*Snd f*))) ;

$(Domain\ (Snd\ f) = |dom|\ f) \wedge (Range\ (Snd\ f)\ |\subseteq|\ |cod|\ f)\rrbracket \Longrightarrow R\rrbracket \Longrightarrow R$
by (*auto simp add*: *isZFfun-def Let-def*)

definition
$SET' :: (ZF, ZF)\ Category$ **where**
$SET' \equiv (\!|$
$Category.Obj = \{x\ .\ True\}$,
$Category.Mor = \{f\ .\ isZFfun\ f\}$,
$Category.Dom = ZFfunDom$,
$Category.Cod = ZFfunCod$,
$Category.Id = \lambda x.\ ZFfun\ x\ x\ (\lambda x\ .\ x)$,
$Category.Comp = ZFfunComp$
$|\!)$

definition $SET \equiv MakeCat\ SET'$

lemma *ZFfunDom*: $|dom|\ (ZFfun\ A\ B\ f) = A$
by (*auto simp add*: *ZFfun-def ZFfunDom-def Fst*)

lemma *ZFfunCod*: $|cod|\ (ZFfun\ A\ B\ f) = B$
by (*auto simp add*: *ZFfun-def ZFfunCod-def Snd Fst*)

lemma *SETfun*:
assumes $\forall\ x\ .\ x\ |\in|\ A \longrightarrow (f\ x)\ |\in|\ B$
shows $isZFfun\ (ZFfun\ A\ B\ f)$
proof(*auto simp add*: *isZFfun-def ZFfun-def isOpair Fst Snd*
ZFfunCod-def ZFfunDom-def isFun-Lambda domain-Lambda Let-def)
{
fix x
have $x\ |\in|\ Range\ (Lambda\ A\ f) \Longrightarrow x\ |\in|\ B$
apply(*insert isFun-Lambda*[*of A f*])
apply (*drule fun-range-witness*[*of Lambda A f x*], *simp*)
by (*auto simp add*: *domain-Lambda Lambda-app assms*)
}
thus *subset* $(Range\ (Lambda\ A\ f))\ B$
by (*auto simp add*: *subset-def*)
{
fix x
have $x\ |\in|\ (Lambda\ A\ f) \Longrightarrow x\ |\in|\ A\ |\times|\ Range\ (Lambda\ A\ f)$
by(*auto simp add*: *CartProd Lambda-def Repl Range*)
}
thus $(Lambda\ A\ f)\ |\subseteq|\ (A\ |\times|\ Range\ (Lambda\ A\ f))$
by (*auto simp add*: *HOLZF.subset-def*)
qed

lemma *ZFCartProd*:
assumes $x\ |\in|\ A\ |\times|\ B$
shows $Fst\ x\ |\in|\ A \wedge Snd\ x\ |\in|\ B \wedge isOpair\ x$
proof−

```
  from CartProd obtain a b
    where a |∈| A
    and   b |∈| B
    and   x = Opair a b using assms by auto
  thus ?thesis using assms by (auto simp add: Fst Snd isOpair-def)
qed

lemma ZFfunDomainOpair:
  assumes isFun f
  and     x |∈| Domain f
  shows   Opair x (app f x) |∈| f
proof−
  have ∃! y . Opair x y |∈| f using assms by (auto simp add: unique-fun-value)
  thus Opair x (app f x) |∈| f by (auto simp add: app-def intro: theI')
qed

lemma ZFFunToLambda:
  assumes 1: isFun f
  and     2: f |⊆| (Domain f) |×| (Range f)
  shows   f = Lambda (Domain f) (λx. app f x)
proof(subst Ext, rule allI, rule iffI)
  {
    fix x assume a: x |∈| f show x |∈| Lambda (Domain f) (λx. app f x)
    proof(simp add: Lambda-def Repl, rule exI[of - (Fst x)], rule conjI)
     have b:isOpair x ∧ Fst x |∈| Domain f using 2 a by (auto simp add: subset-def
ZFCartProd)
      thus Fst x |∈| Domain f ..
      hence Opair (Fst x) (app f (Fst x)) |∈| f using 1 by (simp add: ZFfunDo-
mainOpair)
      moreover have Opair (Fst x) (Snd x) |∈| f using a 2 by (auto simp add:
FstSnd subset-def b)
       ultimately have Snd x = (app f (Fst x)) using 1 by (auto simp add:
isFun-def)
      hence Opair (Fst x) (app f (Fst x)) = Opair (Fst x) (Snd x) by simp
      also have ... = x  using b by (simp add: FstSnd)
      finally show x = Opair (Fst x) (app f (Fst x)) ..
    qed
  }
  moreover
  {
    fix x assume a: x |∈| Lambda (Domain f) (λx. app f x) show x |∈| f
      proof−
        from Lambda-def obtain a where a |∈| Domain f ∧ x = Opair a (app f
a)
          using a by (auto simp add: Repl)
        thus ?thesis using a 1 by (auto simp add: ZFfunDomainOpair)
      qed
  }
qed
```

lemma *ZFfunApp*:
assumes $x \mid\in\mid A$
shows $(ZFfun\ A\ B\ f) \mid@\mid x = f\ x$
proof−
have $(ZFfun\ A\ B\ f) \mid@\mid x = app\ (Lambda\ A\ f)\ x$ **by** (*simp add*: *ZFfun-def ZFfunApp-def Snd*)
also have $... = f\ x$ **using** *assms* **by** (*simp add*: *Lambda-app*)
finally show *?thesis* **.**
qed

lemma *ZFfun*:
assumes *isZFfun f*
shows $f = ZFfun\ (\mid dom\mid f)\ (\mid cod\mid f)\ (\lambda x.\ f \mid@\mid x)$
proof(*auto simp add*: *ZFfun-def*)
have $isOpair\ f \wedge isOpair\ (Fst\ f)$ **using** *assms* **by** (*simp add*: *isZFfun-def*[*of f*] *Let-def*)
hence $f = Opair\ (Opair\ (Fst\ (Fst\ f))\ (Snd\ (Fst\ f)))\ (Snd\ f)$ **by** (*simp add*: *FstSnd*)
hence $f = Opair\ (Opair\ (\mid dom\mid f)\ (\mid cod\mid f))\ (Snd\ f)$ **using** *assms* **by** (*simp add*: *ZFfunDom-def ZFfunCod-def*)
moreover have $Snd\ f = Lambda\ (\mid dom\mid f)\ (\lambda x\ .\ f \mid@\mid x)$
proof−
have $\mid dom\mid f = Domain\ (Snd\ f)$ **using** *assms* **by** (*simp add*: *isZFfun-def*[*of f*] *Let-def*)
moreover have $isFun\ (Snd\ f)$ **using** *assms* **by** (*simp add*: *isZFfun-def*[*of f*] *Let-def*)
moreover have $(\lambda x\ .\ f \mid@\mid x) = (\lambda x\ .\ app\ (Snd\ f)\ x)$ **by**(*simp add*: *ZFfunApp-def*)
moreover have $(Snd\ f) \mid\subseteq\mid (Domain\ (Snd\ f)) \mid\times\mid (Range\ (Snd\ f))$ **using** *assms*
by (*auto simp add*: *isZFfun-def*[*of f*] *Let-def*)
ultimately show *?thesis* **apply** *simp* **by**(*rule ZFFunToLambda*[*of Snd f*])
qed
ultimately show $f = Opair\ (Opair\ (\mid dom\mid f)\ (\mid cod\mid f))\ (Lambda\ (\mid dom\mid f)\ (\lambda x\ .\ f \mid@\mid x))$ **by** *simp*
qed

lemma *ZFfun-ext*:
assumes $\forall\ x\ .\ x \mid\in\mid A \longrightarrow f\ x = g\ x$
shows $(ZFfun\ A\ B\ f) = (ZFfun\ A\ B\ g)$
proof−
have $Lambda\ A\ f = Lambda\ A\ g$ **using** *assms* **by** (*auto simp add*: *Lambda-ext*)
thus *?thesis* **by** (*simp add*: *ZFfun-def*)
qed

lemma *ZFfunExt*:
assumes $\mid dom\mid f = \mid dom\mid g$ **and** $\mid cod\mid f = \mid cod\mid g$ **and** *funf*: *isZFfun f* **and** *fung*: *isZFfun g*

and $\bigwedge x\ .\ x\ |\in|\ (\ |dom|\ f) \Longrightarrow f\ |@|\ x = g\ |@|\ x$
shows $f = g$
proof−
have *1*: $f = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda x.\ f\ |@|\ x)$ **using** *funf* **by** (*rule ZFfun*)
have $g = ZFfun\ (\ |dom|\ g)\ (\ |cod|\ g)\ (\lambda x.\ g\ |@|\ x)$ **using** *fung* **by** (*rule ZFfun*)
hence *2*: $g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda x.\ g\ |@|\ x)$ **using** *assms* **by** *simp*
have $ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda x.\ f\ |@|\ x) = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda x.\ g\ |@|\ x)$
using *assms* **by** (*simp add*: *ZFfun-ext*)
thus *?thesis* **using** *1 2* **by** *simp*
qed

lemma *ZFfunDomAppCod*:
assumes *isZFfun f*
and $x\ |\in|\ |dom|f$
shows $f\ |@|\ x\ |\in|\ |cod|f$
proof(*simp add*: *ZFfunApp-def*)
have *app* (*Snd f*) $x\ |\in|$ *Range* (*Snd f*) **using** *assms* **by** (*auto simp add*: *fun-value-in-range*)
thus *app* (*Snd f*) $x\ |\in|\ |cod|f$ **using** *assms* **by** (*auto simp add*: *HOLZF.subset-def*)
qed

lemma *ZFfunComp*:
assumes $\forall\ x\ .\ x\ |\in|\ A \longrightarrow f\ x\ |\in|\ B$
shows $(ZFfun\ A\ B\ f)\ |o|\ (ZFfun\ B\ C\ g) = ZFfun\ A\ C\ (g\ o\ f)$
proof (*simp add*: *ZFfunComp-def ZFfunDom ZFfunCod*)
{
fix x **assume** *a*: $x\ |\in|\ A$
have $ZFfun\ B\ C\ g\ \ |@|\ (ZFfun\ A\ B\ f\ |@|\ x) = (g\ o\ f)\ x$
proof−
have $(ZFfun\ A\ B\ f\ |@|\ x) = f\ x$ **using** *a* **by** (*simp add*: *ZFfunApp*)
hence $ZFfun\ B\ C\ g\ \ |@|\ (ZFfun\ A\ B\ f\ |@|\ x) = g\ (f\ x)$ **using** *assms a* **by** (*simp add*: *ZFfunApp*)
thus *?thesis* **by** *simp*
qed
}
thus $ZFfun\ A\ C\ (\lambda x.\ ZFfun\ B\ C\ g\ |@|\ (ZFfun\ A\ B\ f\ |@|\ x)) = ZFfun\ A\ C\ (g \circ f)$
by (*simp add*: *ZFfun-ext*)
qed

lemma *ZFfunCompApp*:
assumes *a*:*isZFfun f* **and** *b*:*isZFfun g* **and** *c*:$|dom|g = |cod|f$
shows $f\ |o|\ g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))$
proof−
have *1*: $f = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x)$ **using** *a* **by** (*rule ZFfun*)
have *2*: $g = ZFfun\ (\ |dom|\ g)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ x)$ **using** *b* **by** (*rule ZFfun*)
have *3*: $\forall\ x\ .\ x\ |\in|\ |dom|f \longrightarrow (\lambda x.\ f\ |@|\ x)\ x\ |\in|\ |cod|f$ **using** *a* **by** (*simp add*: *ZFfunDomAppCod*)

hence *4*: $\forall\ x\ .\ x\ |\in|\ |dom|f \longrightarrow (\lambda x.\ g\ |@|\ (f\ |@|\ x))\ x\ |\in|\ |cod|g$
using *a b c* **by** (*simp add*: *ZFfunDomAppCod*)
have $f\ |o|\ g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x)\ |o|$
ZFfun $(\ |cod|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ x)$ **using** *1 2 c* **by** *simp*
hence $f\ |o|\ g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))$
using *3* **by** (*simp add*: *ZFfunComp comp-def*)
thus *?thesis* **using** *4* **by** (*simp add*: *SETfun*)
qed

lemma *ZFfunCompAppZFfun*:
assumes *isZFfun f* **and** *isZFfun g* **and** $|dom|g = |cod|f$
shows *isZFfun* $(f\ |o|\ g)$
proof−
have $f\ |o|\ g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))$ **using** *assms*
by (*simp add*: *ZFfunCompApp*)
moreover have $\forall\ x\ .\ x\ |\in|\ |dom|f \longrightarrow ((\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))\ x)\ |\in|\ |cod|g$
using *assms*
by (*simp add*: *ZFfunDomAppCod*)
ultimately show *?thesis* **by** (*simp add*: *SETfun*)
qed

lemma *ZFfunCompAssoc*:
assumes *a*: *isZFfun f* **and** *b*:*isZFfun h* **and** *c*:$|cod|g = |dom|h$
and *d*:*isZFfun g* **and** *e*:$|cod|f = |dom|g$
shows $f\ |o|\ g\ |o|\ h = f\ |o|\ (g\ |o|\ h)$
proof−
have *1*: $f = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x)$ **using** *a* **by** (*rule ZFfun*)
have *2*: $g = ZFfun\ (\ |dom|\ g)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ x)$ **using** *d* **by** (*rule ZFfun*)
have *3*: $h = ZFfun\ (\ |dom|\ h)\ (\ |cod|\ h)\ (\lambda\ x\ .\ h\ |@|\ x)$ **using** *b* **by** (*rule ZFfun*)
have *4*: $\forall\ x\ .\ x\ |\in|\ |dom|f \longrightarrow (\lambda x.\ f\ |@|\ x)\ x\ |\in|\ |cod|f$ **using** *a* **by** (*simp add*: *ZFfunDomAppCod*)
have $(f\ |o|\ g)\ |o|\ h = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ h)\ (\lambda\ x\ .\ h\ |@|\ (g\ |@|\ (f\ |@|\ x)))$
proof−
have *5*: $\forall\ x\ .\ x\ |\in|\ |dom|f \longrightarrow (\lambda x.\ g\ |@|\ (f\ |@|\ x))\ x\ |\in|\ |cod|g$
using *4 e d* **by** (*simp add*: *ZFfunDomAppCod*)
have $(f\ |o|\ g)\ |o|\ h = (ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x)\ |o|$
ZFfun $(\ |cod|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ x))\ |o|$
ZFfun $(\ |cod|\ g)\ (\ |cod|\ h)\ (\lambda\ x\ .\ h\ |@|\ x)$
using *1 2 3 c e* **by** (*simp*)
thus *?thesis* **using** *4 5* **by** (*simp add*: *ZFfunComp comp-def*)
qed
moreover have $f\ |o|\ (g\ |o|\ h) = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ h)\ (\lambda\ x\ .\ h\ |@|\ (g\ |@|\ (f\ |@|\ x)))$
proof−
have *5*: $\forall\ x\ .\ x\ |\in|\ |dom|g \longrightarrow (\lambda x.\ g\ |@|\ x)\ x\ |\in|\ |cod|g$ **using** *d* **by** (*simp add*: *ZFfunDomAppCod*)
have $f\ |o|\ (g\ |o|\ h) = ZFfun\ (\ |dom|\ f)\ (\ |dom|\ g)\ (\lambda\ x\ .\ f\ |@|\ x)\ |o|$
$(ZFfun\ (\ |dom|\ g)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ x)\ |o|$
ZFfun $(\ |cod|\ g)\ (\ |cod|\ h)\ (\lambda\ x\ .\ h\ |@|\ x))$

using *1 2 3 c e* **by** (*simp*)
thus *?thesis* **using** *4 e 5* **by** (*simp add*: *ZFfunComp comp-def*)
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *ZFfunCompAppDomCod*:
assumes *isZFfun f* **and** *isZFfun g* **and** $|dom|g = |cod|f$
shows $|dom|\ (f\ |o|\ g) = |dom|\ f \wedge |cod|\ (f\ |o|\ g) = |cod|\ g$
proof−
have $f\ |o|\ g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))$ **using** *assms*
by (*simp add*: *ZFfunCompApp*)
thus *?thesis* **by** (*simp add*: *ZFfunDom ZFfunCod*)
qed

lemma *ZFfunIdLeft*:
assumes *a*: *isZFfun f* **shows** $(ZFfun\ (\ |dom|f)\ (\ |dom|f)\ (\lambda x.\ x))\ |o|\ f = f$
proof−
let $?g = (ZFfun\ (\ |dom|f)\ (\ |dom|f)\ (\lambda x.\ x))$
have $ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x) = ?g\ |o|\ f$ **using** *a*
by (*simp add*: *ZFfun-ext ZFfunApp ZFfunCompApp SETfun ZFfunCod ZFfunDom*)
moreover have $f = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x)$ **using** *a* **by** (*rule ZFfun*)
ultimately show *?thesis* **by** *simp*
qed

lemma *ZFfunIdRight*:
assumes *a*: *isZFfun f* **shows** $f\ |o|\ (ZFfun\ (\ |cod|f)\ (\ |cod|f)\ (\lambda x.\ x)) = f$
proof−
let $?g = (ZFfun\ (\ |cod|f)\ (\ |cod|f)\ (\lambda x.\ x))$
have *1*: $\forall\ x\ .\ x\ |\in|\ |dom|f \longrightarrow (\lambda x.\ f\ |@|\ x)\ x\ |\in|\ |cod|f$ **using** *a* **by** (*simp add*: *ZFfunDomAppCod*)
have $ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x) = f\ |o|\ ?g$ **using** *a 1*
by (*simp add*: *ZFfun-ext ZFfunApp ZFfunCompApp SETfun ZFfunCod ZFfunDom*)
moreover have $f = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda\ x\ .\ f\ |@|\ x)$ **using** *a* **by** (*rule ZFfun*)
ultimately show *?thesis* **by** *simp*
qed

