

Category Theory

Alexander Katovsky

February 23, 2021

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	1
2	Universe	11
3	Monadic Equational Theory	18
4	Functor	42
5	Natural Transformation	54
6	The Category of Sets	70
7	Yoneda	91

1 Category

```
theory Category
imports HOL-Library.FuncSet
begin

record ('o,'m) Category =
  Obj :: 'o set (obj1 70)
  Mor :: 'm set (mor1 70)
  Dom :: 'm  $\Rightarrow$  'o (dom1 - [80] 70)
  Cod :: 'm  $\Rightarrow$  'o (cod1 - [80] 70)
```

$Id :: 'o \Rightarrow 'm$ (*id1* - [80] 75)
 $Comp :: 'm \Rightarrow 'm \Rightarrow 'm$ (**infixl** ;;₁ 70)

definition

$MapsTo :: ('o, 'm, 'a)$ *Category-scheme* $\Rightarrow 'm \Rightarrow 'o \Rightarrow 'o \Rightarrow bool$ (*- maps₁ - 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** $\approx_{>1}$ 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 $CcompExt: (case-prod\ (Comp\ C)) \in extensional\ (\{(f,g) \mid f\ g \cdot f \approx_{>}\ g\})$

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 $Cassoc : \llbracket f \approx_{>}\ g ; g \approx_{>}\ h \rrbracket \Longrightarrow (f ;; g) ;; h = f ;; (g ;; h)$
and $Ccompt : \llbracket f\ maps\ X\ to\ Y ; g\ maps\ Y\ to\ Z \rrbracket \Longrightarrow (f ;; g)\ maps\ X\ to\ Z$

definition

$MakeCat :: ('o, 'm, 'a)$ *Category-scheme* $\Rightarrow ('o, 'm, 'a)$ *Category-scheme* **where**
 $MakeCat\ C \equiv \langle$
 $Obj = Obj\ C,$
 $Mor = Mor\ C,$
 $Dom = restrict\ (Dom\ C)\ (Mor\ C),$
 $Cod = restrict\ (Cod\ C)\ (Mor\ C),$
 $Id = restrict\ (Id\ C)\ (Obj\ C),$
 $Comp = \lambda\ f\ g . (restrict\ (case-prod\ (Comp\ C))\ (\{(f,g) \mid f\ g \cdot f \approx_{>C}\ g\}))$
 $(f,g),$
 $\dots = Category.more\ C$
 \rangle

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

lemma *MakeCatCod*: $f \in mor_C \implies cod_C f = cod_{MakeCat\ C} f$
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*: $f \approx_{> MakeCat\ C} g \implies f ;;_{MakeCat\ C} g = f ;;_C g$
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* \implies *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]*: $\llbracket f \text{ maps}_C X \text{ to } Y ; \llbracket f \in mor_C ; dom_C f = X ; cod_C f = Y \rrbracket \implies R \rrbracket \implies R$
by (*auto simp add: MapsTo-def*)

lemma *MapsToI[intro]*: $\llbracket f \in mor_C ; dom_C f = X ; cod_C f = Y \rrbracket \implies f \text{ maps}_C X \text{ to } Y$
by (*auto simp add: MapsTo-def*)

lemma *CompDefinedE[elim]*: $\llbracket f \approx_{> C} g ; \llbracket f \in mor_C ; g \in mor_C ; cod_C f = dom_C g \rrbracket \implies R \rrbracket \implies R$
by (*auto simp add: CompDefined-def*)

lemma *CompDefinedI[intro]*: $\llbracket f \in mor_C ; g \in mor_C ; cod_C f = dom_C g \rrbracket \implies f \approx_{> C} g$
by (*auto simp add: CompDefined-def*)

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

proof-
have f maps $(\text{dom } f)$ to $(\text{dom } g)$
and g maps $(\text{dom } g)$ to $(\text{cod } g)$ **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 $f \approx_{>C} g$
proof(*rule CompDefinedI*)
show $f \in \text{mor}_C$ **and** $g \in \text{mor}_C$ **using** *assms* **by** *auto*
have $\text{cod}_C f = Y$ **and** $\text{dom}_C g = Y$ **using** *assms* **by** *auto*
thus $\text{cod}_C f = \text{dom}_C g$ **by** *simp*
qed

lemma (**in** *Category*) *MapsToMorDomCod*:
assumes $f \approx_{>} g$
shows $f ;; g \in \text{mor}$ **and** $\text{dom } (f ;; g) = \text{dom } f$ **and** $\text{cod } (f ;; g) = \text{cod } g$
proof-
have $(f ;; g)$ maps $(\text{dom } f)$ to $(\text{cod } g)$ **using** *assms* **by** (*simp add: MapsToCompI*)
thus $f ;; g \in \text{mor}$ **and** $\text{dom } (f ;; g) = \text{dom } f$ **and** $\text{cod } (f ;; g) = \text{cod } g$ **by** *auto*
qed

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

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

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

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

lemma (*in Category*) *CatIdInMor*: $X \in \text{obj} \implies \text{id } X \in \text{mor}$
by (*auto simp add: Cidm*)

lemma (*in Category*) *MapsToId*: **assumes** $X \in \text{obj}$ **shows** $\text{id } X \approx > \text{id } X$
proof(*rule CompDefinedI*)
have $\text{id } X$ *maps* X *to* X **using** *assms* **by** (*simp add: Cidm*)
thus $\text{id } X \in \text{mor}$ **and** $\text{id } X \in \text{mor}$ **and** $\text{cod } (\text{id } X) = \text{dom } (\text{id } X)$ **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*: $f ;; g = \text{id } (\text{dom } f)$
and *2*: $h ;; f = \text{id } (\text{cod } f)$
shows $h = g$
proof-
have *mor*: $h \in \text{mor} \wedge g \in \text{mor}$
and *dc*: $\text{dom } f = \text{cod } h \wedge \text{cod } f = \text{dom } g$ **using** *0 z* **by** *auto*
then have $h = h ;; \text{id } (\text{dom } f)$ **by** (*auto simp add: Simps*)
also have $\dots = h ;; (f ;; g)$ **using** *1* **by** *simp+*
also have $\dots = (h ;; f) ;; g$ **using** *0 z* **by** (*simp add: Cassoc*)
also have $\dots = (\text{id } (\text{cod } f)) ;; g$ **using** *2* **by** *simp+*
also have $\dots = g$ **using** *mor dc* **by** (*auto simp add: Simps*)
finally show *?thesis* .
qed

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

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

proof-

have $0: id\ X \in mor$ **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*: $\forall f . f \approx> \iota \longrightarrow f ;; \iota = f$
shows $id\ X = \iota$

proof(*rule LeftRightInvUniq [of id X id X ι]*)

have $0: X \in obj$ **using** *iota* **by** (*auto simp add: Simps*)
hence $id\ X$ *maps* X *to* X **by** (*simp add: Cidm*)
thus $1: id\ X \approx> \iota$ **using** *iota* **by** (*auto simp add: Simps*)
show $id\ X \approx> id\ X$ **using** 0 **by** (*auto simp add: Simps*)
show $(id\ X) ;; \iota = (id\ (dom\ (id\ X)))$ **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* $\Rightarrow 'm \Rightarrow 'm \Rightarrow bool$ (*cin_{v1}* - - 60) **where**
inverse-rel $C\ f\ g \equiv (f \approx>_C g) \wedge (f ;;_C g) = (id_C\ (dom_C\ f)) \wedge (g ;;_C f) = (id_C\ (cod_C\ f))$

definition

isomorphism :: $('o, 'm, 'a)$ *Category-scheme* $\Rightarrow 'm \Rightarrow bool$ (*ciso₁* - [70]) **where**
isomorphism $C\ f \equiv \exists g . inverse-rel\ C\ f\ g$

lemma (**in** *Category*) *Inverse-relI*: $\llbracket f \approx> g ; f ;; g = id\ (dom\ f) ; g ;; f = id\ (cod\ f) \rrbracket \Longrightarrow (cinv\ f\ g)$
by (*auto simp add: inverse-rel-def*)

lemma (**in** *Category*) *Inverse-relE[elim]*: $\llbracket cinv\ f\ g ; \llbracket f \approx> g ; f ;; g = id\ (dom\ f) ; g ;; f = id\ (cod\ f) \rrbracket \Longrightarrow P \rrbracket \Longrightarrow P$
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 $1: f \approx> g$ **using** *assms* **by** *auto*
show $2: g \approx> f$
proof(*rule CompDefinedI*)
show $g \in mor$ **and** $0: f \in mor$ **using** *assms* **by** *auto*

have $f ;; g$ maps dom f to cod g **using** 1 **by** (simp add: MapsToCompI)
hence $\text{cod } g = \text{cod } (f ;; g)$ **by** auto
also have $\dots = \text{cod } (\text{id } (\text{dom } f))$ **using** *assms* **by** (auto simp add: inverse-rel-def)
also have $\dots = \text{dom } f$
proof-
have $\text{dom } f \in \text{obj}$ **using** 0 **by** (simp add: Simps)
hence $\text{id } (\text{dom } f)$ maps (dom f) to (dom f) **by** (simp add: Simps)
thus ?thesis **by** auto
qed
finally show 2: $\text{cod } g = \text{dom } f$ **by** simp
qed
show $g ;; f = \text{id } (\text{dom } g)$ **using** *assms* **by** (auto simp add: inverse-rel-def)
show $f ;; g = \text{id } (\text{cod } g)$ **using** *assms* 1 2 **by** (auto simp add: inverse-rel-def)
qed

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

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

definition
 $\text{inverse} :: ('o, 'm, 'a) \text{Category-scheme} \Rightarrow 'm \Rightarrow 'm$ (Cinv1 - [70]) **where**
 $\text{inverse } C f \equiv \text{THE } g . \text{inverse-rel } C f g$

lemma (in Category) inv2Inv:
assumes $\text{cinv } f g$
shows $\text{ciso } f$ **and** $\text{Cinv } f = g$
proof-
show $\text{ciso } f$ **using** *assms* **by** (auto simp add: isomorphism-def)
hence $\exists! g . \text{cinv } f g$ **using** *assms* **by** (auto simp add: InverseUnique)
thus $\text{Cinv } f = g$ **using** *assms* **by** (auto simp add: inverse-def)
qed

lemma (in *Category*) *iso2Inv*:

assumes *ciso f*

shows $\text{cinv } f \text{ (Cinv } f)$

proof–

have $\exists! g . \text{cinv } f g$ **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 $\text{ciso (Cinv } f) \text{ and (Cinv (Cinv } f)) = f$

proof–

have $\text{cinv } f \text{ (Cinv } f)$ **using** *assms* **by** (*simp add: iso2Inv*)

hence $\text{cinv (Cinv } f) f$ **by** (*simp add: Inverse-relSym[of f]*)

thus $\text{ciso (Cinv } f) \text{ and Cinv (Cinv } f) = f$ **by** (*auto simp add: inv2Inv*)

qed

lemma (in *Category*) *InvIsMor*: $(\text{cinv } f g) \implies (f \in \text{mor} \wedge g \in \text{mor})$

by (*auto simp add: inverse-rel-def*)

lemma (in *Category*) *IsoIsMor*: $\text{ciso } f \implies f \in \text{mor}$

by (*auto simp add: InvIsMor dest: iso2Inv*)

lemma (in *Category*) *InvDomCod*:

assumes *ciso f*

shows $\text{dom (Cinv } f) = \text{cod } f$ **and** $\text{cod (Cinv } f) = \text{dom } f$ **and** $\text{Cinv } f \in \text{mor}$

proof–

have $1: \text{cinv } f \text{ (Cinv } f)$ **using** *assms* **by** (*auto simp add: iso2Inv*)

thus $\text{dom (Cinv } f) = \text{cod } f$ **by** (*auto simp add: inverse-rel-def*)

from 1 **have** $\text{cinv (Cinv } f) f$ **by** (*auto simp add: Inverse-relSym[of f]*)

thus $\text{cod (Cinv } f) = \text{dom } f$ **by** (*auto simp add: inverse-rel-def*)

show $\text{Cinv } f \in \text{mor}$ **using** 1 **by** (*auto simp add: inverse-rel-def*)

qed

lemma (in *Category*) *IsoCompInv*: $\text{ciso } f \implies f \approx > \text{Cinv } f$

by (*auto simp add: IsoIsMor InvDomCod*)

lemma (in *Category*) *InvCompIso*: $\text{ciso } f \implies \text{Cinv } f \approx > f$

by (*auto simp add: IsoIsMor InvDomCod*)

lemma (in *Category*) *IsoInvId1* : $\text{ciso } f \implies (\text{Cinv } f) ;; f = (\text{id (cod } f))$

by (*auto dest: iso2Inv*)

lemma (in *Category*) *IsoInvId2* : $\text{ciso } f \implies f ;; (\text{Cinv } f) = (\text{id (dom } f))$

by (*auto dest: iso2Inv*)

lemma (in *Category*) *IsoCompDef*:

assumes $1: f \approx > g$ **and** $2: \text{ciso } f$ **and** $3: \text{ciso } g$

shows $(Cinv\ g) \approx > (Cinv\ f)$
proof(rule *CompDefinedI*)
show $Cinv\ g \in mor$ **and** $Cinv\ f \in mor$ **using** *assms* **by** (auto simp add:
InvDomCod)
have $cod\ (Cinv\ g) = dom\ g$ **using** 3 **by** (simp add: *InvDomCod*)
also have $\dots = cod\ f$ **using** 1 **by** auto
also have $\dots = dom\ (Cinv\ f)$ **using** 2 **by** (simp add: *InvDomCod*)
finally show $cod\ (Cinv\ g) = dom\ (Cinv\ f)$.
qed

lemma (in *Category*) *IsoCompose*:

assumes 1: $f \approx > g$ **and** 2: *ciso* f **and** 3: *ciso* g
shows *ciso* $(f ;; g)$ **and** $Cinv\ (f ;; g) = (Cinv\ g) ;; (Cinv\ f)$
proof-
have $a: (Cinv\ g) \approx > (Cinv\ f)$ **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 $f ;; g \approx > (Cinv\ g) ;; (Cinv\ f)$
proof(rule *CompDefinedI*)
show $f ;; g \in mor$ **using** d **by** auto
show $(Cinv\ g) ;; (Cinv\ f) \in mor$ **using** c **by** auto
have $cod\ (f ;; g) = cod\ g$ **using** d **by** auto
also have $\dots = dom\ (Cinv\ g)$ **using** *assms* **by** (simp add: *InvDomCod*)
also have $\dots = dom\ ((Cinv\ g) ;; (Cinv\ f))$ **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 $e: g \approx > (Cinv\ g)$ **using** *assms* **by** (simp add: *IsoCompInv*)
have $f: f \in mor$ **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 $\dots = f ;; (id\ (dom\ g)) ;; (Cinv\ f)$ **using** 3 **by** (simp add: *IsoInvId2*)
also have $\dots = f ;; id\ (cod\ f) ;; (Cinv\ f)$ **using** 1 **by** (auto simp add: *Simps*)
also have $\dots = f ;; (Cinv\ f)$ **using** f **by** (auto simp add: *Cidr*)
also have $\dots = id\ (dom\ f)$ **using** 2 **by** (simp add: *IsoInvId2*)
also have $\dots = id\ (dom\ (f ;; g))$ **using** d **by** auto
finally show *?thesis* **by** simp
qed
show $((Cinv\ g) ;; (Cinv\ f)) ;; (f ;; g) = id\ (cod\ (f ;; g))$
proof-
have $e: (Cinv\ f) \approx > f$ **using** *assms* **by** (simp add: *InvCompIso*)
have $f: g \in mor$ **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 $\equiv \exists k . (k \text{ maps }_C A \text{ to } B) \wedge \text{ciso }_C k$

definition

UnitCategory :: (unit, unit) Category **where**
UnitCategory = MakeCat (
Obj = {()} ,
Mor = {()} ,
Dom = ($\lambda f.()$) ,
Cod = ($\lambda f.()$) ,
Id = ($\lambda f.()$) ,
Comp = ($\lambda f g. ()$)
)
)

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 \Rightarrow ('o,'m,'a) Category-scheme
(Op - [65] 65) **where**
OppositeCategory C \equiv (
Obj = Obj C ,
Mor = Mor C ,
Dom = Cod C ,
Cod = Dom C ,
Id = Id C ,
Comp = ($\lambda f g. g ;;_C f$),
... = Category.more C
)
)

lemma OpCatOpCat: Op (Op C) = C

by (simp add: OppositeCategory-def)

lemma OpCatCatAx: Category-axioms C \implies Category-axioms (Op C)

by (*simp add: OppositeCategory-def Category-axioms-def MapsTo-def CompDefined-def*)

lemma *OpCatCatExt*: $\text{ExtCategory } C \implies \text{ExtCategory } (\text{Op } C)$

by (*auto simp add: OppositeCategory-def ExtCategory-def MapsTo-def CompDefined-def extensional-def*)

lemma *OpCatCat*: $\text{Category } C \implies \text{Category } (\text{Op } C)$

by (*intro-locale, simp-all add: Category-def OpCatCatAx OpCatCatExt*)

lemma *MapsToOp*: $f \text{ maps}_C X \text{ to } Y \implies f \text{ maps}_{\text{Op } C} Y \text{ to } X$

by (*simp add: MapsTo-def OppositeCategory-def*)

lemma *MapsToOpOp*: $f \text{ maps}_{\text{Op } C} X \text{ to } Y \implies f \text{ maps}_C Y \text{ to } X$

by (*simp add: MapsTo-def OppositeCategory-def*)

lemma *CompDefOp*: $f \approx_{>C} g \implies g \approx_{>\text{Op } C} f$

by (*simp add: CompDefined-def OppositeCategory-def*)

end

2 Universe

theory *Universe*

imports *HOL-ZF.MainZF*

begin

locale *Universe* =

fixes $U :: \text{ZF (structure)}$

assumes $Uempty : \text{Elem Empty } U$

and $Usubset : \text{Elem } u \ U \implies \text{subset } u \ U$

and $Usingle : \text{Elem } u \ U \implies \text{Elem (Singleton } u) \ U$

and $Upow : \text{Elem } u \ U \implies \text{Elem (Power } u) \ U$

and $Uim : \llbracket \text{Elem } I \ U ; \text{Elem } u \ (\text{Fun } I \ U) \rrbracket \implies \text{Elem (Sum (Range } u)) \ U$

and $Unat : \text{Elem Nat } U$

lemma *ElemLambdaFun* : $(\bigwedge x . \text{Elem } x \ u \implies \text{Elem } (f \ x) \ U) \implies \text{Elem (Lambda } u \ f) \ (\text{Fun } u \ U)$

apply (*subst Elem-Lambda-Fun*)

apply *simp*

done

lemma *RangeRepl*: $\text{Range (Lambda } A \ f) = \text{Repl } A \ f$

apply (*subst Ext*)

apply (*subst Range*)

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

done

lemma (in *Universe*) *Utrans*: $\llbracket \text{Elem } a \ U ; \text{Elem } b \ a \rrbracket \implies \text{Elem } b \ U$
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* : $\text{Elem } u \ U \implies \text{Elem } (\text{Sum } u) \ U$
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* = ($\% x. \text{if } x = \text{Empty} \text{ then } u \text{ else } v$)
 and *?S* = (*Power (Power Empty)*)
 have 1: (*Upair u v*) = *Range (Lambda ?S ?f)*
 by (*subst RangeRepl, simp add: Upair-def*)
 have 2: $\llbracket \text{Elem } u \ U ; \text{Elem } v \ U \rrbracket \implies \text{Elem } (\text{Lambda } ?S \ ?f) \ (\text{Fun } ?S \ U)$
 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*: $\text{Upair } u \ v = \text{union } (\text{Singleton } u) \ (\text{Singleton } v)$
apply (*subst Ext*)
apply (*subst Upair*)
apply (*subst union*)
apply (*subst Singleton*)
apply (*simp*)
done

lemma (in *Universe*) *UniverseUPair*: $\llbracket \text{Elem } u \ U ; \text{Elem } v \ U \rrbracket \implies \text{Elem } (\text{Upair } u$

```

v) U
apply (subst UPairSingleton)
apply (rule UniverseUnion)
apply (rule Usingle, simp)+
done

```

```

lemma (in Universe) UniversePair:  $\llbracket \text{Elem } u \ U ; \text{Elem } v \ U \rrbracket \implies \text{Elem } (\text{Opair } u \ v) \ U$ 
apply (subst Opair-def)
apply ((rule UniverseUPair)+, simp+)+
done

```

```

lemma (in Universe)  $\llbracket \text{Elem } u \ U ; \text{Elem } v \ U \rrbracket \implies \text{Elem } (\text{Sum } (\text{Repl } u \ (\%x \ . \ \text{Singleton } (\text{Opair } x \ v)))) \ U$ 
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:  $\text{Sum } (\text{Repl } A \ (\text{Singleton } o \ f)) = \text{Repl } A \ f$ 

```

```

proof-

```

```

  show ?thesis

```

```

    apply (subst Ext)
    apply (subst Sum)
    apply (subst Repl)+
    apply (auto simp add: Singleton)

```

```

  proof-

```

```

    fix a

```

```

    show  $\text{Elem } a \ A \implies \exists y. \text{Elem } (f \ a) \ y \wedge (\exists a. \text{Elem } a \ A \wedge y = \text{Singleton } (f \ a))$ 

```

```

      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  $\text{Elem } u \ U$  and  $\text{Elem } v \ U$ 

```

```

  shows  $\text{Elem } (\text{CartProd } u \ v) \ U$ 

```

```

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  $\llbracket \text{Elem } u \ U; \text{Elem } v \ U; \text{Elem } x \ u \rrbracket \implies \text{Elem } (\text{Repl } v \ (\text{Opair } x)) \ U$ 
    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  $\llbracket \text{Elem } v \ U; \text{Elem } x \ u; \text{Elem } x \ U; \text{Elem } xa \ v \rrbracket \implies \text{Elem } xa \ U$ 
      by (rule Utrans, simp+)
    qed
  qed
qed

```

```

lemma (in Universe) UniverseSubset:  $\llbracket \text{subset } u \ v; \text{Elem } v \ U \rrbracket \implies \text{Elem } u \ U$ 
  apply (drule-tac HOLZF.Power [of u v, THEN ssubst])
  apply (drule Upow)
  apply (rule Utrans, simp+)
done

```

definition

```

Product :: ZF  $\Rightarrow$  ZF where
Product U = Sep (Fun U (Sum U)) (%f . ( $\forall$  u . Elem u U  $\longrightarrow$  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  $\wedge$  subset A B)  $\longrightarrow$  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)  $\wedge$  (subset b c)  $\longrightarrow$  (subset a c)