lemma *SETCategory*: *Category*(*SET*)
proof−
have *Category-axioms SET′*
by (*auto simp add*: *Category-axioms-def SET′-def ZFfunCompAppDomCod ZFfunCompAppZFfun ZFfunCompAssoc ZFfunIdLeft ZFfunIdRight ZFfunDom ZFfunCod SETfun MapsTo-def CompDefined-def*)
thus *?thesis* **by** (*auto simp add*: *SET-def MakeCat*)
qed

lemma *SETobj*: $X \in Obj\ (SET)$
by (*simp add*: *SET-def SET'-def MakeCat-def*)

lemma *SETcod*: $isZFfun\ (ZFfun\ A\ B\ f) \Longrightarrow cod_{SET}\ ZFfun\ A\ B\ f = B$
by(*simp add*: *SET-def MakeCat-def SET'-def ZFfunCod*)

lemma *SETmor*: $(isZFfun\ f) = (f \in mor_{SET})$
by(*simp add*: *SET-def MakeCat-def SET'-def*)

lemma *SETdom*: $isZFfun\ (ZFfun\ A\ B\ f) \Longrightarrow dom_{SET}\ ZFfun\ A\ B\ f = A$
by(*simp add*: *SET-def MakeCat-def SET'-def ZFfunDom*)

lemma *SETId*: **assumes** $x\ |\in|\ X$ **shows** $(Id\ SET\ X)\ |@|\ x = x$
proof−
have $X \in Obj\ SET$ **by** (*simp add*: *SET-def SET'-def MakeCat-def*)
hence $isZFfun(Id\ SET\ X)$ **by** (*simp add*: *SETCategory Category.CatIdInMor SETmor*)
moreover have $(Id\ SET\ X) = ZFfun\ X\ X\ (\lambda x.\ x)$ **using** *assms* **by** (*simp add*: *SET-def SET'-def MakeCat-def*)
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *ZFfunApp*)
qed

lemma *SETCompE*[*elim*]: $[\![f \approx>_{SET} g\ ;\ [\![isZFfun\ f\ ;\ isZFfun\ g\ ;\ |cod|\ f = |dom|\ g]\!] \Longrightarrow R]\!] \Longrightarrow R$
by (*auto simp add*: *SET-def SET'-def MakeCat-def*)

lemma *SETmapsTo*: $f\ maps_{SET}\ X\ to\ Y \Longrightarrow isZFfun\ f \wedge |dom|\ f = X \wedge |cod|\ f = Y$
by(*auto simp add*: *MapsTo-def SET-def SET'-def MakeCat-def*)

lemma *SETComp*: **assumes** $f \approx>_{SET} g$ **shows** $f\ ;;_{SET}\ g = f\ |o|\ g$
proof−
have *a*: $f \approx>_{MakeCat\ SET'} g$ **using** *assms* **by** (*simp add*: *SET-def*)
have $f\ ;;_{SET}\ g = f\ ;;_{MakeCat\ SET'}\ g$ **by** (*simp add*: *SET-def*)
also have $... = f\ ;;_{SET'}\ g$ **using** *a* **by** (*simp add*: *MakeCatComp2*)
finally show *?thesis* **by** (*simp add*: *SET'-def*)
qed

lemma *SETCompAt*:
assumes $f \approx>_{SET} g$ **and** $x\ |\in|\ |dom|\ f$ **shows** $(f\ ;;_{SET}\ g)\ |@|\ x = g\ |@|\ (f\ |@|\ x)$
proof−
have $f\ ;;_{SET}\ g = f\ |o|\ g$ **using** *assms* **by** (*simp add*: *SETComp*)
also have $... = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))$ **using** *assms* **by** (*auto simp add*: *ZFfunCompApp*)
finally show *?thesis* **using** *assms* **by** (*simp add*: *ZFfunApp*)
qed

lemma *SETZFfun*:
assumes f $maps_{SET}$ X *to* Y **shows** $f = ZFfun\ X\ Y\ (\lambda x\ .\ f\ |@|\ x)$
proof−
have *isZFfun f* **using** *assms* **by** (*auto simp add*: *SETmor*)
hence $f = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ f)\ (\lambda x.\ f\ |@|\ x)$ **by** (*simp add*: *ZFfun*)
moreover have $|dom|\ f = X$ **and** $|cod|\ f = Y$ **using** *assms* **by** (*auto simp add*: *SET-def SET′-def MakeCat-def*)
ultimately show *?thesis* **by** (*simp*)
qed

lemma *SETfunDomAppCod*:
assumes f $maps_{SET}$ X *to* Y **and** $x\ |\in|\ X$
shows $f\ |@|\ x\ |\in|\ Y$
proof−
have *1*: *isZFfun f* **and** $|dom|\ f = X$ **and** *2*: $|cod|\ f = Y$ **using** *assms* **by** (*auto simp add*: *SETmapsTo*)
hence $x\ |\in|\ |dom|\ f$ **using** *assms* **by** *simp*
hence $f\ |@|\ x\ |\in|\ |cod|\ f$ **using** *1* **by** (*simp add*: *ZFfunDomAppCod*)
thus *?thesis* **using** *2* **by** *simp*
qed

record $('o,'m)$ *LSCategory* = $('o,'m)$ *Category* +
mor2ZF :: $'m \Rightarrow ZF$ (‹*m2z*₁-› [*70*] *70*)

definition
ZF2mor (‹*z2m*₁-› [*70*] *70*) **where**
$ZF2mor\ C\ f \equiv THE\ m\ .\ m \in mor_C \wedge m2z_C\ m = f$

definition
$HOMCollection\ C\ X\ Y \equiv \{m2z_C\ f \mid f\ .\ f\ maps_C\ X\ to\ Y\}$

definition
HomSet (‹*Hom*₁ - -› [*65*, *65*] *65*) **where**
$HomSet\ C\ X\ Y \equiv implode\ (HOMCollection\ C\ X\ Y)$

locale *LSCategory* = *Category* +
assumes *mor2ZFInj*: $[\![x \in mor\ ;\ y \in mor\ ;\ m2z\ x = m2z\ y]\!] \Longrightarrow x = y$
and *HOMSetIsSet*: $[\![X \in obj\ ;\ Y \in obj]\!] \Longrightarrow HOMCollection\ C\ X\ Y \in range\ explode$
and *m2zExt*: $mor2ZF\ C \in extensional\ (Mor\ C)$

lemma [*elim*]: $[\![LSCategory\ C\ ;$
$[\![Category\ C\ ;\ [\![x \in mor_C\ ;\ y \in mor_C\ ;\ m2z_C\ x = m2z_C\ y]\!] \Longrightarrow\ x = y;$
$[\![X \in obj_C\ ;\ Y \in obj_C]\!] \Longrightarrow HOMCollection\ C\ X\ Y \in range\ explode]\!] \Longrightarrow R]\!] \Longrightarrow R$
by(*simp add*: *LSCategory-def LSCategory-axioms-def*)

definition

HomFtorMap :: ('o,'m,'a) *LSCategory-scheme* ⇒ 'o ⇒ 'm ⇒ *ZF* (‹*Hom*₁[-,-]› [65,65] 65) **where**
HomFtorMap C X g ≡ *ZFfun* (*Hom*$_C$ *X* (*dom*$_C$ *g*)) (*Hom*$_C$ *X* (*cod*$_C$ *g*)) (λ *f* . *m2z*$_C$ ((*z2m*$_C$ *f*) ;;$_C$ *g*))

definition
HomFtor' :: ('o,'m,'a) *LSCategory-scheme* ⇒ 'o ⇒
('o,*ZF*,'m,*ZF*,⦇*mor2ZF* :: 'm ⇒ *ZF*, ... :: 'a⦈,*unit*) *Functor* (‹*HomP*₁[-,−]› [65] 65) **where**
HomFtor' C X ≡ ⦇
CatDom = *C*,
CatCod = *SET* ,
MapM = λ *g* . *Hom*$_C$[*X*,*g*]
⦈

definition *HomFtor* (‹*Hom*₁[-,−]› [65] 65) **where** *HomFtor C X* ≡ *MakeFtor* (*HomFtor' C X*)

lemma [*simp*]: *LSCategory C* ⟹ *Category C*
by (*simp add*: *LSCategory-def*)

lemma (**in** *LSCategory*) *m2zz2m*:
assumes *f maps X to Y* **shows** (*m2z f*) |∈| (*Hom X Y*)
proof−
have *X* ∈ *Obj C* **and** *Y* ∈ *Obj C* **using** *assms* **by** (*simp add*: *MapsToObj*)+
hence *HOMCollection C X Y* ∈ *range explode* **using** *assms* **by** (*simp add*: *HOMSetIsSet*)
moreover have (*m2z f*) ∈ *HOMCollection C X Y* **using** *assms* **by** (*auto simp add*: *HOMCollection-def*)
ultimately have (*m2z f*) |∈| *implode* (*HOMCollection C X Y*) **by** (*simp add*: *Elem-implode*)
thus *?thesis* **by** (*simp add*: *HomSet-def*)
qed

lemma (**in** *LSCategory*) *m2zz2mInv*:
assumes *f* ∈ *mor*
shows *z2m* (*m2z f*) = *f*
proof−
have *1*: *f* ∈ *mor* ∧ *m2z f* = *m2z f* **using** *assms* **by** *simp*
moreover have ∃! *m* . *m* ∈ *mor* ∧ *m2z m* = (*m2z f*)
proof(*rule ex-ex1I*)
show ∃ *m* . *m* ∈ *mor* ∧ *m2z m* = (*m2z f*)
by(*rule exI*[*of* - *f*], *insert 1*, *simp*)
{
fix *m y* **assume** *m* ∈ *mor* ∧ *m2z m* = (*m2z f*) **and** *y* ∈ *mor* ∧ *m2z y* = (*m2z f*)
thus *m* = *y* **by**(*simp add*: *mor2ZFInj*)
}
qed

ultimately show *?thesis* **by**(*simp add*: *ZF2mor-def the1-equality*)
qed

lemma (**in** *LSCategory*) *z2mm2z*:
assumes $X \in obj$ **and** $Y \in obj$ **and** $f \mid\in\mid (Hom\ X\ Y)$
shows *z2m f maps X to Y* $\wedge$ $m2z\ (z2m\ f) = f$
proof−
have *1*: $\exists\ m\ .$ *m maps X to Y* $\wedge$ $m2z\ m = f$
proof−
have *HOMCollection C X Y* $\in$ *range explode* **using** *assms* **by** (*simp add*: *HOMSetIsSet*)
moreover have $f \mid\in\mid$ *implode* (*HOMCollection C X Y*) **using** *assms*(*3*) **by** (*simp add*: *HomSet-def*)
ultimately have $f \in$ *HOMCollection C X Y* **by** (*simp add*: *HOLZF.Elem-implode*)
thus *?thesis* **by** (*auto simp add*: *HOMCollection-def*)
qed
have *2*: $\exists!\ m\ .\ m \in mor \wedge m2z\ m = f$
proof(*rule ex-ex1I*)
show $\exists\ m\ .\ m \in mor \wedge m2z\ m = f$
proof−
from *1* **obtain** *m* **where** $m \in mor \wedge m2z\ m = f$ **by** *auto*
thus *?thesis* **by** *auto*
qed
{
fix *m y* **assume** $m \in mor \wedge m2z\ m = f$ **and** $y \in mor \wedge m2z\ y = f$
thus $m = y$ **by**(*simp add*: *mor2ZFInj*)
}
qed
thus *?thesis*
proof−
from *1* **obtain** *a* **where** *3*: *a maps X to Y* $\wedge$ $m2z\ a = f$ **by** *auto*
have *4*: $a \in mor$ **using** *3* **by** *auto*
have *z2m f* = *a*
apply (*auto simp add*: *3 ZF2mor-def*[*of - f*])
apply (*rule the1-equality*[*of* $\lambda\ m\ .\ m \in mor \wedge m2z\ m = f\ a$])
apply (*auto simp add*: *2 3 4*)
done
thus *?thesis* **by** (*simp add*: *3*)
qed
qed

lemma *HomFtorMapLemma1*:
assumes *a*: *LSCategory C* **and** *b*: $X \in obj_C$ **and** *c*: $f \in mor_C$ **and** *d*: $x \mid\in\mid (Hom_C\ X\ (dom_C\ f))$
shows $(m2z_C\ ((z2m_C\ x)\ ;;_C\ f)) \mid\in\mid (Hom_C\ X\ (cod_C\ f))$
proof−
have *1*: $dom_C\ f \in obj_C$ **and** *2*: $cod_C\ f \in obj_C$ **using** *a c* **by** (*simp add*: *Category.Simps*)+
have $z2m_C\ x\ maps_C\ X\ to\ (dom_C\ f)$ **using** *a b d 1* **by** (*auto simp add*: *LSCat-*

egory.z2mm2z)
hence ($z2m_C\ x$) $;;_C$ *f maps*$_C$ *X to* ($cod_C\ f$) **using** *a c* **by** (*auto intro*: *Category.Ccompt*)
hence ($m2z_C$ (($z2m_C\ x$) $;;_C\ f$)) |∈| (*Hom*$_C$ *X* ($cod_C\ f$)) **using** *a b d 2*
by (*auto simp add*: *LSCategory.m2zz2m*)
thus *?thesis* **using** *c* **by** (*simp add*: *Category.Simps*)
qed

lemma *HomFtorInMor′*:
assumes *LSCategory C* **and** *X* ∈ *obj*$_C$ **and** *f* ∈ *mor*$_C$
shows *Hom*$_C$[*X*,*f*] ∈ *mor*$_{SET'}$
proof(*simp add*: *HomFtorMap-def*)
{
fix *x* **assume** *x* |∈| (*Hom*$_C$ *X dom*$_C$ *f*)
hence $m2z_C$ (($z2m_C\ x$) $;;_C\ f$) |∈| (*Hom*$_C$ *X cod*$_C$ *f*) **using** *assms* **by** (*blast intro*: *HomFtorMapLemma1*)
}
hence ∀ *x* . *x* |∈| (*Hom*$_C$ *X dom*$_C$ *f*) ⟶ ($m2z_C$ (($z2m_C\ x$) $;;_C\ f$)) |∈| (*Hom*$_C$ *X cod*$_C$ *f*) **by** (*simp*)
hence *isZFfun* (*ZFfun* (*Hom*$_C$ *X dom*$_C$ *f*) (*Hom*$_C$ *X cod*$_C$ *f*) (λ *x* . $m2z_C$ (($z2m_C\ x$) $;;_C\ f$)))
by (*simp add*: *SETfun*)
thus *ZFfun* (*Hom*$_C$ *X dom*$_C$ *f*) (*Hom*$_C$ *X cod*$_C$ *f*) (λ *x* . $m2z_C$ (($z2m_C\ x$) $;;_C$ *f*)) ∈ *mor*$_{SET'}$
by (*simp add*: *SET′-def*)
qed

lemma *HomFtorMor′*:
assumes *LSCategory C* **and** *X* ∈ *obj*$_C$ **and** *f* ∈ *mor*$_C$
shows *Hom*$_C$[*X*,*f*] *maps*$_{SET'}$ *Hom*$_C$ *X* (*dom*$_C$ *f*) *to Hom*$_C$ *X* (*cod*$_C$ *f*)
proof−
have *Hom*$_C$[*X*,*f*] ∈ *mor*$_{SET'}$ **using** *assms* **by** (*simp add*: *HomFtorInMor′*)
moreover have *dom*$_{SET'}$ (*Hom*$_C$[*X*,*f*]) = *Hom*$_C$ *X* (*dom*$_C$ *f*)
by(*simp add*: *HomFtorMap-def SET′-def ZFfunDom*)
moreover have *cod*$_{SET'}$ (*Hom*$_C$[*X*,*f*]) = *Hom*$_C$ *X* (*cod*$_C$ *f*)
by(*simp add*: *HomFtorMap-def SET′-def ZFfunCod*)
ultimately show *?thesis* **by** (*auto simp add*: *SET-def*)
qed

lemma *HomFtorMapsTo*:
⟦*LSCategory C* ; *X* ∈ *obj*$_C$; *f* ∈ *mor*$_C$ ⟧ ⟹ *Hom*$_C$[*X*,*f*] *maps*$_{SET}$ *Hom*$_C$ *X* (*dom*$_C$ *f*) *to Hom*$_C$ *X* (*cod*$_C$ *f*)
by (*simp add*: *HomFtorMor′ SET-def MakeCatMapsTo*)

lemma *HomFtorMor*:
assumes *LSCategory C* **and** *X* ∈ *obj*$_C$ **and** *f* ∈ *mor*$_C$
shows *Hom*$_C$[*X*,*f*] ∈ *Mor SET* **and** *dom*$_{SET}$ (*Hom*$_C$[*X*,*f*]) = *Hom*$_C$ *X* (*dom*$_C$ *f*)
and *cod*$_{SET}$ (*Hom*$_C$[*X*,*f*]) = *Hom*$_C$ *X* (*cod*$_C$ *f*)

proof−
  have $Hom_C[X,f]$ $maps_{SET}$ $Hom_C\ X\ (dom_C\ f)$ *to* $Hom_C\ X\ (cod_C\ f)$ **using** *assms* **by** (*simp add*: *HomFtorMapsTo*)
  thus $Hom_C[X,f] \in Mor\ SET$ **and** $dom_{SET}\ (Hom_C[X,f]) = Hom_C\ X\ (dom_C\ f)$ **and** $cod_{SET}\ (Hom_C[X,f]) = Hom_C\ X\ (cod_C\ f)$
    by *auto*
qed

lemma *HomFtorCompDef′*:
  assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \approx>_C g$
  shows $(Hom_C[X,f]) \approx>_{SET'} (Hom_C[X,g])$
proof(*rule CompDefinedI*)
  have *a*: $f \in mor_C$ **and** *b*: $g \in mor_C$ **using** *assms*(*3*) **by** *auto*
  thus $Hom_C[X,f] \in mor_{SET'}$ **and** $Hom_C[X,g] \in mor_{SET'}$ **using** *assms* **by** (*simp add*:*HomFtorInMor′*)+
  have $(Hom_C[X,f])$ $maps_{SET'}$ $Hom_C\ X\ dom_C\ f$ *to* $Hom_C\ X\ cod_C\ f$
   and $(Hom_C[X,g])$ $maps_{SET'}$ $Hom_C\ X\ dom_C\ g$ *to* $Hom_C\ X\ cod_C\ g$ **using** *assms a b* **by** (*simp add*: *HomFtorMor′*)+
  hence $cod_{SET'}\ (Hom_C[X,f]) = Hom_C\ X\ (cod_C\ f)$
    and $dom_{SET'}\ (Hom_C[X,g]) = Hom_C\ X\ (dom_C\ g)$ **by** *auto*
  moreover have $(cod_C\ f) = (dom_C\ g)$ **using** *assms*(*3*) **by** *auto*
  ultimately show $cod_{SET'}\ (Hom_C[X,f]) = dom_{SET'}\ (Hom_C[X,g])$ **by** *simp*
qed

lemma *HomFtorDist′*:
  assumes *a*: *LSCategory C* **and** *b*: $X \in obj_C$ **and** *c*: $f \approx>_C g$
  shows $(Hom_C[X,f])\ ;;_{SET'}\ (Hom_C[X,g]) = Hom_C[X,f\ ;;_C\ g]$
proof−
  let *?A* = $(Hom_C\ X\ dom_C\ f)$
  let *?B* = $(Hom_C\ X\ dom_C\ g)$
  let *?C* = $(Hom_C\ X\ cod_C\ g)$
  let *?f* = $(\lambda h.\ m2z_C\ ((z2m_C\ h)\ ;;_C\ f))$
  let *?g* = $(\lambda f.\ m2z_C\ ((z2m_C\ f)\ ;;_C\ g))$
  have *1*: $cod_C\ f = dom_C\ g$ **using** *c* **by** *auto*
  have *2*: $dom_C\ (f\ ;;_C\ g) = dom_C\ f$ **and** *3*: $cod_C\ (f\ ;;_C\ g) = cod_C\ g$ **using** *assms*
    by (*auto simp add*: *Category.MapsToMorDomCod*)
  have $(Hom_C[X,f])\ ;;_{SET'}\ (Hom_C[X,g]) = (ZFfun\ ?A\ (Hom_C\ X\ cod_C\ f)\ ?f)\ |o|\ (ZFfun\ ?B\ ?C\ ?g)$
    by (*simp add*: *HomFtorMap-def SET′-def*)
  also have ... = $(ZFfun\ ?A\ ?B\ ?f)\ |o|\ (ZFfun\ ?B\ ?C\ ?g)$ **using** *1* **by** *simp*
  also have ... = $ZFfun\ ?A\ ?C\ (?g\ o\ ?f)$
  proof(*rule ZFfunComp*, *rule allI*, *rule impI*)
    {
      fix *h* **assume** *aa*: $h\ |\in|\ ?A$ **show** $?f\ h\ |\in|\ ?B$
      proof−
        have $f \in mor_C$ **using** *assms* **by** *auto*
      hence $?f\ h\ |\in|\ (Hom_C\ X\ cod_C\ f)$ **using** *assms aa* **by** (*simp add*: *HomFtorMapLemma1*)
        thus *?thesis* **using** *1* **by** *simp*

```
    qed
   }
  qed
  also have ... = ZFfun ?A ?C (λh. m2z_C ((z2m_C h) ;;_C (f ;;_C g)))
  proof(rule ZFfun-ext, rule allI, rule impI, simp add: comp-def)
   {
    fix h assume aa: h |∈| ?A
    show m2z_C ((z2m_C (m2z_C((z2m_C h) ;;_C f))) ;;_C g) = m2z_C ((z2m_C h) ;;_C
(f ;;_C g))
    proof−
     have bb: (z2m_C h) ≈>_C f
     proof(rule CompDefinedI)
      show f ∈ mor_C using c by auto
      hence dom_C f ∈ obj_C using a by (simp add: Category.Cdom)
       hence (z2m_C h) maps_C X to dom_C f using assms aa by (simp add:
LSCategory.z2mm2z)
       thus (z2m_C h) ∈ mor_C and cod_C (z2m_C h) = dom_C f by auto
     qed
    hence (z2m_C h) ;;_C f ∈ mor_C using a by (simp add: Category.MapsToMorDomCod)
     hence z2m_C (m2z_C ((z2m_C h) ;;_C f)) = (z2m_C h) ;;_C f using a by (simp
add: LSCategory.m2zz2mInv)
     hence m2z_C ((z2m_C (m2z_C((z2m_C h) ;;_C f))) ;;_C g) = m2z_C (((z2m_C h)
;;_C f) ;;_C g) by simp
     also have ... = m2z_C ((z2m_C h) ;;_C (f ;;_C g)) using bb c a by (simp add:
Category.Cassoc)
     finally show ?thesis .
    qed
   }
  qed
  also have ... = ZFfun (Hom_C X dom_C (f ;;_C g)) (Hom_C X cod_C (f ;;_C g)) (λh.
m2z_C ((z2m_C h) ;;_C (f ;;_C g)))
    using 2 3 by simp
  also have ... = Hom_C[X,f ;;_C g] by (simp add:  HomFtorMap-def)
  finally show ?thesis by (auto simp add: SET-def)
qed

lemma HomFtorDist:
  assumes LSCategory C and X ∈ obj_C and f ≈>_C g
  shows   (Hom_C[X,f]) ;;_SET (Hom_C[X,g]) = Hom_C[X,f ;;_C g]
proof−
   have (Hom_C[X,f]) ;;_SET' (Hom_C[X,g]) = Hom_C[X,f ;;_C g] using assms by
(simp add: HomFtorDist')
   moreover have (Hom_C[X,f]) ≈>_SET' (Hom_C[X,g]) using assms by (simp
add: HomFtorCompDef')
  ultimately show ?thesis by (simp add: MakeCatComp SET-def)
qed

lemma HomFtorId':
  assumes a: LSCategory C and b: X ∈ obj_C and c: Y ∈ obj_C
```

shows $Hom_C[X,id_C\ Y] = id_{SET'}\ (Hom_C\ X\ Y)$
proof−
have $(id_C\ Y)\ maps_C\ Y\ to\ Y$ **using** $a\ c$ **by** (*simp add*: *Category.Simps*)
hence *1*: $(dom_C\ (id_C\ Y)) = Y$ **and** *2*: $(cod_C\ (id_C\ Y)) = Y$ **by** *auto*
have $Hom_C[X,id_C\ Y] = ZFfun\ (Hom_C\ X\ (dom_C\ (id_C\ Y)))\ (Hom_C\ X\ (cod_C\ (id_C\ Y)))\ (\lambda\ f\ .\ m2z_C\ ((z2m_C\ f)\ ;;_C\ (id_C\ Y)))$
by (*simp add*: *HomFtorMap-def*)
also have $... = ZFfun\ (Hom_C\ X\ Y)\ (Hom_C\ X\ Y)\ (\lambda\ f\ .\ m2z_C\ ((z2m_C\ f)\ ;;_C\ (id_C\ Y)))$ **using** *1 2* **by** *simp*
also have $... = ZFfun\ (Hom_C\ X\ Y)\ (Hom_C\ X\ Y)\ (\lambda\ f\ .\ f)$
proof(*rule ZFfun-ext*, *rule allI*, *rule impI*)
{
fix h **assume** *aa*: $h\ |\in|\ (Hom_C\ X\ Y)$ **show** $m2z_C\ ((z2m_C\ h)\ ;;_C\ (id_C\ Y)) = h$
proof−
have $(z2m_C\ h)\ maps_C\ X\ to\ Y$ **and** *bb*: $m2z_C\ (z2m_C\ h) = h$
using *assms aa* **by** (*simp add*: *LSCategory.z2mm2z*)+
hence $(z2m_C\ h)\ ;;_C\ (id_C\ Y) = (z2m_C\ h)$ **using** a **by** (*auto simp add*: *Category.Simps*)
hence $m2z_C\ ((z2m_C\ h)\ ;;_C\ (id_C\ Y)) = m2z_C\ (z2m_C\ h)$ **by** *simp*
also have $... = h$ **using** *bb* **.**
finally show *?thesis* **.**
qed
}
qed
finally show *?thesis* **by** (*simp add*: *SET'-def*)
qed

lemma *HomFtorId*:
assumes *LSCategory C* **and** $X \in obj_C$ **and** $Y \in obj_C$
shows $Hom_C[X,id_C\ Y] = id_{SET}\ (Hom_C\ X\ Y)$
proof−
have $Hom_C[X,id_C\ Y] = id_{SET'}\ (Hom_C\ X\ Y)$ **using** *assms* **by** (*simp add*: *HomFtorId'*)
moreover have $(Hom_C\ X\ Y) \in obj_{SET'}$ **by** (*simp add*: *SET'-def*)
ultimately show *?thesis* **by** (*simp add*: *MakeCatId SET-def*)
qed

lemma *HomFtorObj'*:
assumes *a*: *LSCategory C*
and *b*: *PreFunctor* $(HomP_C[X,-])$ **and** *c*: $X \in obj_C$ **and** *d*: $Y \in obj_C$
shows $(HomP_C[X,-])\ @@\ Y = Hom_C\ X\ Y$
proof−
let *?F* = $(HomFtor'\ C\ X)$
have $?F\ \#\#\ (id_{CatDom\ ?F}\ Y) = Hom_C[X,id_C\ Y]$ **by** (*simp add*: *HomFtor'-def*)
also have $... = id_{CatCod\ ?F}\ (Hom_C\ X\ Y)$ **using** *assms* **by** (*simp add*: *HomFtorId HomFtor'-def*)
finally have $?F\ \#\#\ (id_{CatDom\ ?F}\ Y) = id_{CatCod\ ?F}\ (Hom_C\ X\ Y)$ **by** *simp*
moreover have $Hom_C\ X\ Y \in obj_{CatCod\ ?F}$ **using** *assms*

```
   by (simp add: HomFtorId HomFtor'-def SET-def SET'-def MakeCatObj)
  moreover have Y ∈ obj_CatDom ?F using d by (simp add: HomFtor'-def)
  ultimately show ?thesis using b by(simp add: PreFunctor.FmToFo[of ?F Y
Hom_C X Y])
qed

lemma HomFtorFtor':
  assumes a: LSCategory C
  and     b: X ∈ obj_C
  shows   FunctorM (HomP_C[X,−])
proof(intro-locales)
  show PF: PreFunctor (HomP_C[X,−])
  proof(auto simp add: HomFtor'-def PreFunctor-def SETCategory a HomFtorDist
b)
    {
      fix Z assume aa: Z ∈ obj_C
      show ∃ Y ∈ obj_SET . Hom_C[X,id_C Z] = id_SET Y
      proof(rule-tac x=Hom_C X Z in Set.rev-bexI)
       show Hom_C X Z ∈ obj_SET by (simp add: SET-def SET'-def MakeCatObj)

         show Hom_C[X,id_C Z] = id_SET (Hom_C X Z) using assms aa by(simp
add:HomFtorId)
      qed
    }
  qed
  {
    fix f Z Y assume aa: f maps_C Z to Y
   have (HomP_C[X,−]) ## f maps_SET ((HomP_C[X,−]) @@ Z) to ((HomP_C[X,−])
@@ Y)
    proof−
       have bb: Z ∈ obj_C and cc: Y ∈ obj_C using aa a by (simp add: Cate-
gory.MapsToObj)+
      have dd: dom_C f = Z and ee: cod_C f = Y and ff: f ∈ mor_C using aa by
auto
      have (HomP_C[X,−]) ## f = Hom_C[X,f] by (simp add: HomFtor'-def)
      moreover have (HomP_C[X,−]) @@ Z = Hom_C X Z
       and (HomP_C[X,−]) @@ Y = Hom_C X Y using assms bb cc PF by (simp
add: HomFtorObj')+
       moreover have Hom_C[X,f] maps_SET (Hom_C X (dom_C f)) to (Hom_C X
(cod_C f))
        using assms ff by (simp add: HomFtorMapsTo)
      ultimately show ?thesis using dd ee by simp
    qed
  }
  thus FunctorM-axioms (HomP_C[X,−]) using PF by (auto simp add: Func-
torM-axioms-def HomFtor'-def)
qed

lemma HomFtorFtor:
```

assumes *a*: *LSCategory C*
and *b*: $X \in obj_C$
shows *Functor* ($Hom_C[X,-]$)
proof−
have *FunctorM* ($HomP_C[X,-]$) **using** *assms* **by** (*rule HomFtorFtor′*)
thus *?thesis* **by** (*simp add*: *HomFtor-def MakeFtor*)
qed

lemma *HomFtorObj*:
assumes *LSCategory C*
and $X \in obj_C$ **and** $Y \in obj_C$
shows ($Hom_C[X,-]$) @@ $Y = Hom_C\ X\ Y$
proof−
have *FunctorM* ($HomP_C[X,-]$) **using** *assms* **by** (*simp add*: *HomFtorFtor′*)
hence *1*: *PreFunctor* ($HomP_C[X,-]$) **by** (*simp add*: *FunctorM-def*)
moreover have *CatDom* ($HomP_C[X,-]$) = *C* **by** (*simp add*: *HomFtor′-def*)
ultimately have ($Hom_C[X,-]$) @@ Y = ($HomP_C[X,-]$) @@ Y **using** *assms*
by (*simp add*: *MakeFtorObj HomFtor-def*)
thus *?thesis* **using** *assms 1* **by** (*simp add*: *HomFtorObj′*)
qed

definition
HomFtorMapContra :: ('*o*,'*m*,'*a*) *LSCategory-scheme* ⇒ '*m* ⇒ '*o* ⇒ *ZF* (‹$HomC_1$[-,-]›
[*65*,*65*] *65*) **where**
HomFtorMapContra C g X ≡ *ZFfun* (Hom_C (cod_C *g*) *X*) (Hom_C (dom_C *g*) *X*)
(λ *f* . $m2z_C$ (*g* ;;$_C$ ($z2m_C$ *f*)))

definition
HomFtorContra′ :: ('*o*,'*m*,'*a*) *LSCategory-scheme* ⇒ '*o* ⇒
('*o*,*ZF*,'*m*,*ZF*,⦇*mor2ZF* :: '*m* ⇒ *ZF*, ... :: '*a*⦈,*unit*) *Functor* (‹$HomP_1$[−,-]›
[*65*] *65*) **where**
HomFtorContra′ C X ≡ ⦇
CatDom = (*Op C*),
CatCod = *SET* ,
MapM = λ *g* . $HomC_C$[*g*,*X*]
⦈

definition *HomFtorContra* (‹Hom_1[−,-]› [*65*] *65*) **where** *HomFtorContra C X* ≡
MakeFtor(*HomFtorContra′ C X*)

lemma *HomContraAt*: *x* |∈| (Hom_C (cod_C *f*) *X*) ⟹ ($HomC_C$[*f*,*X*]) |@| *x* =
$m2z_C$ (*f* ;;$_C$ ($z2m_C$ *x*))
by (*simp add*: *HomFtorMapContra-def ZFfunApp*)

lemma *mor2ZF-Op*: *mor2ZF* (*Op C*) = *mor2ZF C*
apply (*cases C*)
apply (*simp add*: *OppositeCategory-def*)
done

lemma *mor-Op*: $mor_{Op\ C} = mor_C$ **by** (*simp add*: *OppositeCategory-def*)
lemma *obj-Op*: $obj_{Op\ C} = obj_C$ **by** (*simp add*: *OppositeCategory-def*)

lemma *ZF2mor-Op*: *ZF2mor* (*Op C*) *f* = *ZF2mor C f*
by (*simp add*: *ZF2mor-def mor2ZF-Op mor-Op*)

lemma *mapsTo-Op*: *f* $maps_{Op\ C}$ *Y to X* = *f* $maps_C$ *X to Y*
by (*auto simp add*: *OppositeCategory-def mor-Op MapsTo-def*)

lemma *HOMCollection-Op*: *HOMCollection* (*Op C*) *X Y* = *HOMCollection C Y X*
by (*simp add*: *HOMCollection-def mapsTo-Op mor2ZF-Op*)

lemma *Hom-Op*: $Hom_{Op\ C}\ X\ Y = Hom_C\ Y\ X$
by (*simp add*: *HomSet-def HOMCollection-Op*)

lemma *HomFtorContra'*: $HomP_C[-,X] = HomP_{Op\ C}[X,-]$
apply (*simp add*: *HomFtorContra'-def*
HomFtor'-def HomFtorMapContra-def HomFtorMap-def mor2ZF-Op ZF2mor-Op Hom-Op)
by (*simp add*: *OppositeCategory-def*)

lemma *HomFtorContra*: $Hom_C[-,X] = Hom_{Op\ C}[X,-]$
by (*auto simp add*: *HomFtorContra' HomFtorContra-def HomFtor-def*)

lemma *HomFtorContraDom*: *CatDom* ($Hom_C[-,X]$) = *Op C*
by(*simp add*: *HomFtorContra-def HomFtorContra'-def MakeFtor-def*)

lemma *HomFtorContraCod*: *CatCod* ($Hom_C[-,X]$) = *SET*
by(*simp add*: *HomFtorContra-def HomFtorContra'-def MakeFtor-def*)

lemma *LSCategory-Op*: **assumes** *LSCategory C* **shows** *LSCategory* (*Op C*)
proof(*auto simp only*: *LSCategory-def*)
show *Category* (*Op C*) **using** *assms* **by** (*simp add*: *OpCatCat*)
show *LSCategory-axioms* (*Op C*) **using** *assms*
by (*simp add*: *LSCategory-axioms-def mor-Op obj-Op mor2ZF-Op HOMCollection-Op*
LSCategory.mor2ZFInj LSCategory.HOMSetIsSet LSCategory.m2zExt)
qed

lemma *HomFtorContraFtor*:
assumes *LSCategory C*
and $X \in obj_C$
shows $Ftor\ (Hom_C[-,X]) : (Op\ C) \longrightarrow SET$
proof(*auto simp only*: *functor-abbrev-def*)
show *Functor* ($Hom_C[-,X]$)
proof−
have $Hom_C[-,X] = Hom_{Op\ C}[X,-]$ **by** (*simp add*: *HomFtorContra*)