```

by ((*subst subset-def*)+, *simp*)
thus ?thesis **using** *assms* **by** *simp*
qed

lemma *SubsetSepTrans*: $subset\ A\ B \implies subset\ (Sep\ A\ p)\ B$
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*: $Elem\ u\ U \implies Elem\ (Product\ u)\ U$
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*: $x \in range\ explode \implies f\ 'x \in range\ explode$
proof–
assume $x \in range\ explode$
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 $f\ 'x \in range\ explode$ **by** *auto*
qed

definition *subsetFn* $X\ Y \equiv \lambda\ x . (if\ x \in Y\ then\ x\ else\ SOME\ y . y \in Y)$

lemma *subsetFn*: $\llbracket Y \neq \{\} ; Y \subseteq X \rrbracket \implies (subsetFn\ X\ Y)\ 'X = Y$
proof(*auto simp add: subsetFn-def*)
fix x **assume** $x \in Y$
thus $(SOME\ y . y \in Y) \in Y$ **using** *someI-ex*[*of* $\lambda\ x . x \in Y$] **by** *auto*
qed

lemma *ZFSubsetRangeExplode*: $\llbracket X \in range\ explode ; Y \subseteq X \rrbracket \implies Y \in range\ explode$
proof(*cases* $Y = \{\}$, *simp*)
have $explode\ Empty = \{\}$ **using** *explode-Empty* **by** *simp*
thus $\{\} \in range\ explode$ **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 $\bigwedge x . x \in A \implies f x \in \text{range explode}$ **and** $A \in \text{range explode}$

shows $(\bigcup x \in A . f x) \in \text{range explode}$

proof–

let $?S = \text{Sep} (\text{Sum} (\text{Repl} (\text{implode } A) (\text{implode } \circ f))) (\lambda y . \exists x . (\text{Elem } x (\text{implode } A)) \wedge (\text{Elem } y (\text{implode } (f x))))$

have $\text{explode } ?S = (\bigcup x \in A . f x)$

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 $\exists z . \text{Elem } x z \wedge (\exists a . a \in A \wedge z = \text{implode } (f a))$

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*: $(\bigwedge (n :: \text{nat}) . f n \in \text{range explode}) \implies (\bigcup n . f n) \in \text{range explode}$

proof–

assume $a: (\bigwedge (n :: \text{nat}) . f n \in \text{range explode})$

have $\text{explode } \text{Nat} \in \text{range explode}$ **by** *simp*

moreover **have** $\bigwedge n . n \in (\text{explode } \text{Nat}) \implies (f \circ \text{Nat2nat}) n \in \text{range explode}$ **using** a

by(*auto simp add: explode-def*)

moreover **have** $(\bigcup n . f n) = (\bigcup n \in (\text{explode } \text{Nat}) . (f \circ \text{Nat2nat}) n)$

proof(*auto simp add: Nat2nat-def*)

fix $x n$ **assume** $aa: x \in f n$ **show** $\exists n \in (\text{explode } \text{Nat}) . x \in f (\text{inv } \text{nat2Nat } n)$

apply(*rule bexI [where ?x = nat2Nat n]*)

by(*auto simp add: aa inj-nat2Nat explode-Elem*)

qed

ultimately **show** $(\bigcup n . f n) \in \text{range explode}$ **using** *ZFUnionRangeExplode* **by** *simp*

qed

lemma *ZFProdFnInRangeExplode*: $\llbracket A \in \text{range explode} ; B \in \text{range explode} \rrbracket \implies f '(A \times B) \in \text{range explode}$

proof–

assume $a: A \in \text{range explode}$ **and** $b: B \in \text{range explode}$

let $?X = (\text{explode } (\text{CartProd } (\text{implode } A) (\text{implode } B)))$

have $f '(A \times B) = (f \circ (\lambda z . (\text{Fst } z, \text{Snd } z))) ' ?X$

proof(*auto simp add: explode-def CartProd image-def Fst Snd*)


```

{
  fix z y assume z: z ∈ A and y: y ∈ B show ∃ x. (∃ a. Elem a (implode A) ∧
    (∃ b. Elem b (implode B) ∧ x = Opair a b)) ∧ 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 ∃ x ∈ A . ∃ y ∈ 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 ∈ range explode by simp
ultimately show f '(A × B) ∈ range explode using ZFImageRangeExplode by
simp
qed

```

lemma *ZFUnionInRangeExplode*: $\llbracket A \in \text{range explode} ; B \in \text{range explode} \rrbracket \implies A \cup B \in \text{range explode}$

proof–

```

assume A ∈ range explode and B ∈ range explode
from this obtain A' B' where A': A = explode A' and B': B = explode B' by
auto
have A ∪ B = explode (union (implode A) (implode B))
  by(auto simp add: explode-union union explode-Elem A' B')
thus A ∪ B ∈ range explode by auto
qed

```

lemma *SingletonInRangeExplode*: $\{x\} \in \text{range explode}$

proof–

```

have explode (Singleton x) = {x} by(auto simp add: explode-def Singleton)
thus ?thesis by auto
qed

```

definition *ZFTriple* :: $[ZF, ZF, ZF] \Rightarrow ZF$ **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*: $ZFTriple\ a\ b\ c = ZFTriple\ a'\ b'\ c' \implies (a = a' \wedge b = b' \wedge c = c')$

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 $\dots = Suc\ 0$ **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*: $Elem\ x\ Nat \implies Nat2nat\ (SucNat\ x) \neq 0$

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*)

BaseFunctions :: f *set* (*Fn*)

SigDom :: $f \Rightarrow t$ (*sDom*)

SigCod :: $f \Rightarrow t$ (*sCod*)

locale *Signature* =

fixes $S :: (t, f)$ *Signature* (**structure**)

assumes *Domt*: $f \in Fn \implies sDom\ f \in Ty$

and *Codt*: $f \in Fn \implies sCod\ f \in Ty$

definition *funsignature-abbrev* $(- \in Sig\ - : - \rightarrow -)$ **where**

$f \in Sig\ S : A \rightarrow B \equiv f \in (BaseFunctions\ S) \wedge A \in (BaseTypes\ S) \wedge B \in (BaseTypes\ S) \wedge$

$(SigDom\ S\ f) = A \wedge (SigCod\ S\ f) = B \wedge Signature\ S$

lemma *funsignature-abbrevE[elim]*:

$\llbracket f \in \text{Sig } S : A \rightarrow B ; \llbracket f \in (\text{BaseFunctions } S) ; A \in (\text{BaseTypes } S) ; B \in (\text{BaseTypes } S) \rrbracket ;$
 $(\text{SigDom } S f) = A ; (\text{SigCod } S f) = B ; \text{Signature } S \rrbracket \Longrightarrow R \rrbracket$
 $\Longrightarrow R$
by (*simp add: funsignature-abbrev-def*)

datatype ('t,'f) *Expression* = *ExprVar* (Vx) | *ExprApp* 'f ('t,'f) *Expression* (-
E@ -)
datatype ('t,'f) *Language* = *Type* 't (\vdash - *Type*) | *Term* 't ('t,'f) *Expression* 't (Vx
: - \vdash - : -) |
Equation 't ('t,'f) *Expression* ('t,'f) *Expression* 't (Vx : - \vdash
- \equiv - : -)

inductive

WellDefined :: ('t,'f) *Signature* \Rightarrow ('t,'f) *Language* \Rightarrow *bool* (*Sig* - \triangleright -) **where**
WellDefinedTy: $A \in \text{BaseTypes } S \Longrightarrow \text{Sig } S \triangleright \vdash A \text{ Type}$
| *WellDefinedVar*: $\text{Sig } S \triangleright \vdash A \text{ Type} \Longrightarrow \text{Sig } S \triangleright (Vx : A \vdash Vx : A)$
| *WellDefinedFn*: $\llbracket \text{Sig } S \triangleright (Vx : A \vdash e : B) ; f \in \text{Sig } S : B \rightarrow C \rrbracket \Longrightarrow \text{Sig } S \triangleright$
 $(Vx : A \vdash (f \text{ E@ } e) : C)$
| *WellDefinedEq*: $\llbracket \text{Sig } S \triangleright (Vx : A \vdash e1 : B) ; \text{Sig } S \triangleright (Vx : A \vdash e2 : B) \rrbracket \Longrightarrow$
 $\text{Sig } S \triangleright (Vx : A \vdash e1 \equiv e2 : B)$

lemmas *WellDefined.intros* [*intro*]

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

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

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

lemma (**in** *Signature*) *SigTy*: $\bigwedge B . \text{Sig } S \triangleright (Vx : A \vdash e : B) \Longrightarrow (A \in \text{BaseTypes } S \wedge B \in \text{BaseTypes } S)$
proof(*induct e*)

$\{$
fix *B* **assume** *a*: $\text{Sig } S \triangleright Vx : A \vdash Vx : B$
have $A = B$ **using** *a SigId*[*of S*] **by** *simp*
thus $A \in \text{Ty} \wedge B \in \text{Ty}$ **using** *a* **by** *auto*
 $\}$
 $\{$
fix *B f e* **assume** *ih*: $\bigwedge B' . \text{Sig } S \triangleright (Vx : A \vdash e : B') \Longrightarrow A \in \text{Ty} \wedge B' \in \text{Ty}$
and *a*: $\text{Sig } S \triangleright (Vx : A \vdash (f \text{ E@ } e) : B)$
from *a* **obtain** *B'* **where** *f*: $f \in \text{Sig } S : B' \rightarrow B$ **and** $\text{Sig } S \triangleright (Vx : A \vdash e :$
 $B')$ **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* (*iS₁*)
ICategory :: (*'o, 'm*) *Category* (*iC₁*)
ITypes :: *'t* \Rightarrow *'o* (*Ty*[-]₁)
IFunctions :: *'f* \Rightarrow *'m* (*Fn*[-]₁)

locale *Interpretation* =
fixes *I* :: (*'t, 'f, 'o, 'm*) *Interpretation* (**structure**)
assumes *ICat*: *Category* *iC*
and *ISig*: *Signature* *iS*
and *It* : $A \in \text{BaseTypes } iS \implies Ty[A] \in \text{Obj } iC$
and *If* : $(f \in \text{Sig } iS : A \rightarrow B) \implies \text{Fn}[f] \text{ maps}_{iC} Ty[A] \text{ to } Ty[B]$

inductive *Interp* (*L*[-]₁ \rightarrow -) **where**
InterpTy: *Sig* *iS_I* $\triangleright \vdash A \text{ Type} \implies$
 $L[\vdash A \text{ Type}]_I \rightarrow (\text{IObj } Ty[A]_I)$
| *InterpVar*: $L[\vdash A \text{ Type}]_I \rightarrow (\text{IObj } c) \implies$
 $L[\vdash Vx : A \vdash Vx : A]_I \rightarrow (\text{IMor } (\text{Id } iC_I c))$
| *InterpFn*: $\llbracket \text{Sig } iS_I \triangleright Vx : A \vdash e : B ;$
 $f \in \text{Sig } iS_I : B \rightarrow C ;$
 $L[\vdash Vx : A \vdash e : B]_I \rightarrow (\text{IMor } g) \rrbracket \implies$
 $L[\vdash Vx : A \vdash (f E@ e) : C]_I \rightarrow (\text{IMor } (g \text{ ;; } iC_{\text{Category } I} \text{ Fn}[f]_I))$
| *InterpEq*: $\llbracket L[\vdash Vx : A \vdash e1 : B]_I \rightarrow (\text{IMor } g1) ;$
 $L[\vdash Vx : A \vdash e2 : B]_I \rightarrow (\text{IMor } g2) \rrbracket \implies$
 $L[\vdash Vx : A \vdash e1 \equiv e2 : B]_I \rightarrow (\text{IBool } (g1 = g2))$

lemmas *Interp.intros* [*intro*]

inductive-cases *InterpTyE* [*elim!*]: $L[\vdash A \text{ Type}]_I \rightarrow i$
inductive-cases *InterpVarE* [*elim!*]: $L[\vdash Vx : A \vdash Vx : A]_I \rightarrow i$
inductive-cases *InterpFnE* [*elim!*]: $L[\vdash Vx : A \vdash (f E@ e) : C]_I \rightarrow i$
inductive-cases *InterpEqE* [*elim!*]: $L[\vdash Vx : A \vdash e1 \equiv e2 : B]_I \rightarrow i$

lemma (**in** *Interpretation*) *InterpEqEq*[*intro*]:

$\llbracket L[\vdash Vx : A \vdash e1 : B] \rightarrow (\text{IMor } g) ; L[\vdash Vx : A \vdash e2 : B] \rightarrow (\text{IMor } g) \rrbracket \implies L[\vdash Vx : A \vdash e1 \equiv e2 : B] \rightarrow (\text{IBool True})$

proof–

assume *a*: $L[\vdash Vx : A \vdash e1 : B] \rightarrow (\text{IMor } g)$ **and** *b*: $L[\vdash Vx : A \vdash e2 : B] \rightarrow (\text{IMor } g)$

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

qed

lemma (in *Interpretation*) *InterpExprWellDefined*:
 $L[Vx : A \vdash e : B] \rightarrow i \implies \text{Sig } iS \triangleright Vx : A \vdash e : B$
apply (rule *Interp.cases*)
by *auto*

lemma (in *Interpretation*) *WellDefined*: $L[\varphi] \rightarrow i \implies \text{Sig } iS \triangleright \varphi$
apply(rule *Interp.cases*)
by (*auto simp add: InterpExprWellDefined*)

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

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

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

```

lemma (in *Interpretation*) *MorphismsPreserved*:

$\bigwedge B i . L[Vx : A \vdash e : B] \rightarrow i \implies \exists g . i = (IMor\ g) \wedge (g\ maps_{iC}\ Ty[A]\ to\ Ty[B])$

proof(*induct e*)

```

    {
      fix B i assume a: L[Vx : A ⊢ Vx : B] → i show ∃ g . i = IMor g ∧ g mapsiC
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]) mapsiC 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:  $\bigwedge B i . L[Vx : A \vdash e : B] \rightarrow i \implies \exists g . i = IMor\ g \wedge g\ maps_{iC}\ Ty[A]\ to\ Ty[B]$ 
    and b: L[Vx : A ⊢ f E@ e : B] → i
    show ∃ g . i = IMor g ∧ g mapsiC 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 mapsiC Ty[A] to Ty[B'] using g by auto
      have fmaps: Fn[f] mapsiC Ty[B'] to Ty[B] using g If by simp
      have (g ;;iC Fn[f]) mapsiC Ty[A] to Ty[B] using ICat fmaps gmaps
Category.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
    end
  }
}

```

qed
 }
 qed

lemma (in *Interpretation*) *Expr2Mor*: $L[Vx : A \vdash e : B] \rightarrow (IMor\ g) \implies (g\ maps_{iC}\ Ty[A]\ to\ Ty[B])$

proof–

assume $a: L[Vx : A \vdash e : B] \rightarrow (IMor\ g)$
 from *MorphismsPreserved*[of $A\ e\ B\ (IMor\ g)$] obtain g' where $(IMor\ g) = (IMor\ g') \wedge (g'\ maps_{iC}\ Ty[A]\ to\ Ty[B])$
 using a by *auto*
 thus *?thesis* by *simp*

qed

lemma (in *Interpretation*) *WellDefinedExprInterp*: $\bigwedge B. (Sig\ iS \triangleright Vx : A \vdash e : B) \implies (\exists i. L[Vx : A \vdash e : B] \rightarrow i)$

proof(*induct e*)

{
 fix B assume $sig: (Sig\ iS \triangleright Vx : A \vdash Vx : B)$ show $\exists i. L[Vx : A \vdash Vx : B] \rightarrow i$

proof–

have $aEqb: A = B$ using sig by (*simp add: SigId*)
 hence $ty: A \in BaseTypes\ iS$ using $sig\ SigTyId$ by *simp*
 hence $L[Vx : A \vdash Vx : A] \rightarrow (IMor\ (Id\ iC\ Ty[A]))$ by *auto*
 thus *?thesis* using $aEqb$ by *auto*

qed

}
 {

fix $f\ e\ B$ assume $a: \bigwedge B. (Sig\ iS \triangleright Vx : A \vdash e : B) \implies (\exists i. L[Vx : A \vdash e : B] \rightarrow i)$

and $b: (Sig\ iS \triangleright Vx : A \vdash f\ E@ e : B)$

show $\exists i. L[Vx : A \vdash f\ E@ e : B] \rightarrow i$

proof–

from b obtain B' where $B': (Sig\ iS \triangleright (Vx : A \vdash e : B')) \wedge (f \in Sig\ iS : B' \rightarrow B)$ by *auto*

from B' a obtain i where $i: L[Vx : A \vdash e : B'] \rightarrow i$ by *auto*

from *MorphismsPreserved*[of $A\ e\ B'\ i$] obtain g where $g: i = (IMor\ g)$ by *auto*

thus *?thesis* using $B'\ i$ by *auto*

qed

}

qed

lemma (in *Interpretation*) *Sig2Mor*: assumes $(Sig\ iS \triangleright Vx : A \vdash e : B)$ shows $\exists g. L[Vx : A \vdash e : B] \rightarrow (IMor\ g)$

proof–

from *WellDefinedExprInterp*[of $A\ e\ B$] *assms* obtain i where $i: L[Vx : A \vdash e : B] \rightarrow i$ 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 (aS1)
```

```
locale Axioms =  
  fixes Ax :: ('t,'f) Axioms (structure)  
  assumes AxT: (aAxioms Ax)  $\subseteq$   $\{(Vx : A \vdash e1 \equiv e2 : B) \mid A B e1 e2 . Sig$   
  (aSignature Ax)  $\triangleright (Vx : A \vdash e1 \equiv e2 : B)\}$   
  assumes AxSig: Signature (aSignature Ax)
```

```
primrec Subst :: ('t,'f) Expression  $\Rightarrow$  ('t,'f) Expression  $\Rightarrow$  ('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:  $\bigwedge C . \llbracket Sig S \triangleright (Vx : A \vdash e : B); Sig S \triangleright (Vx : B \vdash d : C) \rrbracket$   
 $\implies Sig S \triangleright (Vx : A \vdash (sub e in d) : C)$ 
```

```
proof(induct d)
```

```
{  
  fix C assume a: Sig S  $\triangleright Vx : A \vdash e : B$  and b: Sig S  $\triangleright Vx : B \vdash Vx : C$   
  have BCeq: B = C using b SigId[of S] by simp  
  thus Sig S  $\triangleright Vx : A \vdash sub e in Vx : C$  using a by simp  
}  
{  
  fix f d C assume ih:  $\bigwedge B' . \llbracket Sig S \triangleright Vx : A \vdash e : B; Sig S \triangleright Vx : B \vdash d : B' \rrbracket$   
   $\implies Sig S \triangleright Vx : A \vdash sub e in d : B'$  and a: Sig S  $\triangleright Vx : A \vdash e : B$   
  and b: Sig S  $\triangleright Vx : B \vdash f E@ d : C$   
  from b obtain B' where B':  $f \in Sig S : B' \rightarrow C$  and d: Sig S  $\triangleright Vx : B \vdash d$   
: B' by auto  
  hence Sig S  $\triangleright Vx : A \vdash sub e in d : B'$  using ih a by simp  
  hence Sig S  $\triangleright Vx : A \vdash f E@ (sub e in d) : C$  using B' by auto  
  thus Sig S  $\triangleright Vx : A \vdash (sub e in (f E@ d)) : C$  by auto  
}
```

qed

inductive-set (in Axioms) Theory where

```
Ax: A  $\in$  (aAxioms Ax)  $\implies A \in Theory$   
| Refl: Sig (aSignature Ax)  $\triangleright (Vx : A \vdash e : B) \implies (Vx : A \vdash e \equiv e : B) \in Theory$   
| Symm:  $(Vx : A \vdash e1 \equiv e2 : B) \in Theory \implies (Vx : A \vdash e2 \equiv e1 : B) \in Theory$   
| Trans:  $\llbracket (Vx : A \vdash e1 \equiv e2 : B) \in Theory ; (Vx : A \vdash e2 \equiv e3 : B) \in Theory \rrbracket$   
 $\implies$ 
```


$(Vx : A \vdash e1 \equiv e3 : B) \in Theory$
| *Congr*: $\llbracket (Vx : A \vdash e1 \equiv e2 : B) \in Theory ; f \in Sig (aSignature Ax) : B \rightarrow C \rrbracket$
 \implies
 $(Vx : A \vdash (f E@ e1) \equiv (f E@ e2) : C) \in Theory$
| *Subst*: $\llbracket Sig (aSignature Ax) \triangleright (Vx : A \vdash e1 : B) ; (Vx : B \vdash e2 \equiv e3 : C) \in Theory \rrbracket \implies$
 $(Vx : A \vdash (sub e1 in e2) \equiv (sub e1 in e3) : C) \in Theory$

lemma (in *Axioms*) *Equiv2WellDefined*: $\varphi \in Theory \implies Sig aS \triangleright \varphi$

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'*:

$\bigwedge C . \llbracket Sig aS \triangleright Vx : B \vdash d : C ; (Vx : A \vdash e1 \equiv e2 : B) \in Theory \rrbracket \implies$
 $(Vx : A \vdash (sub e1 in d) \equiv (sub e2 in d) : C) \in Theory$

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** $ih: \bigwedge B' . \llbracket Sig aS \triangleright Vx : B \vdash d : B' ; (Vx : A \vdash e1 \equiv e2 : B) \in$
 $Theory \rrbracket$
 $\implies (Vx : A \vdash (sub e1 in d) \equiv (sub e2 in d) : B') \in Theory$ **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** $B': f \in Sig aS : B' \rightarrow C$ **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 $Ax :: ('t, 'f) \text{ Axioms} +$
assumes $AxSound: \varphi \in (aAxioms Ax) \implies L[\varphi] \rightarrow (IBool True)$
and $Seq[simp]: (aSignature Ax) = iS$

lemma (in *Interpretation*) *Equiv*:

assumes $L[Vx : A \vdash e1 \equiv e2 : B] \rightarrow (IBool True)$
shows $\exists g . (L[Vx : A \vdash e1 : B] \rightarrow (IMor g)) \wedge (L[Vx : A \vdash e2 : B] \rightarrow (IMor g))$

proof–

from $assms \text{ Sig2Mor}[of A e1 B]$ **obtain** $g1$ **where** $g1: L[Vx : A \vdash e1 : B] \rightarrow (IMor g1)$ **by** *auto*

from $assms \text{ Sig2Mor}[of A e2 B]$ **obtain** $g2$ **where** $g2: L[Vx : A \vdash e2 : B] \rightarrow (IMor g2)$ **by** *auto*

have $L[Vx : A \vdash e1 \equiv e2 : B] \rightarrow (IBool (g1 = g2))$ **using** $g1 g2$ **by** *auto*

hence $g1 = g2$ **using** $assms \text{ Functional}[of Vx : A \vdash e1 \equiv e2 : B (IBool True) IBool (g1=g2)]$ **by** *simp*

thus *?thesis* **using** $g1 g2$ **by** *auto*

qed

lemma (in *Interpretation*) *SubstComp*: $\bigwedge h C . [(L[Vx : A \vdash e : B] \rightarrow (IMor g)) ; (L[Vx : B \vdash d : C] \rightarrow (IMor h))] \implies$