moreover have *LSCategory* (*Op C*) **using** *assms* **by** (*simp add*: *LSCategory-Op*)
moreover have $X \in obj_{Op\ C}$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *HomFtorFtor*)
qed
show *CatDom* ($Hom_C[-,X]$) = *Op C* **by**(*simp add*: *HomFtorContra-def HomFtorContra'-def MakeFtor-def*)
show *CatCod* ($Hom_C[-,X]$) = *SET* **by**(*simp add*: *HomFtorContra-def HomFtorContra'-def MakeFtor-def*)
qed

lemma *HomFtorOpObj*:
assumes *LSCategory C*
and $X \in obj_C$ **and** $Y \in obj_C$
shows ($Hom_C[-,X]$) @@ $Y = Hom_C\ Y\ X$
proof−
have *1*: $X \in Obj\ (Op\ C)$ **and** *2*: $Y \in Obj\ (Op\ C)$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)+
have ($Hom_C[-,X]$) @@ $Y = (Hom_{Op\ C}[X,-])$ @@ Y **by** (*simp add*: *HomFtorContra*)
also have ... = ($Hom_{Op\ C}\ X\ Y$) **using** *assms*(*1*) *1 2* **by** (*simp add*: *LSCategory-Op HomFtorObj*)
also have ... = ($Hom_C\ Y\ X$) **by** (*simp add*: *Hom-Op*)
finally show *?thesis* **.**
qed

lemma *HomCHomOp*: $HomC_C[g,X] = Hom_{Op\ C}[X,g]$
apply (*simp add*: *HomFtorContra'-def HomFtor'-def HomFtorMapContra-def HomFtorMap-def mor2ZF-Op ZF2mor-Op Hom-Op*)
by (*simp add*: *OppositeCategory-def*)

lemma *HomFtorContraMapsTo*:
assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \in mor_C$
shows $HomC_C[f,X]\ maps_{SET}\ Hom_C\ (cod_C\ f)\ X$ *to* $Hom_C\ (dom_C\ f)\ X$
proof−
have *LSCategory* (*Op C*) **using** *assms* **by**(*simp add*: *LSCategory-Op*)
moreover have $X \in Obj\ (Op\ C)$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)
moreover have $f \in Mor\ (Op\ C)$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)
ultimately have $Hom_{Op\ C}[X,f]\ maps_{SET}\ Hom_{Op\ C}\ X\ (dom_{Op\ C}\ f)$ *to* $Hom_{Op\ C}\ X\ (cod_{Op\ C}\ f)$ **using** *assms*
by (*simp add*: *HomFtorMapsTo*)
moreover have $HomC_C[f,X] = Hom_{Op\ C}[X,f]$ **by** (*simp add*: *HomCHomOp*)
moreover have $Hom_{Op\ C}\ X\ (dom_{Op\ C}\ f) = Hom_C\ (cod_C\ f)\ X$
proof−

have $Hom_{Op\ C}\ X\ (dom_{Op\ C}\ f) = Hom_C\ (dom_{Op\ C}\ f)\ X$ **by** (*simp add*: *Hom-Op*)
thus *?thesis* **by** (*simp add*: *OppositeCategory-def*)
qed
moreover have $Hom_{Op\ C}\ X\ (cod_{Op\ C}\ f) = Hom_C\ (dom_C\ f)\ X$
proof−
have $Hom_{Op\ C}\ X\ (cod_{Op\ C}\ f) = Hom_C\ (cod_{Op\ C}\ f)\ X$ **by** (*simp add*: *Hom-Op*)
thus *?thesis* **by** (*simp add*: *OppositeCategory-def*)
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *HomFtorContraMor*:
assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \in mor_C$
shows $HomC_C[f,X] \in Mor\ SET$ **and** $dom_{SET}\ (HomC_C[f,X]) = Hom_C\ (cod_C\ f)\ X$
and $cod_{SET}\ (HomC_C[f,X]) = Hom_C\ (dom_C\ f)\ X$
proof−
have $HomC_C[f,X]\ maps_{SET}\ Hom_C\ (cod_C\ f)\ X\ \ to\ Hom_C\ (dom_C\ f)\ X$ **using** *assms* **by** (*simp add*: *HomFtorContraMapsTo*)
thus $HomC_C[f,X] \in Mor\ SET$ **and** $dom_{SET}\ (HomC_C[f,X]) = Hom_C\ (cod_C\ f)\ X$
and $cod_{SET}\ (HomC_C[f,X]) = Hom_C\ (dom_C\ f)\ X$
by *auto*
qed

lemma *HomContraMor*:
assumes *LSCategory C* **and** $f \in Mor\ C$
shows $(Hom_C[-,X])\ \#\#\ f = HomC_C[f,X]$
by(*simp add*: *HomFtorContra-def HomFtorContra′-def MakeFtor-def assms OppositeCategory-def*)

lemma *HomCHom*:
assumes *LSCategory C* **and** $f \in Mor\ C$ **and** $g \in Mor\ C$
shows $(HomC_C[g,dom_C\ f])\ ;;_{SET}\ (Hom_C[dom_C\ g,f]) = (Hom_C[cod_C\ g,f])\ ;;_{SET}\ (HomC_C[g,cod_C\ f])$
proof−
have *ObjDf*: $dom_C\ f \in obj_C$ **and** *ObjDg*: $dom_C\ g \in obj_C$ **using** *assms* **by** (*simp add*: *Category.Cdom*)+
have *ObjCg*: $cod_C\ g \in obj_C$ **and** *ObjCf*: $cod_C\ f \in obj_C$ **using** *assms* **by** (*simp add*: *Category.Ccod*)+
have $(HomC_C[g,dom_C\ f])\ ;;_{SET}\ (Hom_C[dom_C\ g,f]) = (HomC_C[g,dom_C\ f])\ |o|\ (Hom_C[dom_C\ g,f])$
proof−
have $(HomC_C[g,dom_C\ f])\ \approx>_{SET}\ (Hom_C[dom_C\ g,f])$
proof(*rule CompDefinedI*)

show $Hom_C[dom_C\ g,f] \in Mor\ SET$ **using** *assms ObjDg* **by** (*simp add*: *HomFtorMor*)
show $HomC_C[g,dom_C\ f] \in Mor\ SET$ **using** *assms ObjDf* **by** (*simp add*: *HomFtorContraMor*)
show $cod_{SET}\ (HomC_C[g,dom_C\ f]) = dom_{SET}\ (Hom_C[dom_C\ g,f])$ **using** *assms ObjDg ObjDf*
by (*simp add*: *HomFtorMor HomFtorContraMor*)
qed
thus *?thesis* **by**(*simp add*: *SET-def SET'-def MakeCatComp2*)
qed
also have ... = $ZFfun\ (Hom_C\ (cod_C\ g)\ (dom_C\ f))\ (Hom_C\ (dom_C\ g)\ (cod_C\ f))$
$((\lambda\ h\ .\ m2z_C\ ((z2m_C\ h)\ ;;_C\ f))\ o\ (\lambda\ h\ .\ m2z_C\ (g\ ;;_C\ (z2m_C\ h))))$
proof(*simp add*: *HomFtorMapContra-def HomFtorMap-def*, *rule ZFfunComp*, *rule allI*, *rule impI*)
{
fix *x* **assume** *aa*: $x\ |\in|\ (Hom_C\ (cod_C\ g)\ (dom_C\ f))$
show $(m2z_C\ (g\ ;;_C\ (z2m_C\ x)))\ |\in|\ (Hom_C\ (dom_C\ g)\ (dom_C\ f))$
proof(*rule LSCategory.m2zz2m*, *simp-all add*: *assms*(*1*) *ObjDg ObjDf*)
have $g\ maps_C\ (dom_C\ g)\ to\ (cod_C\ g)$ **using** *assms* **by** *auto*
moreover have $(z2m_C\ x)\ maps_C\ (cod_C\ g)\ to\ (dom_C\ f)$ **using** *aa ObjCg ObjDf assms*(*1*)
by (*simp add*: *LSCategory.z2mm2z*)
ultimately show $g\ ;;_C\ (z2m_C\ x)\ maps_C\ (dom_C\ g)\ to\ (dom_C\ f)$ **using** *assms*(*1*)
by (*simp add*: *Category.Ccompt*)
qed
}
qed
also have ... = $ZFfun\ (Hom_C\ (cod_C\ g)\ (dom_C\ f))\ (Hom_C\ (dom_C\ g)\ (cod_C\ f))$
$((\lambda\ h\ .\ m2z_C\ (g\ ;;_C\ (z2m_C\ h)))\ o\ (\lambda\ h\ .\ m2z_C\ ((z2m_C\ h)\ ;;_C\ f)))$
proof(*rule ZFfun-ext*, *rule allI*, *rule impI*)
{
fix *h* **assume** *aa*: $h\ |\in|\ (Hom_C\ (cod_C\ g)\ (dom_C\ f))$
show $((\lambda\ h\ .\ m2z_C\ ((z2m_C\ h)\ ;;_C\ f))\ o\ (\lambda\ h\ .\ m2z_C\ (g\ ;;_C\ (z2m_C\ h))))\ h =$
$((\lambda\ h\ .\ m2z_C\ (g\ ;;_C\ (z2m_C\ h)))\ o\ (\lambda\ h\ .\ m2z_C\ ((z2m_C\ h)\ ;;_C\ f)))\ h$
proof−
have *MapsTo1*: $(z2m_C\ h)\ maps_C\ (cod_C\ g)\ to\ (dom_C\ f)$ **using** *assms*(*1*) *ObjCg ObjDf aa* **by** (*simp add*: *LSCategory.z2mm2z*)
have *CompDef1*: $(z2m_C\ h) \approx>_C f$
proof(*rule CompDefinedI*)
show $f \in mor_C$ **using** *assms* **by** *simp*
show $(z2m_C\ h) \in mor_C$ **and** $cod_C\ (z2m_C\ h) = dom_C\ f$ **using** *MapsTo1* **by** *auto*
qed
have *CompDef2*: $g \approx>_C (z2m_C\ h)$
proof(*rule CompDefinedI*)
show $g \in mor_C$ **using** *assms* **by** *simp*
thus $(z2m_C\ h) \in mor_C$ **and** $cod_C\ g = dom_C\ (z2m_C\ h)$ **using** *MapsTo1* **by** *auto*

qed
have *c1*: $(z2m_C\ h)\ ;;_C\ f \in Mor\ C$ **using** *assms CompDef1* **by** (*simp add*: *Category.MapsToMorDomCod*)
have *c2*: $g\ ;;_C\ (z2m_C\ h) \in Mor\ C$ **using** *assms CompDef2* **by** (*simp add*: *Category.MapsToMorDomCod*)
have $g\ ;;_C\ (z2m_C\ (m2z_C\ ((z2m_C\ h)\ ;;_C\ f))) = g\ ;;_C\ ((z2m_C\ h)\ ;;_C\ f)$ **using** *assms*(*1*) *c1*
by (*simp add*: *LSCategory.m2zz2mInv*)
also have $... = (g\ ;;_C\ (z2m_C\ h))\ ;;_C\ f$ **using** *CompDef1 CompDef2 assms* **by** (*simp add*: *Category.Cassoc*)
also have $... = (z2m_C\ (m2z_C\ (g\ ;;_C\ (z2m_C\ h))))\ ;;_C\ f$ **using** *assms*(*1*) *c2*
by (*simp add*: *LSCategory.m2zz2mInv*)
finally have $g\ ;;_C\ (z2m_C\ (m2z_C\ ((z2m_C\ h)\ ;;_C\ f))) = (z2m_C\ (m2z_C\ (g\ ;;_C\ (z2m_C\ h))))\ ;;_C\ f$ **.**
thus *?thesis* **by** *simp*
qed
}
qed
also have $... = (Hom_C[cod_C\ g,f])\ |o|\ (HomC_C[g,cod_C\ f])$
proof(*simp add*: *HomFtorMapContra-def HomFtorMap-def*, *rule ZFfunComp*[*THEN sym*], *rule allI*, *rule impI*)
{
fix *x* **assume** *aa*: $x\ |\in|\ (Hom_C\ cod_C\ g\ dom_C\ f)$
show $m2z_C\ ((z2m_C\ x)\ ;;_C\ f)\ |\in|\ (Hom_C\ cod_C\ g\ cod_C\ f)$
proof(*rule LSCategory.m2zz2m*, *simp-all add*: *assms*(*1*) *ObjCg ObjCf*)
have $f\ maps_C\ (dom_C\ f)\ to\ (cod_C\ f)$ **using** *assms* **by** *auto*
moreover have $(z2m_C\ x)\ maps_C\ (cod_C\ g)\ to\ (dom_C\ f)$ **using** *aa ObjCg ObjDf assms*(*1*)
by (*simp add*: *LSCategory.z2mm2z*)
ultimately show $(z2m_C\ x)\ ;;_C\ f\ maps_C\ cod_C\ g\ to\ cod_C\ f$ **using** *assms*(*1*)
by (*simp add*: *Category.Ccompt*)
qed
}
qed
also have $... = (Hom_C[cod_C\ g,f])\ ;;_{SET}\ (HomC_C[g,cod_C\ f])$
proof−
have $(Hom_C[cod_C\ g,f])\ \approx>_{SET}\ (HomC_C[g,cod_C\ f])$
proof(*rule CompDefinedI*)
show $Hom_C[cod_C\ g,f] \in Mor\ SET$ **using** *assms ObjCg* **by** (*simp add*: *HomFtorMor*)
show $HomC_C[g,cod_C\ f] \in Mor\ SET$ **using** *assms ObjCf* **by** (*simp add*: *HomFtorContraMor*)
show $cod_{SET}\ (Hom_C[cod_C\ g,f]) = dom_{SET}\ (HomC_C[g,cod_C\ f])$ **using** *assms ObjCg ObjCf*
by (*simp add*: *HomFtorMor HomFtorContraMor*)
qed
thus *?thesis* **by**(*simp add*: *SET-def SET′-def MakeCatComp2*)
qed
finally show *?thesis* **.**

qed

end

7 Yoneda

theory *Yoneda*
imports *NatTrans SetCat*
begin

definition *YFtorNT′ C f* ≡ (|*NTDom* = Hom_C[−,dom_C *f*] , *NTCod* = Hom_C[−,cod_C *f*] ,
NatTransMap = λ *B* . Hom_C[*B*,*f*]|)

definition *YFtorNT C f* ≡ *MakeNT* (*YFtorNT′ C f*)

lemmas *YFtorNT-defs* = *YFtorNT′-def YFtorNT-def MakeNT-def*

lemma *YFtorNTCatDom*: *NTCatDom* (*YFtorNT C f*) = *Op C*
by (*simp add*: *YFtorNT-defs NTCatDom-def HomFtorContraDom*)

lemma *YFtorNTCatCod*: *NTCatCod* (*YFtorNT C f*) = *SET*
by (*simp add*: *YFtorNT-defs NTCatCod-def HomFtorContraCod*)

lemma *YFtorNTApp1*: **assumes** *X* ∈ *Obj* (*NTCatDom* (*YFtorNT C f*)) **shows** (*YFtorNT C f*) $$ *X* = Hom_C[*X*,*f*]
proof−
have (*YFtorNT C f*) $$ *X* = (*YFtorNT′ C f*) $$ *X* **using** *assms* **by** (*simp add*: *MakeNTApp YFtorNT-def*)
thus *?thesis* **by** (*simp add*: *YFtorNT′-def*)
qed

definition
YFtor′ C ≡ (|
CatDom = *C* ,
CatCod = *CatExp* (*Op C*) *SET* ,
MapM = λ *f* . *YFtorNT C f*
|)

definition *YFtor C* ≡ *MakeFtor*(*YFtor′ C*)

lemmas *YFtor-defs* = *YFtor′-def YFtor-def MakeFtor-def*

lemma *YFtorNTNatTrans′*:
assumes *LSCategory C* **and** *f* ∈ *Mor C*
shows *NatTransP* (*YFtorNT′ C f*)
proof(*auto simp only*: *NatTransP-def*)
have *Fd*: *Ftor* (*NTDom* (*YFtorNT′ C f*)) : (*Op C*) ⟶ *SET* **using** *assms*
by (*simp add*: *HomFtorContraFtor Category.Cdom YFtorNT′-def*)