$(L[Vx : A \vdash (sub e in d) : C] \rightarrow (IMor (g ;;_{iC} h)))$

proof(*induct d, auto*)

{

fix $h C$ **assume** $a: L[Vx : A \vdash e : B] \rightarrow IMor g$ **and** $b: L[Vx : B \vdash Vx : C] \rightarrow (IMor h)$

show $L[Vx : A \vdash e : C] \rightarrow IMor (g ;;_{iC} h)$

proof–

have $beqc: B = C$ **using** $b \text{ SigId}[of iS B C] \text{ InterpExprWellDefined}$ **by** *simp*

have $L[Vx : B \vdash Vx : B] \rightarrow (IMor (Id iC Ty[B]))$ **using** $beqc b$ **by** *auto*

hence $h: h = Id iC Ty[B]$ **using** $b beqc$ **by** (*auto simp add: Functional*)

have $g \text{ maps}_{iC} Ty[A]$ **to** $Ty[B]$ **using** $a \text{ MorphismsPreserved}$ **by** *auto*

hence $g \in Mor iC$ **and** $cod_{iC} g = Ty[B]$ **by** *auto*

hence $(g ;;_{iC} h) = g$ **using** $h \text{ ICat Category.Cidr}[of iC g]$ **by** *simp*

thus *?thesis* **using** $beqc a$ **by** *simp*

qed

}

{

fix $f d C' h C$ **assume**

$i: \bigwedge h C' . L[Vx : B \vdash d : C'] \rightarrow IMor h \implies L[Vx : A \vdash (sub e in d) : C'] \rightarrow (IMor (g ;;_{iC} h))$ **and**

$a: L[Vx : A \vdash e : B] \rightarrow IMor g f \in Sig iS : C' \rightarrow C L[Vx : B \vdash d : C'] \rightarrow IMor h$

show $L[Vx : A \vdash f E@ (sub e in d) : C] \rightarrow (IMor (g ;;_{iC} (h ;;_{iC} Fn[f])))$

proof–

have $L[Vx : A \vdash (sub e in d) : C'] \rightarrow (IMor (g ;;_{iC} h))$ **using** $i a$ **by** *auto*

moreover **hence** $Sig iS \triangleright Vx : A \vdash (sub e in d) : C'$ **using** $\text{InterpExprWellDefined}$ **by** *simp*

ultimately **have** $1: L[Vx : A \vdash f E@ (sub e in d) : C] \rightarrow (IMor ((g ;;_{iC} h))$

```

;;iC Fn[[f]]) using a(2) by auto
  have g mapsiC Ty[[A]] to Ty[[B]] and h mapsiC Ty[[B]] to Ty[[C]] and Fn[[f]]
  mapsiC Ty[[C]] to Ty[[C]]
  using a If Expr2Mor by simp+
  hence g ≈iC h and h ≈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*: $\varphi \in \text{Theory} \implies L[\varphi] \rightarrow (\text{IBool True})$

proof(rule *Theory.induct*, (simp-all add: *AxSound*)+)

```

{
  fix A e B assume sig: Sig iS ▷ Vx : A ⊢ e : B show L[[Vx : A ⊢ e ≡ e : B]] →
  (IBool True)
  proof-
    from sig Sig2Mor[of A e B] obtain g where g: L[[Vx : A ⊢ e : B]] → (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 ⊢ e1 ≡ e2 : B]] → (IBool True) show L[[Vx
  : A ⊢ e2 ≡ e1 : B]] → (IBool True)
  proof-
    from a obtain g where g1: L[[Vx : A ⊢ e1 : B]] → (IMor g) and g2: L[[Vx :
  A ⊢ e2 : B]] → (IMor g)
    using Equiv by auto
    thus ?thesis by auto
  qed
}
{
  fix A e1 e2 B e3 assume a: L[[Vx : A ⊢ e1 ≡ e2 : B]] → (IBool True) and b:
  L[[Vx : A ⊢ e2 ≡ e3 : B]] → (IBool True)
  show L[[Vx : A ⊢ e1 ≡ e3 : B]] → (IBool True)
  proof-
    from a obtain g where g1: L[[Vx : A ⊢ e1 : B]] → (IMor g) and g2: L[[Vx :
  A ⊢ e2 : B]] → (IMor g)
    using Equiv by auto
    from b obtain g' where g1': L[[Vx : A ⊢ e2 : B]] → (IMor g') and g2': L[[Vx
  : A ⊢ e3 : B]] → (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 ⊢ e1≡e2 : B] → (IBool True) and b: f
  ∈ Sig iS : B → C
  show L[Vx : A ⊢ (f E@ e1) ≡ (f E@ e2) : C] → (IBool True)
  proof-
    from a obtain g where g1: L[Vx : A ⊢ e1 : B] → (IMor g) and g2: L[Vx :
  A ⊢ e2 : B] → (IMor g)
    using Equiv by auto
    have s1: Sig iS ▷ Vx : A ⊢ e1 : B and s2: Sig iS ▷ Vx : A ⊢ e2 : B using
  g1 g2 InterpExprWellDefined by simp+
    have L[Vx : A ⊢ (f E@ e1) : C] → (IMor (g ;;iC Fn[f]))
    and L[Vx : A ⊢ (f E@ e2) : C] → (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 ▷ (Vx : A ⊢ e1 : B) and b: L[Vx : B ⊢
  e2≡e3 : C] → (IBool True)
  show L[Vx : A ⊢ (sub e1 in e2) ≡ (sub e1 in e3) : C] → (IBool True)
  proof-
    from b obtain g where g1: L[Vx : B ⊢ e2 : C] → (IMor g) and g2: L[Vx :
  B ⊢ e3 : C] → (IMor g)
    using Equiv by auto
    from a Sig2Mor[of A e1 B] obtain f where f: L[Vx : A ⊢ e1 : B] → (IMor
  f) by auto
    have L[Vx : A ⊢ (sub e1 in e2) : C] → (IMor (f ;;iC g)) using SubstComp
  g1 f by simp
    moreover have L[Vx : A ⊢ (sub e1 in e3) : C] → (IMor (f ;;iC g)) using
  SubstComp g2 f by simp
    ultimately show ?thesis using InterpEqEq by simp
  qed
}
qed

```

```

record ('t,'f) TermEquivCIT =
  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) ∈ range explode

```

```

lemma [simp]: ZFAxioms T ⇒ Axioms T by (simp add: ZFAxioms-def)

```

```

primrec Expr2ZF :: (ZF,ZF) Expression ⇒ ZF where
  Expr2ZFFVx: 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 \Rightarrow (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 \implies e = Vx

by(cases e, auto simp add: ZFType-def ZFTSnd Elem-Empty-Nat Elem-SucNat-Nat ZFSucNeg0)

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 \implies

$\exists f e' . e = (f E@ e') \wedge (Suc (ZFDepth (Expr2ZF e'))) = (ZFDepth (Expr2ZF e))$

by(cases e, auto simp add: ZFType-def ZFTSnd ZFDepth-def ZFTFst ZFZero ZFDepthIn-Nat Nat2nat-SucNat)

lemma *Expr2ZFDepth0*: *ZFDepth* (Expr2ZF e) = 0 \implies *ZFType* (Expr2ZF e) = 0

by(cases e, auto simp add: ZFDepth-def ZFTFst ZFType-def ZFTSnd ZFSucNeg0 ZFDepthInNat)

lemma *Expr2ZFDepthSuc*: *ZFDepth* (Expr2ZF e) = Suc n \implies *ZFType* (Expr2ZF e) = 1

by(cases e, auto simp add: ZFDepth-def ZFTFst ZFType-def ZFTSnd ZFSucZero ZFSucNeg0 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: $\bigwedge n e d . \llbracket n = ZFDepth (Expr2ZF e) ; Expr2ZF e = Expr2ZF d \rrbracket \implies e = d$

proof–

fix n **show** $\bigwedge e d . \llbracket n = ZFDepth (Expr2ZF e) ; Expr2ZF e = Expr2ZF d \rrbracket \implies e = d$

proof(induct n)

{

fix e d **assume** a: $0 = ZFDepth (Expr2ZF e)$ **and** b: $Expr2ZF e = Expr2ZF d$

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:  $\wedge e' d' . \llbracket n = \text{ZFDepth} (\text{Expr2ZF } e') ; \text{Expr2ZF } e' = \text{Expr2ZF } d' \rrbracket \implies e' = d'$ 
  and a:  $\text{Suc } n = \text{ZFDepth} (\text{Expr2ZF } e)$  and b:  $\text{Expr2ZF } e = \text{Expr2ZF } d$ 
  have  $\text{ZFType} (\text{Expr2ZF } e) = 1$  and  $\text{ZFType} (\text{Expr2ZF } d) = 1$  using a b
  Expr2ZFDepthSuc[of e n] by simp+
  from this Expr2ZFType1[of e] Expr2ZFType1[of d] obtain fe fd e' d'
  where e:  $e = (\text{fe } E@ e') \wedge (\text{Suc} (\text{ZFDepth} (\text{Expr2ZF } e'))) = (\text{ZFDepth} (\text{Expr2ZF } e))$  and
  d:  $d = (\text{fd } E@ d') \wedge (\text{Suc} (\text{ZFDepth} (\text{Expr2ZF } d'))) = (\text{ZFDepth} (\text{Expr2ZF } d))$  by auto
  hence  $\text{Suc} (\text{ZFDepth} (\text{Expr2ZF } e')) = \text{Suc } n$  using a by simp
  hence ne:  $(\text{ZFDepth} (\text{Expr2ZF } e')) = n$  by (rule Suc-inject)
  have edat:  $\text{ZFData} (\text{Expr2ZF } e) = \text{Opair } \text{fe} (\text{Expr2ZF } e')$  using e Expr2Data
  by simp
  have ddat:  $\text{ZFData} (\text{Expr2ZF } d) = \text{Opair } \text{fd} (\text{Expr2ZF } d')$  using d Expr2Data
  by simp
  have fd-eq-fe:  $\text{fe} = \text{fd}$  and  $(\text{Expr2ZF } e') = (\text{Expr2ZF } d')$  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  $\text{Expr2ZF } e = \text{Expr2ZF } d \implies e = d$  using a by auto
qed

```

definition $\text{TermEquivClGen } T A e B \equiv \{e' . (\forall x : A \vdash e' \equiv e : B) \in \text{Axioms.Theory } T\}$

definition $\text{TermEquivCl}' T A e B \equiv (\llbracket \text{TDomain} = A , \text{TExprSet} = \text{TermEquivClGen } T A e B , \text{TCodomain} = B \rrbracket)$

definition $m2ZF :: (ZF, ZF) \text{TermEquivCl} T \Rightarrow ZF$ **where**
 $m2ZF t \equiv \text{ZFTriple} (\text{TDomain } t) (\text{implode} (\text{Expr2ZF}' (\text{TExprSet } t))) (\text{TCodomain } t)$

definition $\text{ZF2m} :: (ZF, ZF) \text{Axioms} \Rightarrow ZF \Rightarrow (ZF, ZF) \text{TermEquivCl} T$ **where**
 $\text{ZF2m } T \equiv \text{inv-into} \{ \text{TermEquivCl}' T A e B \mid A e B . \text{True} \} m2ZF$

lemma $\text{TDomain} (\text{TermEquivCl}' T A e B) = A$ **by** (simp add: $\text{TermEquivCl}'\text{-def}$)

lemma $\text{TCodomain} (\text{TermEquivCl}' T A e B) = B$ **by** (simp add: $\text{TermEquivCl}'\text{-def}$)

primrec $\text{WellFormedToSet} :: (ZF, ZF) \text{Signature} \Rightarrow \text{nat} \Rightarrow (ZF, ZF) \text{Expression set}$ **where**

$WFS0: WellFormedToSet S 0 = \{Vx\}$
 $| WFS: WellFormedToSet S (Suc n) = (WellFormedToSet S n) \cup$
 $\{ f E@ e \mid f e . f \in BaseFunctions S \wedge e \in (WellFormedToSet S n) \}$

lemma *WellFormedToSetInRangeExplode*: $ZFAxioms T \implies (Expr2ZF ' (WellFormedToSet aS_T n)) \in range explode$

proof((*induct n*),(*simp add: WellFormedToSet-def SingletonInRangeExplode*))

fix n **assume** $zfx: ZFAxioms T$ **and** $ih: ZFAxioms T \implies Expr2ZF ' WellFormedToSet aS_T n \in range explode$

hence $a: Expr2ZF ' WellFormedToSet aS_T n \in range explode$ **by** *simp*

have $Expr2ZF ' \{ f E@ e \mid f e . f \in BaseFunctions aS_T \wedge e \in (WellFormedToSet aS_T n) \}$

$= (\lambda (f, ze) . Expr2ZF (f E@ (ZF2Expr ze))) ' ((BaseFunctions aS_T) \times Expr2ZF ' (WellFormedToSet aS_T n))$

proof (*auto simp add: image-Collect image-def ZF2Expr-def Expr2ZFinj*)

fix $f e$

assume $f: f \in BaseFunctions aS_T$ **and** $e: e \in WellFormedToSet aS_T n$

show $\exists x \in BaseFunctions aS_T . \exists y. (\exists x \in WellFormedToSet aS_T n. y = Expr2ZF x) \wedge$

$ZFTriple (SucNat (ZFTFst (Expr2ZF e))) (SucNat Empty) (Opair f (Expr2ZF e)) =$

$ZFTriple (SucNat (ZFTFst (Expr2ZF (inv Expr2ZF y)))) (SucNat Empty) (Opair x (Expr2ZF (inv Expr2ZF y)))$

apply(*rule* $beI[\mathbf{where} \ ?x = f]$, *rule* $exI[\mathbf{where} \ ?x = Expr2ZF e]$)

apply (*auto simp add: Expr2ZFinj f e*)

done

show $\exists x. (\exists f e. x = f E@ e \wedge f \in (BaseFunctions aS_T) \wedge e \in WellFormedToSet aS_T n) \wedge$

$ZFTriple (SucNat (ZFTFst (Expr2ZF e))) (SucNat Empty) (Opair f (Expr2ZF e)) = Expr2ZF x$

apply(*rule* $exI[\mathbf{where} \ ?x = f E@ e]$)

apply (*auto simp add: f e*)

done

qed

moreover **have** $(BaseFunctions aS_T) \in range explode$ **using** $zfx ZFAxioms.fnzf$ **by** *simp*

ultimately **have** $Expr2ZF ' \{ f E@ e \mid f e . f \in BaseFunctions aS_T \wedge e \in (WellFormedToSet aS_T n) \} \in range explode$

using $a ZFProdFnInRangeExplode$ **by** *simp*

moreover **have** $Expr2ZF ' WellFormedToSet aS_T (Suc n) = (Expr2ZF ' (WellFormedToSet aS_T n)) \cup$

$(Expr2ZF ' \{ f E@ e \mid f e . f \in BaseFunctions aS_T \wedge e \in (WellFormedToSet aS_T n) \})$ **by** *auto*

ultimately **show** $Expr2ZF ' WellFormedToSet aS_T (Suc n) \in range explode$ **using** $ZFUnionInRangeExplode a$ **by** *simp*

qed

lemma *WellDefinedToWellFormedSet*: $\bigwedge B . (Sig S \triangleright (Vx : A \vdash e : B)) \implies \exists n. e \in WellFormedToSet S n$

```

proof(induct e)
{
  fix B assume Sig S ▷ Vx : A ⊢ Vx : B
  show ∃ n. Vx ∈ WellFormedToSet S n by (rule exI[of - 0], auto)
}
{
  fix f e B assume ih: ∧ B'. Sig S ▷ Vx : A ⊢ e : B' ⇒ ∃ n. e ∈ WellFormedToSet S n
  and Sig S ▷ Vx : A ⊢ (f E@ e) : B
  from this obtain B' where Sig S ▷ (Vx : A ⊢ e : B') and f: f ∈ Sig S : B' → B by auto
  from this obtain n where a: e ∈ WellFormedToSet S n using ih by auto
  have ff: f ∈ BaseFunctions S using f by auto
  show ∃ n. (f E@ e) ∈ WellFormedToSet S n by(rule exI[of - Suc n], auto simp add: a ff)
}
qed

```

lemma *TermSetInSet*: *ZFAxioms T* ⇒ *Expr2ZF* ‘ (*TermEquivClGen T A e B*) ∈ *range explode*

```

proof-
  assume zfax: ZFAxioms T
  have (TermEquivClGen T A e B) ⊆ {e' . (Sig aST ▷ (Vx : A ⊢ e' : B))}
  proof(auto simp add: TermEquivClGen-def)
    fix x assume Vx : A ⊢ x≡e : B ∈ Axioms.Theory T
    hence Sig aST ▷ Vx : A ⊢ x≡e : B by (auto simp add: Axioms.Equiv2WellDefined zfax)
    thus Sig aST ▷ Vx : A ⊢ x : B by auto
  qed
  also have ... ⊆ (∪ n . (WellFormedToSet aST n)) by (auto simp add: WellDefinedToWellFormedSet)
  finally have Expr2ZF ‘ (TermEquivClGen T A e B) ⊆ Expr2ZF ‘ (∪ n . (WellFormedToSet aST n)) by auto
  also have ... = (∪ n . (Expr2ZF ‘ (WellFormedToSet aST n))) by auto
  finally have Expr2ZF ‘ (TermEquivClGen T A e B) ⊆ (∪ n . (Expr2ZF ‘ (WellFormedToSet aST n))) by auto
  moreover have (∪ n . (Expr2ZF ‘ (WellFormedToSet aST n))) ∈ range explode

  using zfax ZFUnionNatInRangeExplode WellFormedToSetInRangeExplode by auto
  ultimately show ?thesis using ZFSubsetRangeExplode[of
    (∪ n . (Expr2ZF ‘ (WellFormedToSet aST n))) Expr2ZF ‘ (TermEquivClGen T A e B)] by simp
  qed

```

lemma *m2ZFinj-on*: *ZFAxioms T* ⇒ *inj-on m2ZF* {*TermEquivCl' T A e B* | *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*: $ZFAxioms\ T \implies ZF2m\ T\ (m2ZF\ (TermEquivCl'\ T\ A\ e\ B)) = (TermEquivCl'\ T\ A\ e\ B)$

proof–

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

definition *TermEquivCl* ($[-,-,-]_1$) **where** $[A,e,B]_T \equiv m2ZF\ (TermEquivCl'\ T\ A\ e\ B)$

definition *CLDomain* $T \equiv TDomain\ o\ ZF2m\ T$

definition *CLCodomain* $T \equiv TCodomain\ o\ ZF2m\ T$

definition *CanonicalComp* $T\ f\ g \equiv$

$THE\ h.\ \exists\ e\ e'.\ h = [CLDomain\ T\ f,\ sub\ e\ in\ e',\ CLCodomain\ T\ g]_T \wedge$
 $f = [CLDomain\ T\ f,\ e,\ CLCodomain\ T\ f]_T \wedge g = [CLDomain\ T\ g,\ e',\ CLCodomain\ T\ g]_T$

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

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

lemma *Equiv2Cl*: **assumes** *Axioms T* **and** $(Vx : A \vdash e \equiv d : B) \in Axioms.Theory\ T$ **shows** $[A,e,B]_T = [A,d,B]_T$

proof–

have $\{e'.\ (Vx : A \vdash e' \equiv e : B) \in Axioms.Theory\ T\} = \{e'.\ (Vx : A \vdash e' \equiv d : B) \in Axioms.Theory\ T\}$

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*: $Sig\ aS_T \triangleright (Vx : A \vdash e : B)$ **and** *cl*: $[A,e,B]_T = [A,d,B]_T$

shows $(Vx : A \vdash e \equiv d : B) \in Axioms.Theory\ T$

proof–

have $ZF2m\ T\ (m2ZF\ (TermEquivCl'\ T\ A\ e\ B)) = ZF2m\ T\ (m2ZF\ (TermEquivCl'\ T\ A\ d\ B))$

$T A d B$) **using** cl **by** (*simp add: TermEquivCl-def*)
hence $a: \text{TermEquivCl}' T A e B = \text{TermEquivCl}' T A d B$ **using** $assms$ **by** (*simp add: ZF2m*)
have $(\forall x : A \vdash e \equiv e : B) \in \text{Axioms.Theory } T$ **using** axt sa $\text{Axioms.Refl[of } T]$
by *simp*
hence $e \in \text{TermEquivClGen } T A e B$ **by** (*simp add: TermEquivClGen-def*)
hence $e \in \text{TermEquivClGen } T A d B$ **using** a **by** (*simp add: TermEquivCl'-def*)
thus $?thesis$ **by** (*simp add: TermEquivClGen-def*)
qed

lemma *CanonicalCompWellDefined:*

assumes $zaxt: \text{ZF}Axioms T$ **and** $\text{Sig } aS_T \triangleright (\forall x : A \vdash d : B)$ **and** $\text{Sig } aS_T \triangleright (\forall x : B \vdash d' : C)$
shows $\text{CanonicalComp } T [A,d,B]_T [B,d',C]_T = [A, \text{sub } d \text{ in } d', C]_T$
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 $axt: \text{Axioms } T$ **using** $assms$ **by** *simp*
have $a: (\forall x : A \vdash d \equiv e : B) \in \text{Axioms.Theory } T$ **using** $assms$ de Cl2Equiv **by**
simp
hence $sa: (\forall x : A \vdash (\text{sub } d \text{ in } d') \equiv (\text{sub } e \text{ in } d') : C) \in \text{Axioms.Theory } T$
using $assms$ $\text{Axioms.Subst'[of } T]$ **by** *simp*
have $b: (\forall x : B \vdash d' \equiv e' : C) \in \text{Axioms.Theory } T$ **using** $assms$ de' Cl2Equiv
by *simp*
have $\text{Sig } aS_T \triangleright (\forall x : A \vdash d \equiv e : B)$ **using** a axt $\text{Axioms.Equiv2WellDefined[of } T]$ **by** *simp*
hence $\text{Sig } aS_T \triangleright (\forall x : A \vdash e : B)$ **by** *auto*
hence $(\forall x : A \vdash (\text{sub } e \text{ in } d') \equiv (\text{sub } e \text{ in } e') : C) \in \text{Axioms.Theory } T$ **using** b
 $\text{Axioms.Subst[of } T]$ axt **by** *simp*
hence $(\forall x : A \vdash (\text{sub } e \text{ in } e') \equiv (\text{sub } d \text{ in } d') : C) \in \text{Axioms.Theory } T$ **using**
 sa axt
by (*blast intro: Axioms.Symm[of } T] $\text{Axioms.Trans[of } T]$)
thus $[A, \text{sub } e \text{ in } e', C]_T = [A, \text{sub } d \text{ in } d', C]_T$ **using** $zaxt$ Equiv2Cl **by** *simp*
qed*

definition $\text{CanonicalCat}' T \equiv \langle \rangle$

$\text{Obj} = \text{BaseTypes } (aS_T),$
 $\text{Mor} = \{[A,e,B]_T \mid A e B . \text{Sig } aS_T \triangleright (\forall x : A \vdash e : B)\},$
 $\text{Dom} = \text{CLDomain } T,$
 $\text{Cod} = \text{CLCodomain } T,$
 $\text{Id} = (\lambda A . [A, \forall x, A]_T),$
 $\text{Comp} = \text{CanonicalComp } T$
 \rangle

definition $\text{CanonicalCat } T \equiv \text{MakeCat } (\text{CanonicalCat}' T)$

lemma *CanonicalCat'MapsTo:*

assumes f $\text{maps}_{\text{CanonicalCat}' T} X$ $\text{to } Y$ **and** $zx: \text{ZF}Axioms T$
shows $\exists ef . f = [X, ef, Y]_T \wedge \text{Sig } (a\text{Signature } T) \triangleright (\forall x : X \vdash ef : Y)$

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 $CanonicalCatCat': ZFAxioms T \implies Category-axioms (CanonicalCat' T)$

proof(*auto simp only: Category-axioms-def*)

have $obj: obj_{CanonicalCat' T} = BaseType (aSignature T)$ **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** $domf fwd obj CLDomain[of T A e B]$ $absig zx$

by *simp*

show $cod_{CanonicalCat' T} f \in obj_{CanonicalCat' T}$ **using** $codf fwd obj CLCodomain[of T A e B]$ $absig zx$

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 $(Id (CanonicalCat' T) (dom_{CanonicalCat' T} f)) \ddot{::} CanonicalCat' T f = [A, sub Vx in e, B]_T$

using $f domf CLDomain[of T A e B]$ $idA axt CanonicalCompWellDefined[of T] Awd fwd zx$

by (*simp add: CanonicalCat'-def*)

thus $(Id (CanonicalCat' T) (dom_{CanonicalCat' T} f)) \ddot{::} CanonicalCat' T f = f$

using $f SubstXinE[of e]$ **by** *simp*

have $f \ddot{::} CanonicalCat' T (Id (CanonicalCat' T) (cod_{CanonicalCat' T} f)) = [A,$