have *Fc*: *Ftor* (*NTCod* (*YFtorNT′ C f*)) : (*Op C*) ⟶ *SET* **using** *assms*
by (*simp add*: *HomFtorContraFtor Category.Ccod YFtorNT′-def*)
show *Functor* (*NTDom* (*YFtorNT′ C f*)) **using** *Fd* **by** *auto*
show *Functor* (*NTCod* (*YFtorNT′ C f*)) **using** *Fc* **by** *auto*
show *NTCatDom* (*YFtorNT′ C f*) = *CatDom* (*NTCod* (*YFtorNT′ C f*))
by(*simp add*: *YFtorNT′-def NTCatDom-def HomFtorContraDom*)
show *NTCatCod* (*YFtorNT′ C f*) = *CatCod* (*NTDom* (*YFtorNT′ C f*))
by(*simp add*: *YFtorNT′-def NTCatCod-def HomFtorContraCod*)
{
fix *X* **assume** *a*: *X* ∈ *Obj* (*NTCatDom* (*YFtorNT′ C f*))
show (*YFtorNT′ C f*) $$ *X maps*$_{NTCatCod\ (YFtorNT'\ C\ f)}$ (*NTDom* (*YFtorNT′ C f*) @@ *X*) *to* (*NTCod* (*YFtorNT′ C f*) @@ *X*)
proof−
have *Obj*: *X* ∈ *Obj C* **using** *a* **by** (*simp add*: *NTCatDom-def YFtorNT′-def HomFtorContraDom OppositeCategory-def*)
have *H1*: (*Hom*$_C$[−,*dom*$_C$ *f*]) @@ *X* = *Hom*$_C$ *X dom*$_C$ *f* **using** *assms Obj* **by**(*simp add*: *HomFtorOpObj Category.Cdom*)
have *H2*: (*Hom*$_C$[−,*cod*$_C$ *f*]) @@ *X* = *Hom*$_C$ *X cod*$_C$ *f* **using** *assms Obj* **by**(*simp add*: *HomFtorOpObj Category.Ccod*)
have *Hom*$_C$[*X*,*f*] *maps*$_{SET}$ (*Hom*$_C$ *X dom*$_C$ *f*) *to* (*Hom*$_C$ *X cod*$_C$ *f*) **using** *assms Obj* **by** (*simp add*: *HomFtorMapsTo*)
thus *?thesis* **using** *H1 H2* **by**(*simp add*: *YFtorNT′-def NTCatCod-def NTCatDom-def HomFtorContraCod*)
qed
}
{
fix *g X Y* **assume** *a*: *g maps*$_{NTCatDom\ (YFtorNT'\ C\ f)}$ *X to Y*
show ((*NTDom* (*YFtorNT′ C f*)) ## *g*) ;;$_{NTCatCod\ (YFtorNT'\ C\ f)}$ (*YFtorNT′ C f* $$ *Y*) =
((*YFtorNT′ C f*) $$ *X*) ;;$_{NTCatCod\ (YFtorNT'\ C\ f)}$ (*NTCod* (*YFtorNT′ C f*) ## *g*)
proof−
have *M1*: *g maps*$_{Op\ C}$ *X to Y* **using** *a* **by** (*auto simp add*: *NTCatDom-def YFtorNT′-def HomFtorContraDom*)
have *D1*: *dom*$_C$ *g* = *Y* **and** *C1*: *cod*$_C$ *g* = *X* **using** *M1* **by** (*auto simp add*: *OppositeCategory-def*)
have *morf*: *f* ∈ *Mor C* **and** *morg*: *g* ∈ *Mor C* **using** *assms M1* **by** (*auto simp add*: *OppositeCategory-def*)
have *H1*: (*HomC*$_C$[*g*,*dom*$_C$ *f*]) = (*Hom*$_C$[−,*dom*$_C$ *f*]) ## *g*
and *H2*: (*HomC*$_C$[*g*,*cod*$_C$ *f*]) = (*Hom*$_C$[−,*cod*$_C$ *f*]) ## *g* **using** *M1*
by (*auto simp add*: *HomFtorContra-def HomFtorContra′-def MakeFtor-def*)
have (*HomC*$_C$[*g*,*dom*$_C$ *f*]) ;;$_{SET}$ (*Hom*$_C$[*dom*$_C$ *g*,*f*]) = (*Hom*$_C$[*cod*$_C$ *g*,*f*]) ;;$_{SET}$ (*HomC*$_C$[*g*,*cod*$_C$ *f*]) **using** *assms morf morg*
by (*simp add*: *HomCHom*)
hence ((*Hom*$_C$[−,*dom*$_C$ *f*]) ## *g*) ;;$_{SET}$ (*Hom*$_C$[*Y*,*f*]) = (*Hom*$_C$[*X*,*f*]) ;;$_{SET}$ ((*Hom*$_C$[−,*cod*$_C$ *f*]) ## *g*)
using *H1 H2 D1 C1* **by** *simp*
thus *?thesis* **by** (*simp add*: *YFtorNT′-def NTCatCod-def HomFtorContraCod*)

qed
}
qed

lemma *YFtorNTNatTrans*:
assumes *LSCategory C* **and** $f \in Mor\ C$
shows *NatTrans* (*YFtorNT C f*)
by (*simp add*: *assms YFtorNTNatTrans' YFtorNT-def MakeNT*)

lemma *YFtorNTMor*:
assumes *LSCategory C* **and** $f \in Mor\ C$
shows *YFtorNT C f* $\in Mor$ (*CatExp* (*Op C*) *SET*)
proof(*auto simp add*: *CatExp-def CatExp'-def MakeCatMor*)
have $f \in Mor\ C$ **using** *assms* **by** *auto*
thus *NatTrans* (*YFtorNT C f*) **using** *assms* **by** (*simp add*: *YFtorNTNatTrans*)
show *NTCatDom* (*YFtorNT C f*) = *Op C* **by** (*simp add*: *YFtorNTCatDom*)
show *NTCatCod* (*YFtorNT C f*) = *SET* **by** (*simp add*: *YFtorNTCatCod*)
qed

lemma *YFtorNtMapsTo*:
assumes *LSCategory C* **and** $f \in Mor\ C$
shows *YFtorNT C f maps*$_{CatExp\ (Op\ C)\ SET}$ (*Hom*$_C$[−,*dom*$_C$ *f*]) *to* (*Hom*$_C$[−,*cod*$_C$ *f*])
proof(*rule MapsToI*)
have $f \in Mor\ C$ **using** *assms* **by** *auto*
thus *1*: *YFtorNT C f* $\in$ *mor*$_{CatExp\ (Op\ C)\ SET}$ **using** *assms* **by** (*simp add*: *YFtorNTMor*)
show *dom*$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C f* = *Hom*$_C$[−,*dom*$_C$ *f*] **using** *1* **by**(*simp add*:*CatExpDom YFtorNT-defs*)
show *cod*$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C f* = *Hom*$_C$[−,*cod*$_C$ *f*] **using** *1* **by**(*simp add*:*CatExpCod YFtorNT-defs*)
qed

lemma *YFtorNTCompDef*:
assumes *LSCategory C* **and** $f \approx>_C g$
shows *YFtorNT C f* $\approx>_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*
proof(*rule CompDefinedI*)
have $f \in Mor\ C$ **and** $g \in Mor\ C$ **using** *assms* **by** *auto*
hence *1*: *YFtorNT C f maps*$_{CatExp\ (Op\ C)\ SET}$ (*Hom*$_C$[−,*dom*$_C$ *f*]) *to* (*Hom*$_C$[−,*cod*$_C$ *f*])
and *2*: *YFtorNT C g maps*$_{CatExp\ (Op\ C)\ SET}$ (*Hom*$_C$[−,*dom*$_C$ *g*]) *to* (*Hom*$_C$[−,*cod*$_C$ *g*])
using *assms* **by** (*simp add*: *YFtorNtMapsTo*)+
thus *YFtorNT C f* $\in$ *mor*$_{CatExp\ (Op\ C)\ SET}$
and *YFtorNT C g* $\in$ *mor*$_{CatExp\ (Op\ C)\ SET}$ **by** *auto*
have *cod*$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C f* = (*Hom*$_C$[−,*cod*$_C$ *f*]) **using** *1* **by** *auto*
moreover have *dom*$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g* = (*Hom*$_C$[−,*dom*$_C$ *g*])

using *2* **by** *auto*
moreover have $cod_C\ f = dom_C\ g$ **using** *assms* **by** *auto*
ultimately show $cod_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ f = dom_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ g$ **by** *simp*
qed

lemma *PreSheafCat*: *LSCategory C* $\Longrightarrow$ *Category* (*CatExp* (*Op C*) *SET*)
by(*simp add*: *YFtor′-def OpCatCat SETCategory CatExpCat*)

lemma *YFtor′Obj1*:
assumes *X* $\in$ *Obj* (*CatDom* (*YFtor′ C*)) **and** *LSCategory C*
shows (*YFtor′ C*) ## (*Id* (*CatDom* (*YFtor′ C*)) *X*) = *Id* (*CatCod* (*YFtor′ C*)) (Hom_C [−,*X*])
proof(*simp add*: *YFtor′-def*, *rule NatTransExt*)
have *Obj*: *X* $\in$ *Obj C* **using** *assms* **by** (*simp add*: *YFtor′-def*)
have *HomObj*: (Hom_C[−,*X*]) $\in$ *Obj* (*CatExp* (*Op C*) *SET*) **using** *assms Obj* **by**(*simp add*: *CatExp-defs HomFtorContraFtor*)
hence *Id*: *Id* (*CatExp* (*Op C*) *SET*) (Hom_C[−,*X*]) $\in$ *Mor* (*CatExp* (*Op C*) *SET*) **using** *assms*
by (*simp add*: *PreSheafCat Category.CatIdInMor*)
have *CAT*: *Category*(*CatExp* (*Op C*) *SET*) **using** *assms* **by** (*simp add*: *PreSheaf-Cat*)
have *HomObj*: (Hom_C[−,*X*]) $\in$ *Obj* (*CatExp* (*Op C*) *SET*) **using** *assms Obj*
by(*simp add*: *CatExp-defs HomFtorContraFtor*)
show *NatTrans* (*YFtorNT C* (*Id C X*))
proof(*rule YFtorNTNatTrans*)
show *LSCategory C* **using** *assms*(*2*) **.**
show *Id C X* $\in$ *Mor C* **using** *assms Obj* **by** (*simp add*: *Category.CatIdInMor*)
qed
show *NatTrans*(*Id* (*CatExp* (*Op C*) *SET*) (Hom_C[−,*X*])) **using** *Id* **by** (*simp add*: *CatExp-defs*)
show *NTDom* (*YFtorNT C* (*Id C X*)) = *NTDom* (*Id* (*CatExp* (*Op C*) *SET*) (Hom_C[−,*X*]))
proof(*simp add*: *YFtorNT-defs*)
have $Hom_C[-,dom_C\ (Id\ C\ X)] = Hom_C[-,X]$ **using** *assms Obj* **by** (*simp add*: *Category.CatIdDomCod*)
also have ... = $dom_{CatExp\ (Op\ C)\ SET}$ (*Id* (*CatExp* (*Op C*) *SET*) (Hom_C[−,*X*])) **using** *CAT HomObj*
by (*simp add*: *Category.CatIdDomCod*)
also have ... = *NTDom* (*Id* (*CatExp* (*Op C*) *SET*) (Hom_C[−,*X*])) **using** *Id* **by** (*simp add*: *CatExpDom*)
finally show $Hom_C[-,dom_C\ (Id\ C\ X)] = NTDom\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-,X]))$ **.**
qed
show *NTCod* (*YFtorNT C* (*Id C X*)) = *NTCod* (*Id* (*CatExp* (*Op C*) *SET*) (Hom_C[−,*X*]))
proof(*simp add*: *YFtorNT-defs*)
have $Hom_C[-,cod_C\ (Id\ C\ X)] = Hom_C[-,X]$ **using** *assms Obj* **by** (*simp add*: *Category.CatIdDomCod*)

also have ... = $cod_{CatExp\ (Op\ C)\ SET}\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-,X]))$
using *CAT HomObj*
by (*simp add*: *Category.CatIdDomCod*)
also have ... = *NTCod* (*Id* (*CatExp* (*Op C*) *SET*) ($Hom_C[-,X]$)) **using** *Id*
by (*simp add*: *CatExpCod*)
finally show $Hom_C[-,cod_C\ (Id\ C\ X)] = NTCod\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-,X]))$ **.**
qed
{
fix *Y* **assume** *a*: $Y \in Obj\ (NTCatDom\ (YFtorNT\ C\ (Id\ C\ X)))$
show (*YFtorNT C* (*Id C X*)) $$ *Y* = (*Id* (*CatExp* (*Op C*) *SET*) ($Hom_C[-,X]$)) $$ *Y*
proof−
have *CD*: *CatDom* ($Hom_C[-,X]$) = *Op C* **by** (*simp add*: *HomFtorContraDom*)
have *CC*: *CatCod* ($Hom_C[-,X]$) = *SET* **by** (*simp add*: *HomFtorContraCod*)
have *ObjY*: $Y \in Obj\ C$ **and** *ObjYOp*: $Y \in Obj\ (Op\ C)$ **using** *a* **by**(*simp add*: *YFtorNTCatDom OppositeCategory-def*)+
have (*YFtorNT C* (*Id C X*)) $$ *Y* = ($Hom_C[Y,(Id\ C\ X)]$) **using** *a* **by** (*simp add*: *YFtorNTApp1*)
also have ... = $id_{SET}\ (Hom_C\ Y\ X)$ **using** *Obj ObjY assms* **by** (*simp add*: *HomFtorId*)
also have ... = $id_{SET}\ ((Hom_C[-,X])\ @@\ Y)$ **using** *Obj ObjY assms* **by** (*simp add*: *HomFtorOpObj*)
also have ... = (*IdNatTrans* ($Hom_C[-,X]$)) $$ *Y* **using** *CD CC ObjYOp* **by** (*simp add*: *IdNatTrans-map*)
also have ... = (*Id* (*CatExp* (*Op C*) *SET*) ($Hom_C[-,X]$)) $$ *Y* **using** *HomObj* **by** (*simp add*: *CatExpId*)
finally show *?thesis* **.**
qed
}
qed

lemma *YFtorPreFtor*:
assumes *LSCategory C*
shows *PreFunctor* (*YFtor′ C*)
proof(*auto simp only*: *PreFunctor-def*)
have *CAT*: *Category*(*CatExp* (*Op C*) *SET*) **using** *assms* **by** (*simp add*: *PreSheaf-Cat*)
{
fix *f g* **assume** *a*: $f \approx>_{CatDom\ (YFtor'\ C)}\ g$
show (*YFtor′ C*) ## ($f\ ;;_{CatDom\ (YFtor'\ C)}\ g$) = ((*YFtor′ C*) ## *f*) $;;_{CatCod\ (YFtor'\ C)}$ ((*YFtor′ C*) ## *g*)
proof(*simp add*: *YFtor′-def*, *rule NatTransExt*)
have *CD*: $f \approx>_C g$ **using** *a* **by** (*simp add*: *YFtor′-def*)
have *CD2*: $YFtorNT\ C\ f \approx>_{CatExp\ (Op\ C)\ SET} YFtorNT\ C\ g$ **using** *CD assms* **by** (*simp add*: *YFtorNTCompDef*)
have *Mor1*: $YFtorNT\ C\ f\ ;;_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ g \in Mor\ (CatExp\ (Op\ C)\ SET)$ **using** *CAT CD2*

by (*simp add*: *Category.MapsToMorDomCod*)
show *NatTrans* (*YFtorNT C* (*f* ;;$_C$ *g*)) **using** *assms* **by** (*simp add*: *Category.MapsToMorDomCod CD YFtorNTNatTrans*)
show *NatTrans* (*YFtorNT C f* ;;$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*) **using** *Mor1* **by** (*simp add*: *CatExpMorNT*)
show *NTDom* (*YFtorNT C* (*f* ;;$_C$ *g*)) = *NTDom* (*YFtorNT C f* ;;$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*)
proof−
have *1*: *YFtorNT C f* ∈ *mor*$_{CatExp\ (Op\ C)\ SET}$ **using** *CD2* **by** *auto*
have *NTDom* (*YFtorNT C* (*f* ;;$_C$ *g*)) = *Hom*$_C$[−,*dom*$_C$ (*f* ;;$_C$ *g*)] **by** (*simp add*: *YFtorNT-defs*)
also have ... = *Hom*$_C$[−,*dom*$_C$ *f*] **using** *CD assms* **by** (*simp add*: *Category.MapsToMorDomCod*)
also have ... = *NTDom* (*YFtorNT C f*) **by** (*simp add*: *YFtorNT-defs*)
also have ... = *dom*$_{CatExp\ (Op\ C)\ SET}$ (*YFtorNT C f*) **using** *1* **by** (*simp add*: *CatExpDom*)
also have ... = *dom*$_{CatExp\ (Op\ C)\ SET}$ (*YFtorNT C f* ;;$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*) **using** *CD2 CAT*
by (*simp add*: *Category.MapsToMorDomCod*)
finally show *?thesis* **using** *Mor1* **by** (*simp add*: *CatExpDom*)
qed
show *NTCod* (*YFtorNT C* (*f* ;;$_C$ *g*)) = *NTCod* (*YFtorNT C f* ;;$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*)
proof−
have *1*: *YFtorNT C g* ∈ *mor*$_{CatExp\ (Op\ C)\ SET}$ **using** *CD2* **by** *auto*
have *NTCod* (*YFtorNT C* (*f* ;;$_C$ *g*)) = *Hom*$_C$[−,*cod*$_C$ (*f* ;;$_C$ *g*)] **by** (*simp add*: *YFtorNT-defs*)
also have ... = *Hom*$_C$[−,*cod*$_C$ *g*] **using** *CD assms* **by** (*simp add*: *Category.MapsToMorDomCod*)
also have ... = *NTCod* (*YFtorNT C g*) **by** (*simp add*: *YFtorNT-defs*)
also have ... = *cod*$_{CatExp\ (Op\ C)\ SET}$ (*YFtorNT C g*) **using** *1* **by** (*simp add*: *CatExpCod*)
also have ... = *cod*$_{CatExp\ (Op\ C)\ SET}$ (*YFtorNT C f* ;;$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*) **using** *CD2 CAT*
by (*simp add*: *Category.MapsToMorDomCod*)
finally show *?thesis* **using** *Mor1* **by** (*simp add*: *CatExpCod*)
qed
{
fix *X* **assume** *a*: *X* ∈ *Obj* (*NTCatDom* (*YFtorNT C* (*f* ;;$_C$ *g*)))
show *YFtorNT C* (*f* ;;$_C$ *g*) $$ *X* = (*YFtorNT C f* ;;$_{CatExp\ (Op\ C)\ SET}$ *YFtorNT C g*) $$ *X*
proof−
have *Obj*: *X* ∈ *Obj C* **and** *ObjOp*: *X* ∈ *Obj* (*Op C*) **using** *a* **by** (*simp add*: *YFtorNTCatDom OppositeCategory-def*)+
have *App1*: (*Hom*$_C$[*X*,*f*]) = (*YFtorNT C f*) $$ *X*
and *App2*: (*Hom*$_C$[*X*,*g*]) = (*YFtorNT C g*) $$ *X* **using** *a* **by** (*simp add*: *YFtorNTApp1 YFtorNTCatDom*)+
have (*YFtorNT C* (*f* ;;$_C$ *g*)) $$ *X* = (*Hom*$_C$[*X*,(*f* ;;$_C$ *g*)]) **using** *a* **by**

(*simp add*: *YFtorNTApp1*)
also have ... = ($Hom_C[X,f]$) ;;$_{SET}$ ($Hom_C[X,g]$) **using** *CD assms Obj*
by (*simp add*: *HomFtorDist*)
also have ... = ((*YFtorNT C f*) $$ *X*) ;;$_{SET}$ ((*YFtorNT C g*) $$ *X*)
using *App1 App2* **by** *simp*
finally show *?thesis* **using** *ObjOp CD2* **by** (*simp add*: *CatExpDist*)
qed
}
qed
}
{
fix *X* **assume** *a*: *X* ∈ *Obj* (*CatDom* (*YFtor′ C*))
show ∃ *Y* ∈ *Obj* (*CatCod* (*YFtor′ C*)) . *YFtor′ C* ## (*Id* (*CatDom* (*YFtor′*
C)) *X*) = *Id* (*CatCod* (*YFtor′ C*)) *Y*
proof(*rule-tac x*=Hom_C [−,*X*] **in** *Set.rev-bexI*)
have *X* ∈ *Obj C* **using** *a* **by**(*simp add*: *YFtor′-def*)
thus Hom_C [−,*X*] ∈ *Obj* (*CatCod* (*YFtor′ C*)) **using** *assms* **by**(*simp add*:
YFtor′-def CatExp-defs HomFtorContraFtor)
show (*YFtor′ C*) ## (*Id* (*CatDom* (*YFtor′ C*)) *X*) = *Id* (*CatCod* (*YFtor′*
C)) (Hom_C [−,*X*]) **using** *a assms*
by (*simp add*: *YFtor′Obj1*)
qed
}
show *Category* (*CatDom* (*YFtor′ C*)) **using** *assms* **by** (*simp add*: *YFtor′-def*)
show *Category* (*CatCod* (*YFtor′ C*)) **using** *CAT* **by** (*simp add*: *YFtor′-def*)
qed

lemma *YFtor′Obj*:
assumes *X* ∈ *Obj* (*CatDom* (*YFtor′ C*))
and *LSCategory C*
shows (*YFtor′ C*) @@ *X* = Hom_C [−,*X*]
proof(*rule PreFunctor.FmToFo*, *simp-all add*: *assms YFtor′Obj1 YFtorPreFtor*)
have *X* ∈ *Obj C* **using** *assms* **by**(*simp add*: *YFtor′-def*)
thus Hom_C [−,*X*] ∈ *Obj* (*CatCod* (*YFtor′ C*)) **using** *assms* **by**(*simp add*:
YFtor′-def CatExp-defs HomFtorContraFtor)
qed

lemma *YFtorFtor′*:
assumes *LSCategory C*
shows *FunctorM* (*YFtor′ C*)
proof(*auto simp only*: *FunctorM-def*)
show *PreFunctor* (*YFtor′ C*) **using** *assms* **by** (*rule YFtorPreFtor*)
show *FunctorM-axioms* (*YFtor′ C*)
proof(*auto simp add*:*FunctorM-axioms-def*)
{
fix *f X Y* **assume** *aa*: *f maps*$_{CatDom\ (YFtor'\ C)}$ *X to Y*
show *YFtor′ C* ## *f maps*$_{CatCod\ (YFtor'\ C)}$ *YFtor′ C* @@ *X to YFtor′ C*
@@ *Y*
proof−

```
      have Mor1: f maps_C X to Y using aa by (simp add: YFtor'-def)
    have Category (CatDom (YFtor' C)) using assms by (simp add: YFtor'-def)
      hence Obj1: X ∈ Obj (CatDom (YFtor' C)) and
            Obj2: Y ∈ Obj (CatDom (YFtor' C)) using aa assms by (simp add:
Category.MapsToObj)+
      have (YFtor' C ## f) = YFtorNT C f by (simp add: YFtor'-def)
      moreover have YFtor' C @@ X = Hom_C [−,X]
        and YFtor' C @@ Y = Hom_C [−,Y] using Obj1 Obj2 assms by (simp
add: YFtor'Obj)+
     moreover have CatCod (YFtor' C) = CatExp (Op C) SET by (simp add:
YFtor'-def)
      moreover have YFtorNT C f maps_CatExp (Op C) SET (Hom_C [−,X]) to
(Hom_C [−,Y])
        using assms Mor1 by (auto simp add: YFtorNtMapsTo)
      ultimately show ?thesis by simp
    qed
   }
  qed
qed

lemma YFtorFtor: assumes LSCategory C shows Ftor (YFtor C) : C ⟶
(CatExp (Op C) SET)
proof(auto simp only: functor-abbrev-def)
 show Functor (YFtor C) using assms by(simp add: MakeFtor YFtor-def YFtorFtor')
 show CatDom (YFtor C) = C and CatCod (YFtor C) = (CatExp (Op C) SET)

   using assms by(simp add: MakeFtor-def YFtor-def YFtor'-def)+
qed

lemma YFtorObj:
  assumes LSCategory C and X ∈ Obj C
  shows (YFtor C) @@ X = Hom_C [−,X]
proof−
  have CatDom (YFtor' C) = C by (simp add: YFtor'-def)
  moreover hence (YFtor' C) @@ X = Hom_C [−,X] using assms by(simp add:
YFtor'Obj)
   moreover have PreFunctor (YFtor' C) using assms by (simp add: YFtor-
PreFtor)
  ultimately show ?thesis using assms by (simp add: MakeFtorObj YFtor-def)
qed

lemma YFtorObj2:
  assumes LSCategory C and X ∈ Obj C and Y ∈ Obj C
  shows ((YFtor C) @@ Y) @@ X = Hom_C X Y
proof−
 have Hom_C X Y = ((Hom_C[−,Y]) @@ X) using assms by (simp add: HomFtorOpObj)
 also have ... = ((YFtor C @@ Y) @@ X) using assms by (simp add: YFtorObj)
 finally show ?thesis by simp
qed
```

lemma *YFtorMor*: $[\![LSCategory\ C\ ; f \in Mor\ C]\!] \Longrightarrow (YFtor\ C)\ \#\#\ f = YFtorNT\ C\ f$
by (*simp add*: *YFtor-defs MakeFtorMor*)

definition $YMap\ C\ X\ \eta \equiv (\eta\ \$\$\ X)\ |@|\ (m2z_C\ (id_C\ X))$
definition *YMapInv'* $C\ X\ F\ x \equiv (\!|$
$NTDom = ((YFtor\ C)\ @@\ X)$,
$NTCod = F$,
$NatTransMap = \lambda\ B\ .\ ZFfun\ (Hom_C\ B\ X)\ (F\ @@\ B)\ (\lambda\ f\ .\ (F\ \#\#\ (z2m_C\ f))\ |@|\ x)$
$|\!)$
definition *YMapInv* $C\ X\ F\ x \equiv MakeNT\ (YMapInv'\ C\ X\ F\ x)$

lemma *YMapInvApp*:
assumes $X \in Obj\ C$ **and** $B \in Obj\ C$ **and** $LSCategory\ C$
shows $(YMapInv\ C\ X\ F\ x)\ \$\$\ B = ZFfun\ (Hom_C\ B\ X)\ (F\ @@\ B)\ (\lambda\ f\ .\ (F\ \#\#\ (z2m_C\ f))\ |@|\ x)$
proof−
have $NTCatDom\ (MakeNT\ (YMapInv'\ C\ X\ F\ x)) = CatDom\ (NTDom\ (YMapInv'\ C\ X\ F\ x))$ **by** (*simp add*: *MakeNT-def NTCatDom-def*)
also have ... $= CatDom\ (Hom_C[-,X])$ **using** *assms* **by** (*simp add*: *YFtorObj YMapInv'-def*)
also have ... $= Op\ C$ **using** *assms HomFtorContraFtor*[*of C X*] **by** *auto*
finally have $NTCatDom\ (MakeNT\ (YMapInv'\ C\ X\ F\ x)) = Op\ C$ **.**
hence *1*: $B \in Obj\ (NTCatDom\ (MakeNT\ (YMapInv'\ C\ X\ F\ x)))$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)
have $(YMapInv\ C\ X\ F\ x)\ \$\$\ B = (MakeNT\ (YMapInv'\ C\ X\ F\ x))\ \$\$\ B$ **by** (*simp add*: *YMapInv-def*)
also have ... $= (YMapInv'\ C\ X\ F\ x)\ \$\$\ B$ **using** *1* **by**(*simp add*: *MakeNTApp*)
finally show *?thesis* **by** (*simp add*: *YMapInv'-def*)
qed

lemma *YMapImage*:
assumes $LSCategory\ C$ **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$
and $NT\ \eta : (YFtor\ C\ @@\ X) \Longrightarrow F$
shows $(YMap\ C\ X\ \eta)\ |\in|\ (F\ @@\ X)$
proof(*simp only*: *YMap-def*)
have $(YFtor\ C\ @@\ X) = (Hom_C[-,X])$ **using** *assms* **by** (*auto simp add*: *YFtorObj*)
moreover have $Ftor\ (Hom_C[-,X]) : (Op\ C) \longrightarrow SET$ **using** *assms* **by** (*simp add*: *HomFtorContraFtor*)
ultimately have $CatDom\ (YFtor\ C\ @@\ X) = Op\ C$ **by** *auto*
hence *Obj*: $X \in Obj\ (CatDom\ (YFtor\ C\ @@\ X))$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)
moreover have $CatCod\ F = SET$ **using** *assms* **by** *auto*
moreover have $\eta\ \$\$\ X\ maps_{CatCod\ F}\ ((YFtor\ C\ @@\ X)\ @@\ X)\ to\ (F\ @@\ X)$

using *assms Obj* **by** (*simp add*: *NatTransMapsTo*)
ultimately have η \$\$ *X maps*$_{SET}$ ((*YFtor C* @@ *X*) @@ *X*) *to* (*F* @@ *X*) **by** *simp*
moreover have (*m2z*$_C$ (*Id C X*)) |∈| ((*YFtor C* @@ *X*) @@ *X*)
proof−
have (*Id C X*) *maps*$_C$ *X to X* **using** *assms* **by** (*simp add*: *Category.Simps*)
moreover have ((*YFtor C* @@ *X*) @@ *X*) = *Hom*$_C$ *X X* **using** *assms* **by** (*simp add*: *YFtorObj2*)
ultimately show *?thesis* **using** *assms* **by** (*simp add*: *LSCategory.m2zz2m*)
qed
ultimately show ((*η* \$\$ *X*) |@| (*m2z*$_C$ (*Id C X*))) |∈| (*F* @@ *X*) **by** (*simp add*: *SETfunDomAppCod*)
qed

lemma *YMapInvNatTransP*:
assumes *LSCategory C* **and** *Ftor F* : (*Op C*) ⟶ *SET* **and** *xobj*: *X* ∈ *Obj C* **and** *xinF*: *x* |∈| (*F* @@ *X*)
shows *NatTransP* (*YMapInv*′ *C X F x*)
proof(*auto simp only*: *NatTransP-def*, *simp-all add*: *YMapInv*′*-def NTCatCod-def NTCatDom-def*)
have *yf*: (*YFtor C* @@ *X*) = *Hom*$_C$[−,*X*] **using** *assms* **by** (*simp add*: *YFtorObj*)
hence *hf*: *Ftor* (*YFtor C* @@ *X*) : (*Op C*) ⟶ *SET* **using** *assms* **by** (*simp add*: *HomFtorContraFtor*)
thus *Functor* (*YFtor C* @@ *X*) **by** *auto*
show *ftf*: *Functor F* **using** *assms* **by** *auto*
have *df*: *CatDom F* = *Op C* **and** *cf*: *CatCod F* = *SET* **using** *assms* **by** *auto*
have *dy*: *CatDom* ((*YFtor C*) @@ *X*) = *Op C* **and** *cy*: *CatCod* ((*YFtor C*) @@ *X*) = *SET* **using** *hf* **by** *auto*
show *CatDom* ((*YFtor C*) @@ *X*) = *CatDom F* **using** *df dy* **by** *simp*
show *CatCod F* = *CatCod* ((*YFtor C*) @@ *X*) **using** *cf cy* **by** *simp*
{
fix *Y* **assume** *yobja*: *Y* ∈ *Obj* (*CatDom* ((*YFtor C*) @@ *X*))
show *ZFfun* (*Hom*$_C$ *Y X*) (*F* @@ *Y*) (*λf*. (*F* ## (*z2m*$_C$ *f*)) |@| *x*) *maps*$_{CatCod\ F}$ ((*YFtor C* @@ *X*) @@ *Y*) *to* (*F* @@ *Y*)
proof(*simp add*: *cf*, *rule MapsToI*)
have *yobj*: *Y* ∈ *Obj C* **using** *yobja dy* **by** (*simp add*: *OppositeCategory-def*)
have *zffun*: *isZFfun* (*ZFfun* (*Hom*$_C$ *Y X*) (*F* @@ *Y*) (*λf*. (*F* ## *z2m*$_C$*f*) |@| *x*))
proof(*rule SETfun*, *rule allI*, *rule impI*)
{
fix *y* **assume** *yhom*: *y* |∈| (*Hom*$_C$ *Y X*) **show** (*F* ## (*z2m*$_C$ *y*)) |@| *x* |∈| (*F* @@ *Y*)
proof−
let *?f* = (*F* ## (*z2m*$_C$ *y*))
have (*z2m*$_C$ *y*) *maps*$_C$ *Y to X* **using** *yhom yobj assms* **by** (*simp add*: *LSCategory.z2mm2z*)
hence (*z2m*$_C$ *y*) *maps*$_{Op\ C}$ *X to Y* **by** (*simp add*: *MapsToOp*)
hence *?f maps*$_{SET}$ (*F* @@ *X*) *to* (*F* @@ *Y*) **using** *assms* **by** (*simp add*: *FunctorMapsTo*)

hence *isZFfun (?f)* **and** $|dom|$ *?f* = *F @@ X* **and** $|cod|$ *?f* = *F @@ Y* **by** (*simp add*: *SETmapsTo*)+
thus (*?f* $|@|$ *x*) $|\in|$ (*F @@ Y*) **using** *assms ZFfunDomAppCod*[*of ?f x*] **by** *simp*
qed
}
qed
show *ZFfun* (*Hom*$_C$ *Y X*) (*F @@ Y*) (λ*f*. (*F ## z2m*$_C$*f*) $|@|$ *x*) ∈ *mor*$_{SET}$ **using** *zffun*
by(*simp add*: *SETmor*)
show *cod*$_{SET}$ *ZFfun* (*Hom*$_C$ *Y X*) (*F @@ Y*) (λ*f*. (*F ## z2m*$_C$*f*) $|@|$ *x*) = *F @@ Y* **using** *zffun*
by(*simp add*: *SETcod*)
have (*Hom*$_C$ *Y X*) = (*YFtor C @@ X*) *@@ Y* **using** *assms yobj* **by** (*simp add*: *YFtorObj2*)
thus *dom*$_{SET}$ *ZFfun* (*Hom*$_C$ *Y X*) (*F @@ Y*) (λ*f*. (*F ## z2m*$_C$*f*) $|@|$ *x*) = (*YFtor C @@ X*) *@@ Y* **using** *zffun*
by(*simp add*: *SETdom*)
qed
}
{
fix *f Z Y* **assume** *fmaps*: *f maps*$_{CatDom}$ ((*YFtor C*) *@@ X*) *Z to Y*
have *fmapsa*: *f maps*$_{Op\ C}$ *Z to Y* **using** *fmaps dy* **by** *simp*
hence *fmapsb*: *f maps*$_C$ *Y to Z* **by** (*rule MapsToOpOp*)
hence *fmor*: *f* ∈ *Mor C* **and** *fdom*: *dom*$_C$ *f* = *Y* **and** *fcod*: *cod*$_C$ *f* = *Z* **by** (*auto simp add*: *OppositeCategory-def*)
hence *hc*: (*Hom*$_C$[−,*X*]) *##* *f* = (*HomC*$_C$[*f*,*X*]) **using** *assms* **by** (*simp add*: *HomContraMor*)
have *yobj*: *Y* ∈ *Obj C* **and** *zobj*: *Z* ∈ *Obj C* **using** *fmapsb assms* **by** (*simp add*: *Category.MapsToObj*)+
have *Ffmaps*: (*F ## f*) *maps*$_{SET}$ (*F @@ Z*) *to* (*F @@ Y*) **using** *assms fmapsa* **by** (*simp add*: *FunctorMapsTo*)
have *Fzmaps*: ⋀ *h A B* . ⟦*h* $|\in|$ (*Hom*$_C$ *A B*) ; *A* ∈ *Obj C* ; *B* ∈ *Obj C*⟧ ⟹ (*F ##* (*z2m*$_C$ *h*)) *maps*$_{SET}$ (*F @@ B*) *to* (*F @@ A*)
proof−
fix *h A B* **assume** *h*: *h* $|\in|$ (*Hom*$_C$ *A B*) **and** *oA*: *A* ∈ *Obj C* **and** *oB*: *B* ∈ *Obj C*
have (*z2m*$_C$ *h*) *maps*$_C$ *A to B* **using** *assms h oA oB* **by** (*simp add*: *LSCategory.z2mm2z*)
hence (*z2m*$_C$ *h*) *maps*$_{Op\ C}$ *B to A* **by** (*rule MapsToOp*)
thus (*F ##* (*z2m*$_C$ *h*)) *maps*$_{SET}$ (*F @@ B*) *to* (*F @@ A*) **using** *assms* **by** (*simp add*: *FunctorMapsTo*)
qed
have *hHomF*: ⋀*h*. *h* $|\in|$ (*Hom*$_C$ *Z X*) ⟹ (*F ##* (*z2m*$_C$ *h*)) $|@|$ *x* $|\in|$ (*F @@ Z*) **using** *xobj zobj xinF*
proof−
fix *h* **assume** *h*: *h* $|\in|$ (*Hom*$_C$ *Z X*)
have (*F ##* (*z2m*$_C$ *h*)) *maps*$_{SET}$ (*F @@ X*) *to* (*F @@ Z*) **using** *xobj zobj h* **by** (*simp add*: *Fzmaps*)