```

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) (codCanonicalCat' T f)) = f
using f Subst-def by simp
}
{
  fix X assume a: X ∈ objCanonicalCat' T
  have X ∈ BaseTypes (aSignature T) using a by (simp add: CanonicalCat'-def)
  hence b: Sig (aSignature T) ▷ (Vx : X ⊢ Vx : X) by auto
  show Id (CanonicalCat' T) X mapsCanonicalCat' T X to X
  apply(rule MapsToI)
  apply(auto simp add: CanonicalCat'-def CLLDomain CLCodomain zx)
  apply(rule exI)+
  apply(auto simp add: b)
  done
}
{
  fix f g h assume a: f ≈CanonicalCat' T g and b: g ≈CanonicalCat' T h
  from a b have fm: f ∈ Mor (CanonicalCat' T) and gm: g ∈ Mor (CanonicalCat'
T)
  and hm: h ∈ 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)
▷ (Vx : A ⊢ 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) ▷ (Vx : B' ⊢ 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) ▷ (Vx : C' ⊢ eh : D)
  by (auto simp add: CanonicalCat'-def)
  from fg have Beq: B = B' using f g zx CLCodomain[of T A ef B] CLLDomain[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] CLLDo-
main[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 aST ▷ Vx : A ⊢ 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 aST ▷ Vx : B ⊢ 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) ▷ (Vx : X ⊢ 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) ▷ (Vx : Y ⊢ 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) ▷ (Vx : X ⊢ sub ef in eg : Z)
    using fwd gwd SubstWellDefined[of aST] 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 ∈ Mor (CanonicalCat' T) using fg fgwd by (auto
simp add: CanonicalCat'-def)
    show Cod (CanonicalCat' T) (CanonicalComp T f g) = Z
      using fg CLCdomain[of T] zx by (simp add: CanonicalCat'-def)
    qed
  thus f :: CanonicalCat' T g maps CanonicalCat' T X to Z by (simp add: Canoni-
calCat'-def)
}
qed

```

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

definition *CanonicalInterpretation where*
CanonicalInterpretation T ≡ (|
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 $CIObj$: $Obj\ (CanonicalCat\ T) = BaseType\ (aSignature\ T)$
by (*simp add: MakeCat-def CanonicalCat-def CanonicalCat'-def*)

lemma $CIMor$: $ZFAxioms\ T \implies [A, e, B]_T \in Mor\ (CanonicalCat\ T) = Sig\ (aSignature\ T) \triangleright (Vx : A \vdash e : B)$

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 $CIDom$: $\llbracket ZFAxioms\ T ; [A, e, B]_T \in Mor(CanonicalCat\ T) \rrbracket \implies Dom\ (CanonicalCat\ T)\ [A, e, B]_T = A$

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 $CICod$: $\llbracket ZFAxioms\ T ; [A, e, B]_T \in Mor(CanonicalCat\ T) \rrbracket \implies Cod\ (CanonicalCat\ T)\ [A, e, B]_T = B$

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 $CIId$: $\llbracket A \in BaseType\ (aSignature\ T) \rrbracket \implies Id\ (CanonicalCat\ T)\ A = [A, Vx, A]_T$

by(*simp add: MakeCat-def CanonicalCat-def CanonicalCat'-def*)

lemma $CIComp$:

assumes $ZFAxioms\ T$ **and** $Sig\ (aSignature\ T) \triangleright (Vx : A \vdash e : B)$ **and** $Sig\ (aSignature\ T) \triangleright (Vx : B \vdash d : C)$
shows $[A, e, B]_T \mathrel{::} CanonicalCat\ T\ [B, d, C]_T = [A, sub\ e\ in\ d, C]_T$
proof–
have $[A, e, B]_T \approx >_{CanonicalCat'\ T} [B, d, C]_T$
proof(*rule CompDefinedI*)
show $[A, e, B]_T \in Mor\ (CanonicalCat'\ T)$ **and** $[B, d, C]_T \in Mor\ (CanonicalCat'\ T)$
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 $[A, e, B]_T \mathrel{::} CanonicalCat\ T\ [B, d, C]_T = [A, e, B]_T \mathrel{::} CanonicalCat'\ T\ [B, d, C]_T$
by *(auto simp add: MakeCatComp CanonicalCat'-def)*
thus $[A, e, B]_T \mathrel{::} CanonicalCat\ T\ [B, d, C]_T = [A, sub\ e\ in\ d, C]_T$
using *CanonicalCompWellDefined[of T] assms by (simp add: CanonicalCat'-def)*
qed

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

lemma *CIInterpretation*: $ZFAxioms\ T \implies Interpretation\ (CI\ T)$
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** $A \in BaseTypes\ (iS_{CI}\ T)$ **thus** $Ty[A]_{CI}\ T \in Obj\ (iC_{CI}\ T)$
 by(*simp add: CanonicalInterpretation-def CIObj[of T]*)
}
{
 fix $f\ A\ B$ **assume** $a: f \in Sig\ iS_{CI}\ T : A \rightarrow B$
 hence $[sDom_{ISignature\ (CI\ T)}\ f, fE@Vx, sCod_{ISignature\ (CI\ T)}\ f]_T \in mor\ CanonicalCat\ T$
 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*: $ZFAxioms\ T \implies (\bigwedge B . Sig\ iS_{CI}\ T \triangleright (Vx : A \vdash e : B) \implies L[Vx : A \vdash e : B]_{CI}\ T \rightarrow (IMor\ [A, e, B]_T))$
proof(*induct e*)

```

assume xt: ZFAxioms T
{
  fix B assume a: Sig iSCI T ▷ Vx : A ⊢ Vx : B
  have aeqb: A = B using a SigId[of iSCI T A B] Interpretation.InterpExprWellDefined[of
CI T] by simp
  moreover have bt: A ∈ BaseTypes iSCI T using a SigTyId aeqb by simp
  ultimately have b: L[Vx : A ⊢ Vx : B]CI T → (IMor (Id iCCI T Ty[A]CI T))
by auto
  have Ty[A]CI T = A by (simp add: CanonicalInterpretation-def)
  hence (Id iCCI T Ty[A]CI T) = [A, Vx, A]T using bt CIId by (simp add:
CanonicalInterpretation-def)
  thus L[Vx : A ⊢ Vx : B]CI T → (IMor [A, Vx, B]T) using aeqb b by simp
}
{
  fix f e B assume ih: ∧B'. [[ZFAxioms T ; Sig iSCI T ▷ (Vx : A ⊢ e : B')]] ⇒
(L[Vx : A ⊢ e : B']CI T → (IMor [A, e, B']T))
  and a: Sig iSCI T ▷ Vx : A ⊢ f E@ e : B
  from a obtain B' where sig: Sig iSCI T ▷ (Vx : A ⊢ e : B') and f: f ∈ Sig
iSCI T : B' → B by auto
  hence L[Vx : A ⊢ e : B']CI T → (IMor [A, e, B']T) using ih xt by auto
  hence b: L[Vx : A ⊢ f E@ e : B]CI T → (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 iSCI T ▷ Vx : A ⊢ e : B' and Sig iSCI T ▷ Vx : B' ⊢ 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 ⊢ f E@ e : B]CI T → (IMor [A, f E@ e, B]T) using b by simp
}
qed

```

lemma CIModel: ZFAxioms T ⇒ 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: aST = iSCI T by (simp add: CanonicalInterpretation-def)
{
  fix φ assume a: φ ∈ aAxioms T
  from a Axioms.AxT obtain A B e d where φ: φ = Vx : A ⊢ e ≡ d : B and
sig: Sig aST ▷ Vx : A ⊢ e ≡ d : B
  using axt by blast

```


have $Sig\ aS_T \triangleright Vx : A \vdash e : B$ **using** *sig* **by** *auto*
hence $e : L[Vx : A \vdash e : B]_{CI\ T} \rightarrow (IMor\ [A, e, B]_T)$ **using** *aeq\ CIInterp2Mor*
xt **by** *simp*
have $Sig\ aS_T \triangleright Vx : A \vdash d : B$ **using** *sig* **by** *auto*
hence $d : L[Vx : A \vdash d : B]_{CI\ T} \rightarrow (IMor\ [A, d, B]_T)$ **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 $L[\varphi]_{CI\ T} \rightarrow (IBool\ True)$ **using** $e\ d\ \varphi$ *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** $L[\varphi]_{CI\ T} \rightarrow (IBool\ True)$ **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 $CI\ T\ \varphi\ True$] **obtain** $A\ B\ e\ d$ **where** $\varphi: \varphi = (Vx : A \vdash e \equiv d : B)$ **using** *ii\ assms* **by** *auto*
from *Interpretation.Equiv*[of $CI\ T\ A\ e\ d\ B$] **obtain** g
where $g1: L[Vx : A \vdash e : B]_{CI\ T} \rightarrow (IMor\ g)$
and $g2: L[Vx : A \vdash d : B]_{CI\ T} \rightarrow (IMor\ g)$ **using** *ii\ \varphi\ 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 $ie: L[Vx : A \vdash e : B]_{CI\ T} \rightarrow (IMor\ [A, e, B]_T)$ **using** ewd *CIInterp2Mor*
assms **by** *simp*
have $id: L[Vx : A \vdash d : B]_{CI\ T} \rightarrow (IMor\ [A, d, B]_T)$ **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 $[A, d, B]_T = g$
using $g2\ id$ *ii* *Interpretation.Functional*[of $CI\ T\ Vx : A \vdash d : B\ IMor\ [A, d, B]_T\ IMor\ g$] **by** *simp*
ultimately $[A, e, B]_T = [A, d, B]_T$ **by** *simp*
thus *?thesis* **using** $\varphi\ ewd\ Cl2Equiv\ aeq\ assms$ **by** *simp*
qed

lemma *Complete*:

assumes $ZFAxioms\ T$

and $\bigwedge (I :: (ZF, ZF, ZF, ZF)\ Interpretation) . Model\ I\ T \implies (L[\varphi]_I \rightarrow (IBool\ True))$

shows $\varphi \in Axioms.Theory\ T$

proof–

have *Model* ($CI\ T$) T **using** *assms\ CIModel* **by** *simp*

thus *?thesis* **using** *CIComplete*[of $T\ \varphi$] *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