thus $(F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x\ |\in|\ (F \mathbin{@@} Z)$ **using** *assms* **by** (*simp add*: *SETfunDomAppCod*)
qed
have *Ff*: $F \mathbin{\#\#} f = ZFfun\ (F \mathbin{@@} Z)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} f)\ |@|\ h)$ **using** *Ffmaps* **by** (*simp add*: *SETZFfun*)
have *compdefa*: $ZFfun\ (Hom_C\ Z\ X)\ (Hom_C\ Y\ X)\ (\lambda h.\ m2z_C\ (f\ ;;_C\ (z2m_C\ h)))$ $\approx>_{SET}$
$ZFfun\ (Hom_C\ Y\ X)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x)$
proof(*rule CompDefinedI*, *simp-all add*: *SETmor*[*THEN sym*])
show $isZFfun\ (ZFfun\ (Hom_C\ Z\ X)\ (Hom_C\ Y\ X)\ (\lambda h.\ m2z_C\ (f\ ;;_C\ (z2m_C\ h))))$
proof(*rule SETfun*, *rule allI*, *rule impI*)
fix h **assume** h: $h\ |\in|\ (Hom_C\ Z\ X)$
have $(z2m_C\ h)\ maps_C\ Z\ to\ X$ **using** *assms h xobj zobj* **by** (*simp add*: *LSCategory.z2mm2z*)
hence $f\ ;;_C\ (z2m_C\ h)\ maps_C\ Y\ to\ X$ **using** *fmapsb assms*(*1*) **by** (*simp add*: *Category.Ccompt*)
thus $(m2z_C\ (f\ ;;_C\ (z2m_C\ h)))\ |\in|\ (Hom_C\ Y\ X)$ **using** *assms* **by** (*simp add*: *LSCategory.m2zz2m*)
qed
moreover show $isZFfun\ (ZFfun\ (Hom_C\ Y\ X)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x))$
proof(*rule SETfun*, *rule allI*, *rule impI*)
fix h **assume** h: $h\ |\in|\ (Hom_C\ Y\ X)$
have $(F \mathbin{\#\#} (z2m_C\ h))\ maps_{SET}\ (F \mathbin{@@} X)\ to\ (F \mathbin{@@} Y)$ **using** *xobj yobj h* **by** (*simp add*: *Fzmaps*)
thus $(F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x\ |\in|\ (F \mathbin{@@} Y)$ **using** *assms* **by** (*simp add*: *SETfunDomAppCod*)
qed
ultimately show $cod_{SET}(ZFfun\ (Hom_C\ Z\ X)\ (Hom_C\ Y\ X)\ (\lambda h.\ m2z_C\ (f\ ;;_C\ (z2m_C\ h)))) =$
$dom_{SET}(ZFfun\ (Hom_C\ Y\ X)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x))$ **by** (*simp add*: *SETcod SETdom*)
qed
have *compdefb*: $ZFfun\ (Hom_C\ Z\ X)\ (F \mathbin{@@} Z)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x)$ $\approx>_{SET}$
$ZFfun\ (F \mathbin{@@} Z)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} f)\ |@|\ h)$
proof(*rule CompDefinedI*, *simp-all add*: *SETmor*[*THEN sym*])
show $isZFfun\ (ZFfun\ (Hom_C\ Z\ X)\ (F \mathbin{@@} Z)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x))$ **using** *hHomF* **by** (*simp add*: *SETfun*)
moreover show $isZFfun\ (ZFfun\ (F \mathbin{@@} Z)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} f)\ |@|\ h))$
proof(*rule SETfun*, *rule allI*, *rule impI*)
fix h **assume** h: $h\ |\in|\ (F \mathbin{@@} Z)$
have $F \mathbin{\#\#} f\ maps_{SET}\ F \mathbin{@@} Z\ to\ F \mathbin{@@} Y$ **using** *Ffmaps* **.**
thus $(F \mathbin{\#\#} f)\ |@|\ h\ |\in|\ (F \mathbin{@@} Y)$ **using** h **by** (*simp add*: *SETfunDomAppCod*)
qed
ultimately show $cod_{SET}\ (ZFfun\ (Hom_C\ Z\ X)\ (F \mathbin{@@} Z)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h))\ |@|\ x)) =$

dom_{SET} ($ZFfun$ (F @@ Z) (F @@ Y) ($\lambda h.$ (F ## f) |@| h)) **by** (*simp add*: *SETcod SETdom*)
qed
have $ZFfun$ (Hom_C Z X) (Hom_C Y X) ($\lambda h.$ $m2z_C$ (f ;;$_C$ ($z2m_C$ h))) ;;$_{SET}$
$ZFfun$ (Hom_C Y X) (F @@ Y) ($\lambda h.$ (F ## ($z2m_C$ h)) |@| x) =
$ZFfun$ (Hom_C Z X) (Hom_C Y X) ($\lambda h.$ $m2z_C$ (f ;;$_C$ ($z2m_C$ h))) |o|
$ZFfun$ (Hom_C Y X) (F @@ Y) ($\lambda h.$ (F ## ($z2m_C$ h)) |@| x) **using** *Ff compdefa* **by** (*simp add*: *SETComp*)
also have ... = $ZFfun$ (Hom_C Z X) (F @@ Y) (($\lambda h.$ (F ## ($z2m_C$ h)) |@| x) o ($\lambda h.$ $m2z_C$ (f ;;$_C$ ($z2m_C$ h))))
proof(*rule ZFfunComp*, *rule allI*, *rule impI*)
{
fix h **assume** h: h |∈| (Hom_C Z X)
show ($m2z_C$ (f ;;$_C$ ($z2m_C$ h))) |∈| (Hom_C Y X)
proof−
have Z ∈ Obj C **using** *fmapsb assms* **by** (*simp add*: *Category.MapsToObj*)
hence ($z2m_C$ h) $maps_C$ Z *to* X **using** *assms* h **by** (*simp add*: *LSCategory.z2mm2z*)
hence f ;;$_C$ ($z2m_C$ h) $maps_C$ Y *to* X **using** *fmapsb assms*(*1*) **by** (*simp add*: *Category.Ccompt*)
thus *?thesis* **using** *assms* **by** (*simp add*: *LSCategory.m2zz2m*)
qed
}
qed
also have ... = $ZFfun$ (Hom_C Z X) (F @@ Y) (($\lambda h.$ (F ## f) |@| h) o ($\lambda h.$ (F ## ($z2m_C$ h)) |@| x))
proof(*rule ZFfun-ext*, *rule allI*, *rule impI*, *simp*)
{
fix h **assume** h: h |∈| (Hom_C Z X)
have *zObj*: Z ∈ Obj C **using** *fmapsb assms* **by** (*simp add*: *Category.MapsToObj*)
hence *hmaps*: ($z2m_C$ h) $maps_C$ Z *to* X **using** *assms* h **by** (*simp add*: *LSCategory.z2mm2z*)
hence ($z2m_C$ h) ∈ Mor C **and** dom_C ($z2m_C$ h) = cod_C f **using** *fcod* **by** *auto*
hence *CompDef-hf*: f ≈>$_C$ ($z2m_C$ h) **using** *fmor* **by** *auto*
hence *CompDef-hfOp*: ($z2m_C$ h) ≈>$_{Op\ C}$ f **by** (*simp add*: *CompDefOp*)
hence *CompDef-FhfOp*: (F ## ($z2m_C$ h)) ≈>$_{SET}$ (F ## f) **using** *assms* **by** (*simp add*: *FunctorCompDef*)
hence ($z2m_C$ h) $maps_{Op\ C}$ X *to* Z **using** *hmaps* **by** (*simp add*: *MapsToOp*)
hence (F ## ($z2m_C$ h)) $maps_{SET}$ (F @@ X) *to* (F @@ Z) **using** *assms* **by** (*simp add*: *FunctorMapsTo*)
hence *xin*: x |∈| |dom|(F ## ($z2m_C$ h)) **using** *assms* **by** (*simp add*: *SETmapsTo*)
have (f ;;$_C$ ($z2m_C$ h)) ∈ Mor C **using** *CompDef-hf assms* **by**(*simp add*: *Category.MapsToMorDomCod*)
hence (F ## ($z2m_C$ ($m2z_C$ (f ;;$_C$ ($z2m_C$ h))))) |@| x = (F ## (f ;;$_C$ ($z2m_C$ h))) |@| x
using *assms* **by** (*simp add*: *LSCategory.m2zz2mInv*)

also have ... = $(F \mathbin{\#\#} ((z2m_C\ h) \mathbin{;;_{Op\ C}} f)) \mathbin{|@|} x$ **by** (*simp add*: *Opposite-Category-def*)
also have ... = $((F \mathbin{\#\#} (z2m_C\ h)) \mathbin{;;_{SET}} (F \mathbin{\#\#} f)) \mathbin{|@|} x$ **using** *assms CompDef-hfOp* **by** (*simp add*: *FunctorComp*)
also have ... = $(F \mathbin{\#\#} f) \mathbin{|@|} ((F \mathbin{\#\#} (z2m_C\ h)) \mathbin{|@|} x)$ **using** *CompDef-FhfOp xin* **by**(*rule SETCompAt*)
finally show $(F \mathbin{\#\#} (z2m_C\ (m2z_C\ (f \mathbin{;;_C} (z2m_C\ h))))) \mathbin{|@|} x = (F \mathbin{\#\#} f) \mathbin{|@|} ((F \mathbin{\#\#} (z2m_C\ h)) \mathbin{|@|} x)$ **.**
}
qed
also have ... = $ZFfun\ (Hom_C\ Z\ X)\ (F \mathbin{@@} Z)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h)) \mathbin{|@|} x) \mathbin{|o|}$
$ZFfun\ (F \mathbin{@@} Z)\ (F \mathbin{@@} Y)\ (\lambda h.\ (F \mathbin{\#\#} f) \mathbin{|@|} h)$
by(*rule ZFfunComp*[*THEN sym*], *rule allI*, *rule impI*, *simp add*: *hHomF*)
also have ... = $ZFfun\ (Hom_C\ Z\ X)\ (F \mathbin{@@} Z)\ (\lambda h.\ (F \mathbin{\#\#} (z2m_C\ h)) \mathbin{|@|} x) \mathbin{;;_{SET}} (F \mathbin{\#\#} f)$
using *Ff compdefb* **by** (*simp add*: *SETComp*)
finally show $(((YFtor\ C) \mathbin{@@} X) \mathbin{\#\#} f) \mathbin{;;_{CatCod\ F}} ZFfun\ (Hom_C\ Y\ X)\ (F \mathbin{@@} Y)\ (\lambda f.\ (F \mathbin{\#\#} (z2m_C\ f)) \mathbin{|@|} x) =$
$ZFfun\ (Hom_C\ Z\ X)\ (F \mathbin{@@} Z)\ (\lambda f.\ (F \mathbin{\#\#} (z2m_C\ f)) \mathbin{|@|} x) \mathbin{;;_{CatCod\ F}} (F \mathbin{\#\#} f)$
by(*simp add*: *cf yf hc fdom fcod HomFtorMapContra-def*)
}
qed

lemma *YMapInvNatTrans*:
assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$ **and** $x \mathbin{|\in|} (F \mathbin{@@} X)$
shows $NatTrans\ (YMapInv\ C\ X\ F\ x)$
by (*simp add*: *assms YMapInv-def MakeNT YMapInvNatTransP*)

lemma *YMapInvImage*:
assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$
and $x \mathbin{|\in|} (F \mathbin{@@} X)$
shows $NT\ (YMapInv\ C\ X\ F\ x) : (YFtor\ C \mathbin{@@} X) \Longrightarrow F$
proof(*auto simp only*: *nt-abbrev-def*)
show $NatTrans\ (YMapInv\ C\ X\ F\ x)$ **using** *assms* **by** (*simp add*: *YMapInvNatTrans*)
show $NTDom\ (YMapInv\ C\ X\ F\ x) = YFtor\ C \mathbin{@@} X$ **by**(*simp add*: *YMapInv-def MakeNT-def YMapInv'-def*)
show $NTCod\ (YMapInv\ C\ X\ F\ x) = F$ **by**(*simp add*: *YMapInv-def MakeNT-def YMapInv'-def*)
qed

lemma *YMap1*:
assumes *LSCat*: *LSCategory C* **and** *Fftor*: $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** *XObj*: $X \in Obj\ C$
and *NT*: $NT\ \eta : (YFtor\ C \mathbin{@@} X) \Longrightarrow F$
shows $YMapInv\ C\ X\ F\ (YMap\ C\ X\ \eta) = \eta$

proof(*rule NatTransExt*)
have (*YMap C X η*) |∈| (*F @@ X*) **using** *assms* **by** (*simp add*: *YMapImage*)
hence *1*: *NT* (*YMapInv C X F* (*YMap C X η*)) : (*YFtor C @@ X*) ⟹ *F* **using** *assms* **by** (*simp add*: *YMapInvImage*)
thus *NatTrans* (*YMapInv C X F* (*YMap C X η*)) **by** *auto*
show *NatTrans η* **using** *assms* **by** *auto*
have *NTDYI*: *NTDom* (*YMapInv C X F* (*YMap C X η*)) = (*YFtor C @@ X*) **using** *1* **by** *auto*
moreover have *NTDeta*: *NTDom η* = (*YFtor C @@ X*) **using** *assms* **by** *auto*
ultimately show *NTDom* (*YMapInv C X F* (*YMap C X η*)) = *NTDom η* **by** *simp*
have *NTCod* (*YMapInv C X F* (*YMap C X η*)) = *F* **using** *1* **by** *auto*
moreover have *NTCeta*: *NTCod η* = *F* **using** *assms* **by** *auto*
ultimately show *NTCod* (*YMapInv C X F* (*YMap C X η*)) = *NTCod η* **by** *simp*
{
fix *Y* **assume** *Yobja*: *Y* ∈ *Obj* (*NTCatDom* (*YMapInv C X F* (*YMap C X η*)))
have *CCF*: *CatCod F* = *SET* **using** *assms* **by** *auto*
have *Ftor* (*Hom*$_C$[−,*X*]) : (*Op C*) ⟶ *SET* **using** *LSCat XObj* **by** (*simp add*: *HomFtorContraFtor*)
hence *CDH*: *CatDom* (*Hom*$_C$[−,*X*]) = *Op C* **by** *auto*
hence *CDYF*: *CatDom* (*YFtor C @@ X*) = *Op C* **using** *XObj LSCat* **by** (*auto simp add*: *YFtorObj*)
hence *NTCatDom* (*YMapInv C X F* (*YMap C X η*)) = *Op C* **using** *LSCat XObj NTDYI CDH* **by** (*simp add*: *NTCatDom-def*)
hence *YObjOp*: *Y* ∈ *Obj* (*Op C*) **using** *Yobja* **by** *simp*
hence *YObj*: *Y* ∈ *Obj C* **and** *XObjOp*: *X* ∈ *Obj* (*Op C*) **using** *XObj* **by** (*simp add*: *OppositeCategory-def*)+
have *yinv-mapsTo*: ((*YMapInv C X F* (*YMap C X η*)) $$ *Y*) *maps*$_{SET}$ (*Hom*$_C$ *Y X*) *to* (*F @@ Y*)
proof−
have ((*YMapInv C X F* (*YMap C X η*)) $$ *Y*) *maps*$_{SET}$ ((*YFtor C @@ X*) @@ *Y*) *to* (*F @@ Y*)
using *1 CCF CDYF YObjOp NatTransMapsTo*[*of* (*YMapInv C X F* (*YMap C X η*)) (*YFtor C @@ X*) *F Y*] **by** *simp*
thus *?thesis* **using** *LSCat XObj YObj* **by** (*simp add*: *YFtorObj2*)
qed
have *eta-mapsTo*: (*η* $$ *Y*) *maps*$_{SET}$ (*Hom*$_C$ *Y X*) *to* (*F @@ Y*)
proof−
have (*η* $$ *Y*) *maps*$_{SET}$ ((*YFtor C @@ X*) @@ *Y*) *to* (*F @@ Y*)
using *NT CDYF CCF YObjOp NatTransMapsTo*[*of η* (*YFtor C @@ X*) *F Y*] **by** *simp*
thus *?thesis* **using** *LSCat XObj YObj* **by** (*simp add*: *YFtorObj2*)
qed
show (*YMapInv C X F* (*YMap C X η*)) $$ *Y* = *η* $$ *Y*
proof(*rule ZFfunExt*)
show |*dom*|(*YMapInv C X F* (*YMap C X η*) $$ *Y*) = |*dom*|(*η* $$ *Y*)
using *yinv-mapsTo eta-mapsTo* **by** (*simp add*: *SETmapsTo*)

show $|cod|(YMapInv\ C\ X\ F\ (YMap\ C\ X\ \eta)\ \$\$\ Y) = |cod|(\eta\ \$\$\ Y)$
using *yinv-mapsTo eta-mapsTo* **by** (*simp add*: *SETmapsTo*)
show $isZFfun\ (YMapInv\ C\ X\ F\ (YMap\ C\ X\ \eta)\ \$\$\ Y)$
using *yinv-mapsTo* **by** (*simp add*: *SETmapsTo*)
show $isZFfun\ (\eta\ \$\$\ Y)$
using *eta-mapsTo* **by** (*simp add*: *SETmapsTo*)
{
fix f **assume** $fdomYinv$: $f\ |\in|\ |dom|(YMapInv\ C\ X\ F\ (YMap\ C\ X\ \eta)\ \$\$\ Y)$
have $fHom$: $f\ |\in|\ (Hom_C\ Y\ X)$ **using** *yinv-mapsTo fdomYinv* **by** (*simp add*: *SETmapsTo*)
hence $fMapsTo$: $(z2m_C\ f)\ maps_C\ Y\ to\ X$ **using** *assms YObj* **by** (*simp add*: *LSCategory.z2mm2z*)
hence $fCod$: $(cod_C\ (z2m_C\ f)) = X$ **and** $fDom$: $(dom_C\ (z2m_C\ f)) = Y$ **and** $fMor$: $(z2m_C\ f) \in Mor\ C$ **by** *auto*
have $(YMapInv\ C\ X\ F\ (YMap\ C\ X\ \eta)\ \$\$\ Y)\ |@|\ f =$
$(F\ \#\#\ (z2m_C\ f))\ |@|\ ((\eta\ \$\$\ X)\ |@|\ (m2z_C\ (Id\ C\ X)))$
using *fHom assms YObj* **by** (*simp add*: *ZFfunApp YMapInvApp YMap-def*)
also have $... = ((\eta\ \$\$\ X)\ ;;_{SET}\ (F\ \#\#\ (z2m_C\ f)))\ |@|\ (m2z_C\ (Id\ C\ X))$
proof−
have *aa*: $(\eta\ \$\$\ X)\ maps_{SET}\ ((YFtor\ C\ @@\ X)\ @@\ X)\ to\ (F\ @@\ X)$
using *NT CDYF CCF YObjOp XObjOp NatTransMapsTo*[*of* $\eta\ (YFtor\ C\ @@\ X)\ F\ X$] **by** *simp*
have *bb*: $(F\ \#\#\ (z2m_C\ f))\ maps_{SET}\ (F\ @@\ X)\ to\ (F\ @@\ Y)$
using *fMapsTo Fftor* **by** (*simp add*: *MapsToOp FunctorMapsTo*)
have $(\eta\ \$\$\ X) \approx>_{SET}\ (F\ \#\#\ (z2m_C\ f))$ **using** *aa bb* **by** (*simp add*: *MapsToCompDef*)
moreover have $(m2z_C\ (Id\ C\ X))\ |\in|\ |dom|\ (\eta\ \$\$\ X)$ **using** *assms aa*
by (*simp add*: *SETmapsTo YFtorObj2 Category.Cidm LSCategory.m2zz2m*)
ultimately show *?thesis* **by** (*simp add*: *SETCompAt*)
qed
also have $... = ((HomC_C[z2m_C\ f,X])\ ;;_{SET}\ (\eta\ \$\$\ Y))\ |@|\ (m2z_C\ (Id\ C\ X))$
proof−
have *NTDom* $\eta = (Hom_C[-,X])$ **using** *NTDeta assms* **by** (*simp add*: *YFtorObj*)
moreover hence *NTCatDom* $\eta = Op\ C$ **by** (*simp add*: *NTCatDom-def HomFtorContraDom*)
moreover have *NTCatCod* $\eta = SET$ **using** *assms* **by** (*auto simp add*: *NTCatCod-def*)
moreover have *NatTrans* η **and** *NTCod* $\eta = F$ **using** *assms* **by** *auto*
moreover have $(z2m_C\ f)\ maps_{Op\ C}\ X\ to\ Y$
using *fMapsTo MapsToOp*[**where** *?f* $= (z2m_C\ f)$ **and** *?X* $= Y$ **and** *?Y* $= X$ **and** *?C* $= C$] **by** *simp*
ultimately have $(\eta\ \$\$\ X)\ ;;_{SET}\ (F\ \#\#\ (z2m_C\ f)) = ((Hom_C[-,X])\ \#\#\ (z2m_C\ f))\ ;;_{SET}\ (\eta\ \$\$\ Y)$
using *NatTransP.NatTrans*[*of* $\eta\ (z2m_C\ f)\ X\ Y$] **by** *simp*
moreover have $((Hom_C[-,X])\ \#\#\ (z2m_C\ f)) = (HomC_C[(z2m_C\ f),X])$ **using** *assms fMor* **by** (*simp add*: *HomContraMor*)
ultimately show *?thesis* **by** *simp*

qed
also have ... = (η \$\$ Y) |@| (($HomC_C[z2m_C\ f,X]$) |@| ($m2z_C$ (Id C X)))
proof−
have ($HomC_C[z2m_C\ f,X]$) $\approx>_{SET}$ (η \$\$ Y)
using *fCod fDom XObj LSCat fMor HomFtorContraMapsTo*[*of* C X $z2m_C\ f$] *eta-mapsTo* **by** (*simp add*: *MapsToCompDef*)
moreover have |*dom*| ($HomC_C[z2m_C\ f,X]$) = (Hom_C (cod_C ($z2m_C\ f$)) X)
by (*simp add*: *ZFfunDom HomFtorMapContra-def*)
moreover have ($m2z_C$ (Id C X)) |∈| (Hom_C (cod_C ($z2m_C\ f$)) X)
using *assms fCod* **by** (*simp add*: *Category.Cidm LSCategory.m2zz2m*)
ultimately show *?thesis* **by** (*simp add*: *SETCompAt*)
qed
also have ... = (η \$\$ Y) |@| ($m2z_C$ (($z2m_C\ f$) ;;$_C$ ($z2m_C$ ($m2z_C$ (Id C X)))))
proof−
have (Id C X) $maps_C$ (cod_C ($z2m_C\ f$)) *to* X **using** *assms fCod* **by** (*simp add*: *Category.Cidm*)
hence ($m2z_C$ (Id C X)) |∈| (Hom_C (cod_C ($z2m_C\ f$)) X) **using** *assms* **by** (*simp add*: *LSCategory.m2zz2m*)
thus *?thesis* **by** (*simp add*: *HomContraAt*)
qed
also have ... = (η \$\$ Y) |@| ($m2z_C$ (($z2m_C\ f$) ;;$_C$ (Id C X)))
using *assms* **by** (*simp add*: *LSCategory.m2zz2mInv Category.CatIdInMor*)
also have ... = (η \$\$ Y) |@| ($m2z_C$ ($z2m_C\ f$)) **using** *assms*(*1*) *fCod fMor Category.Cidr*[*of* C ($z2m_C\ f$)] **by** (*simp*)
also have ... = (η \$\$ Y) |@| f **using** *assms YObj fHom* **by** (*simp add*: *LSCategory.z2mm2z*)
finally show ($YMapInv$ C X F ($YMap$ C X η) \$\$ Y) |@| f = (η \$\$ Y) |@| f .
}
qed
}
qed

lemma *YMap2*:
assumes *LSCategory* C **and** *Ftor* F : (Op C) ⟶ *SET* **and** X ∈ *Obj* C
and x |∈| (F @@ X)
shows $YMap$ C X ($YMapInv$ C X F x) = x
proof(*simp only*: *YMap-def*)
let *?η* = ($YMapInv$ C X F x)
have (*?η* \$\$ X) = $ZFfun$ (Hom_C X X) (F @@ X) (λ f . (F ## ($z2m_C\ f$)) |@| x) **using** *assms* **by** (*simp add*: *YMapInvApp*)
moreover have ($m2z_C$ (Id C X)) |∈| (Hom_C X X) **using** *assms* **by** (*simp add*: *Category.Simps LSCategory.m2zz2m*)
ultimately have (*?η* \$\$ X) |@| ($m2z_C$ (Id C X)) = (F ## ($z2m_C$ ($m2z_C$ (Id C X)))) |@| x
by (*simp add*: *ZFfunApp*)
also have ... = (F ## (Id C X)) |@| x **using** *assms* **by** (*simp add*: *Cate-*

gory.CatIdInMor LSCategory.m2zz2mInv)
also have ... = (*Id SET* (*F* @@ *X*)) |@| *x*
proof−
have *X* ∈ *Obj* (*Op C*) **using** *assms* **by** (*auto simp add*: *OppositeCategory-def*)
hence (*F* ## (*Id* (*Op C*) *X*)) = (*Id SET* (*F* @@ *X*))
using *assms* **by**(*simp add*: *FunctorId*)
moreover have (*Id* (*Op C*) *X*) = (*Id C X*) **using** *assms* **by** (*auto simp add*: *OppositeCategory-def*)
ultimately show *?thesis* **by** *simp*
qed
also have ... = *x* **using** *assms* **by** (*simp add*: *SETId*)
finally show (*?η* $$ *X*) |@| (*m2z*$_C$ (*Id C X*)) = *x* **.**
qed

lemma *YFtorNT-YMapInv*:
assumes *LSCategory C* **and** *f maps*$_C$ *X to Y*
shows *YFtorNT C f* = *YMapInv C X* (*Hom*$_C$[−, *Y*]) (*m2z*$_C$ *f*)
proof(*simp only*: *YFtorNT-def YMapInv-def*, *rule NatTransExt'*)
have *Cf*: *cod*$_C$ *f* = *Y* **and** *Df*: *dom*$_C$ *f* = *X* **using** *assms* **by** *auto*
thus *NTCod* (*YFtorNT' C f*) = *NTCod* (*YMapInv' C X* (*Hom*$_C$[−, *Y*]) (*m2z*$_C$*f*))
by(*simp add*: *YFtorNT'-def YMapInv'-def*)
have *Hom*$_C$[−,*dom*$_C$ *f*] = *YFtor C* @@ *X* **using** *Df assms* **by** (*simp add*: *YFtorObj Category.MapsToObj*)
thus *NTDom* (*YFtorNT' C f*) = *NTDom* (*YMapInv' C X* (*Hom*$_C$[−, *Y*]) (*m2z*$_C$*f*))
by(*simp add*: *YFtorNT'-def YMapInv'-def*)
{
fix *Z* **assume** *ObjZ1*: *Z* ∈ *Obj* (*NTCatDom* (*YFtorNT' C f*))
have *ObjZ2*: *Z* ∈ *Obj C* **using** *ObjZ1* **by** (*simp add*: *YFtorNT'-def NTCat-Dom-def OppositeCategory-def HomFtorContraDom*)
moreover have *ObjX*: *X* ∈ *Obj C* **and** *ObjY*: *Y* ∈ *Obj C* **using** *assms* **by** (*simp add*: *Category.MapsToObj*)+
moreover
{
fix *x* **assume** *x*: *x* |∈| (*Hom*$_C$ *Z X*)
have *m2z*$_C$ ((*z2m*$_C$ *x*) ;;$_C$ *f*) = ((*Hom*$_C$[−, *Y*]) ## (*z2m*$_C$ *x*)) |@| (*m2z*$_C$*f*)
proof−
have *morf*: *f* ∈ *Mor C* **using** *assms* **by** *auto*
have *mapsx*: (*z2m*$_C$ *x*) *maps*$_C$ *Z to X* **using** *x assms*(*1*) *ObjZ2 ObjX* **by** (*simp add*: *LSCategory.z2mm2z*)
hence *morx*: (*z2m*$_C$ *x*) ∈ *Mor C* **by** *auto*
hence (*Hom*$_C$[−, *Y*]) ## (*z2m*$_C$ *x*) = (*HomC*$_C$[(*z2m*$_C$ *x*), *Y*]) **using** *assms* **by** (*simp add*: *HomContraMor*)
moreover have (*HomC*$_C$[(*z2m*$_C$ *x*), *Y*]) |@| (*m2z*$_C$ *f*) = *m2z*$_C$ ((*z2m*$_C$ *x*) ;;$_C$ (*z2m*$_C$ (*m2z*$_C$ *f*)))
proof (*rule HomContraAt*)
have *cod*$_C$ (*z2m*$_C$ *x*) = *X* **using** *mapsx* **by** *auto*
thus (*m2z*$_C$ *f*) |∈| (*Hom*$_C$ (*cod*$_C$ (*z2m*$_C$ *x*)) *Y*) **using** *assms* **by** (*simp add*: *LSCategory.m2zz2m*)
qed

```
      moreover have (z2m_C (m2z_C f)) = f using assms morf by (simp add:
LSCategory.m2zz2mInv)
       ultimately show ?thesis by simp
     qed
   }
    ultimately show (YFtorNT' C f) $$ Z = (YMapInv' C X (Hom_C[−,Y])
(m2z_C f)) $$ Z using Cf Df assms
   apply(simp add: YFtorNT'-def YMapInv'-def HomFtorMap-def HomFtorOpObj)
    apply(rule ZFfun-ext, rule allI, rule impI, simp)
    done
 }
qed

lemma YMapYoneda:
 assumes LSCategory C and f maps_C X to Y
 shows YFtor C ## f = YMapInv C X (YFtor C @@ Y) (m2z_C f)
proof−
 have f ∈ Mor C using assms by auto
 moreover have Y ∈ Obj C using assms by (simp add: Category.MapsToObj)
 moreover have YFtorNT C f = YMapInv C X (Hom_C[−,Y]) (m2z_C f) using
assms by (simp add: YFtorNT-YMapInv)
 ultimately show ?thesis using assms by (simp add: YFtorMor YFtorObj)
qed

lemma YonedaFull:
 assumes LSCategory C and X ∈ Obj C and Y ∈ Obj C
 and NT η : (YFtor C @@ X) ⟹ (YFtor C @@ Y)
 shows YFtor C ## (z2m_C (YMap C X η)) = η
 and z2m_C (YMap C X η) maps_C X to Y
proof−
  have ftor: Ftor (YFtor C @@ Y) : (Op C) ⟶ SET using assms by (simp
add: YFtorObj HomFtorContraFtor)
 hence (YMap C X η) |∈| ((YFtor C @@ Y) @@ X) using assms by (simp add:
YMapImage)
 hence yh: (YMap C X η) |∈| (Hom_C X Y) using assms by (simp add: YFtorObj2)
 thus (z2m_C (YMap C X η)) maps_C X to Y using assms by (simp add: LSCat-
egory.z2mm2z)
 hence YFtor C ## (z2m_C (YMap C X η)) = YMapInv C X (YFtor C @@ Y)
(m2z_C (z2m_C (YMap C X η)))
   using assms yh by (simp add: YMapYoneda)
 also have ... = YMapInv C X (YFtor C @@ Y) (YMap C X η)
   using assms yh by (simp add: LSCategory.z2mm2z)
 finally show YFtor C ## (z2m_C (YMap C X η)) = η using assms ftor by
(simp add: YMap1)
qed

lemma YonedaFaithful:
 assumes LSCategory C and f maps_C X to Y and g maps_C X to Y
 and YFtor C ## f = YFtor C ## g
```

shows $f = g$
proof−
have *ObjX*: $X \in Obj\ C$ **and** *ObjY*: $Y \in Obj\ C$ **using** *assms* **by** (*simp add*: *Category.MapsToObj*)+
have *M2Zf*: $(m2z_C\ f)$ |∈| $((YFtor\ C\ @@\ Y)\ @@\ X)$ **and** *M2Zg*: $(m2z_C\ g)$ |∈| $((YFtor\ C\ @@\ Y)\ @@\ X)$
using *assms ObjX ObjY* **by** (*simp add*: *LSCategory.m2zz2m YFtorObj2*)+
have *Ftor*: $Ftor\ (YFtor\ C\ @@\ Y) : (Op\ C) \longrightarrow SET$ **using** *assms ObjY* **by** (*simp add*: *YFtorObj HomFtorContraFtor*)
have *Morf*: $f \in Mor\ C$ **and** *Morg*: $g \in Mor\ C$ **using** *assms* **by** *auto*
have $YMapInv\ C\ X\ (YFtor\ C\ @@\ Y)\ (m2z_C\ f) = YMapInv\ C\ X\ (YFtor\ C\ @@\ Y)\ (m2z_C\ g)$
using *assms* **by** (*simp add*: *YMapYoneda*)
hence $YMap\ C\ X\ (YMapInv\ C\ X\ (YFtor\ C\ @@\ Y)\ (m2z_C\ f)) = YMap\ C\ X\ (YMapInv\ C\ X\ (YFtor\ C\ @@\ Y)\ (m2z_C\ g))$
by *simp*
hence $(m2z_C\ f) = (m2z_C\ g)$ **using** *assms ObjX ObjY M2Zf M2Zg Ftor* **by** (*simp add*: *YMap2*)
thus $f = g$ **using** *assms Morf Morg* **by** (*simp add*: *LSCategory.mor2ZFInj*)
qed

lemma *YonedaEmbedding*:
assumes *LSCategory C* **and** $A \in Obj\ C$ **and** $B \in Obj\ C$ **and** $(YFtor\ C)\ @@\ A = (YFtor\ C)\ @@\ B$
shows $A = B$
proof−
have *AObjOp*: $A \in Obj\ (Op\ C)$ **and** *BObjOp*: $B \in Obj\ (Op\ C)$ **using** *assms* **by** (*simp add*: *OppositeCategory-def*)+
hence *FtorA*: $Ftor\ (Hom_C[-,A]) : (Op\ C) \longrightarrow SET$ **and** *FtorB*: $Ftor\ (Hom_C[-,B]) : (Op\ C) \longrightarrow SET$
using *assms* **by** (*simp add*: *HomFtorContraFtor*)+
have $Hom_C[-,A] = Hom_C[-,B]$ **using** *assms* **by** (*simp add*: *YFtorObj*)
hence $(Hom_C[-,A])\ \#\#\ (Id\ (Op\ C)\ A) = (Hom_C[-,B])\ \#\#\ (Id\ (Op\ C)\ A)$ **by** *simp*
hence $Id\ SET\ ((Hom_C[-,A])\ @@\ A) = Id\ SET\ ((Hom_C[-,B])\ @@\ A)$
using *AObjOp BObjOp FtorA FtorB* **by** (*simp add*: *FunctorId*)
hence $Id\ SET\ (Hom_C\ A\ A) = Id\ SET\ (Hom_C\ A\ B)$ **using** *assms* **by** (*simp add*: *HomFtorOpObj*)
hence $Hom_C\ A\ A = Hom_C\ A\ B$ **using** *SETCategory* **by** (*simp add*: *Category.IdInj SETobj*[*of* $Hom_C\ A\ A$] *SETobj*[*of* $Hom_C\ A\ B$])
moreover have $(m2z_C\ (Id\ C\ A))$ |∈| $(Hom_C\ A\ A)$ **using** *assms* **by** (*simp add*: *Category.Cidm LSCategory.m2zz2m*)
ultimately have $(m2z_C\ (Id\ C\ A))$ |∈| $(Hom_C\ A\ B)$ **by** *simp*
hence $(z2m_C\ (m2z_C\ (Id\ C\ A)))\ maps_C\ A\ to\ B$ **using** *assms* **by** (*simp add*: *LSCategory.z2mm2z*)
hence $(Id\ C\ A)\ maps_C\ A\ to\ B$ **using** *assms* **by** (*simp add*: *Category.CatIdInMor LSCategory.m2zz2mInv*)
hence $cod_C\ (Id\ C\ A) = B$ **by** *auto*
thus *?thesis* **using** *assms* **by** (*simp add*: *Category.CatIdDomCod*)

qed

end

References

[1] A. Katovsky. Category theory in Isabelle/HOL, 2010. http://www.srcf.ucam.org/~apk32/Isabelle/Category/Cat.pdf.