$MakeFtor :: ('o1, 'o2, 'm1, 'm2, 'a, 'b, 'r) \text{ Functor-scheme} \Rightarrow ('o1, 'o2, 'm1, 'm2, 'a, 'b, 'r) \text{ Functor-scheme}$ **where**
 $MakeFtor F \equiv \{$
 $\quad CatDom = CatDom F,$
 $\quad CatCod = CatCod F,$
 $\quad MapM = restrict (MapM F) (Mor (CatDom F)),$
 $\quad \dots = Functor.more F$
 $\}$

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

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

definition

$functor-abbrev (Ftor - : - \longrightarrow - [81])$ **where**
 $Ftor F : A \longrightarrow B \equiv (Functor F) \wedge (CatDom F = A) \wedge (CatCod F = B)$

lemma $functor-abbrevE[elim]: \llbracket Ftor F : A \longrightarrow B ; \llbracket (Functor F) ; (CatDom F = A) ; (CatCod F = B) \rrbracket \rrbracket \Longrightarrow R \rrbracket \Longrightarrow R$
by ($auto$ $simp$ add: $functor-abbrev-def$)

definition

$functor-comp-def (- \approx > ; ; - [81])$ **where**
 $functor-comp-def F G \equiv (Functor F) \wedge (Functor G) \wedge (CatDom G = CatCod F)$

lemma $functor-comp-def[elim]: \llbracket F \approx > ; ; G ; \llbracket Functor F ; Functor G ; CatDom G = CatCod F \rrbracket \rrbracket \Longrightarrow R \rrbracket \Longrightarrow R$
by ($auto$ $simp$ add: $functor-comp-def-def$)

lemma (**in** $Functor$) $FunctorMapsTo$:

assumes $f \in mor_{CatDom F}$
shows $F \### f \text{ maps}_{CatCod F} (F \text{ @@ } (dom_{CatDom F} f)) \text{ to } (F \text{ @@ } (cod_{CatDom F} f))$
proof–
have $f \text{ maps}_{CatDom F} (dom_{CatDom F} f) \text{ to } (cod_{CatDom F} f)$ **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 $f \in mor_{CatDom F}$
shows $dom_{CatCod F} (F \### f) = F \text{ @@ } (dom_{CatDom F} f)$ **and** $cod_{CatCod F} (F \### f) = F \text{ @@ } (cod_{CatDom F} f)$
proof–
have $F \### f \text{ maps}_{CatCod F} (F \text{ @@ } (dom_{CatDom F} f)) \text{ to } (F \text{ @@ } (cod_{CatDom F} f))$ **using** $assms$ **by** ($simp$ add: $FunctorMapsTo$)
thus $dom_{CatCod F} (F \### f) = F \text{ @@ } (dom_{CatDom F} f)$ **and** $cod_{CatCod F} (F \### f) = F \text{ @@ } (cod_{CatDom F} f)$

$\#\# f = F \text{ @@ } (\text{cod}_{\text{CatDom } F} f)$
 by *auto*
 qed

lemma (in *Functor*) *FunctorCompPreserved*: $f \in \text{mor}_{\text{CatDom } F} \implies F \#\# f \in \text{mor}_{\text{CatCod } F}$
 by (auto dest: *FunctorMapsTo*)

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

lemma *FunctorComp*: $\llbracket \text{Ftor } F : A \longrightarrow B ; f \approx_{> A} g \rrbracket \implies F \#\# (f ;;_A g) = (F \#\# f) ;;_B (F \#\# g)$
 by (auto simp add: *PreFunctor.FunctorComp*)

lemma *FunctorCompDef*: $\llbracket \text{Ftor } F : A \longrightarrow B ; f \approx_{> A} g \rrbracket \implies (F \#\# f) \approx_{> B} (F \#\# g)$
 by (auto simp add: *Functor.FunctorCompDef*)

lemma *FunctorMapsTo*:
 assumes $\text{Ftor } F : A \longrightarrow B$ and $f \text{ maps}_A X \text{ to } Y$
 shows $(F \#\# f) \text{ maps}_B (F \text{ @@ } X) \text{ to } (F \text{ @@ } Y)$
proof–
 have $b: \text{CatCod } F = B$ and $a: \text{CatDom } F = A$ and $\text{ff}: \text{Functor } F$ using *assms*
 by *auto*
 have $\text{df}: (\text{dom}_{\text{CatDom } F} f) = X$ and $\text{cf}: (\text{cod}_{\text{CatDom } F} f) = Y$ using *a assms*
 by *auto*
 have $f \in \text{mor}_{\text{CatDom } F}$ using *assms* by *auto*
 hence $F \#\# f \text{ maps}_{\text{CatCod } F} (F \text{ @@ } (\text{dom}_{\text{CatDom } F} f)) \text{ to } (F \text{ @@ } (\text{cod}_{\text{CatDom } F} f))$ using *ff*
 by (simp add: *Functor.FunctorMapsTo*)
 thus *?thesis* using *df cf b* by *simp*
 qed

lemma (in *PreFunctor*) *FunctorId2*:
 assumes $X \in \text{obj}_{\text{CatDom } F}$
 shows $F \text{ @@ } X \in \text{obj}_{\text{CatCod } F} \wedge F \#\# (\text{id}_{\text{CatDom } F} X) = \text{id}_{\text{CatCod } F} (F \text{ @@ } X)$

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

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

lemma (**in** *Functor*) *DomFunctor*: $f \in \text{mor}_{\text{CatDom } F} \implies \text{dom}_{\text{CatCod } F} (F \#\# f) = F \@\@ (\text{dom}_{\text{CatDom } F} f)$
by (*simp add: FunctorCodDom*)

lemma (**in** *Functor*) *CodFunctor*: $f \in \text{mor}_{\text{CatDom } F} \implies \text{cod}_{\text{CatCod } F} (F \#\# f) = F \@\@ (\text{cod}_{\text{CatDom } F} f)$
by (*simp add: FunctorCodDom*)

lemma (**in** *Functor*) *FunctorId3Dom*:
assumes $f \in \text{mor}_{\text{CatDom } F}$
shows $F \#\# (\text{id}_{\text{CatDom } F} (\text{dom}_{\text{CatDom } F} f)) = \text{id}_{\text{CatCod } F} (\text{dom}_{\text{CatCod } F} (F \#\# f))$
proof-
have $(\text{dom}_{\text{CatDom } F} f) \in \text{obj}_{\text{CatDom } F}$ **using** *assms* **by** (*simp add: Category.Cdom*)
hence $F \#\# (\text{id}_{\text{CatDom } F} (\text{dom}_{\text{CatDom } F} f)) = \text{id}_{\text{CatCod } F} (F \@\@ (\text{dom}_{\text{CatDom } F} f))$ **by** (*simp add: FunctorId2*)
also have $\dots = \text{id}_{\text{CatCod } F} (\text{dom}_{\text{CatCod } F} (F \#\# f))$ **using** *assms* **by** (*simp add: DomFunctor*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *Functor*) *FunctorId3Cod*:
assumes $f \in \text{mor}_{\text{CatDom } F}$
shows $F \#\# (\text{id}_{\text{CatDom } F} (\text{cod}_{\text{CatDom } F} f)) = \text{id}_{\text{CatCod } F} (\text{cod}_{\text{CatCod } F} (F \#\# f))$
proof-

have $(\text{cod}_{\text{CatDom } F} f) \in \text{obj}_{\text{CatDom } F}$ **using** *assms* **by** (*simp add: Category.Ccod*)
hence $F \#\# (\text{id}_{\text{CatDom } F} (\text{cod}_{\text{CatDom } F} f)) = \text{id}_{\text{CatCod } F} (F \@\@ (\text{cod}_{\text{CatDom } F} f))$ **by** (*simp add: FunctorId2*)
also have $\dots = \text{id}_{\text{CatCod } F} (\text{cod}_{\text{CatCod } F} (F \#\# f))$ **using** *assms* **by** (*simp add: CodFunctor*)
finally show *?thesis* **by** *simp*
qed

lemma (*in PreFunctor*) *FmToFo*: $\llbracket X \in \text{obj}_{\text{CatDom } F} ; Y \in \text{obj}_{\text{CatCod } F} ; F \#\# (\text{id}_{\text{CatDom } F} X) = \text{id}_{\text{CatCod } F} Y \rrbracket \implies F \@\@ X = Y$
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** $a: X \in \text{obj}_{\text{CatDom } F}$ **have** $\text{id}_{\text{CatDom } F} X \in \text{mor}_{\text{CatDom } F}$
proof–
have *Category (CatDom F)* **using** *assms* **by** (*simp add: PreFunctor-def*)
hence $\text{id}_{\text{CatDom } F} X$ *maps* $\text{CatDom } F$ 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*: $f \in \text{mor}_{\text{CatDom } F} \implies \text{MakeFtor } F \#\# f = F \#\# f$
by (*simp add: MakeFtor-def*)

lemma *MakeFtorObj*:

assumes *PreFunctor F* **and** $X \in \text{obj}_{\text{CatDom } F}$

shows $\text{MakeFtor } F \@\@ X = F \@\@ X$

proof–

have $X \in \text{obj}_{\text{CatDom } (\text{MakeFtor } F)}$ **using** *assms(2)* **by** (*simp add: MakeFtor-def*)
moreover have $(F \@\@ X) \in \text{obj}_{\text{CatCod } (\text{MakeFtor } F)}$ **using** *assms* **by** (*simp add: PreFunctor.FunctorId2 MakeFtor-def*)
moreover have $\text{MakeFtor } F \#\# \text{id}_{\text{CatDom } (\text{MakeFtor } F)} X = \text{id}_{\text{CatCod } (\text{MakeFtor } F)} (F \@\@ X)$
proof–
have *Category (CatDom F)* **using** *assms(1)* **by** (*simp add: PreFunctor-def*)
hence $\text{id}_{\text{CatDom } F} X$ *maps* $\text{CatDom } F$ X *to* X **using** *assms(2)* **by** (*auto simp add: Category.Cidm*)
hence $\text{id}_{\text{CatDom } F} X \in \text{mor}_{\text{CatDom } F}$ **by** *auto*
hence $\text{MakeFtor } F \#\# \text{id}_{\text{CatDom } (\text{MakeFtor } F)} X = F \#\# \text{id}_{\text{CatDom } F} X$ **by** (*simp add: MakeFtor-def*)
also have $\dots = \text{id}_{\text{CatCod } F} (F \@\@ X)$ **using** *assms* **by** (*simp add: PreFunctor.FunctorId2*)

finally show *?thesis* **by** (*simp add: MakeFtor-def*)
qed
moreover have *PreFunctor (MakeFtor F)* **using** *assms(1)* **by** (*simp add: MakeFtorPreFtor*)
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 $((\text{MakeFtor } F) \#\# f) \text{ maps}_{\text{CatCod}} (\text{MakeFtor } F) ((\text{MakeFtor } F) \@\@ X) \text{ to } ((\text{MakeFtor } F) \@\@ Y)$
proof-
have $((\text{MakeFtor } F) \#\# f) = F \#\# f$ **using** *aa* **by** (*auto simp add: MakeFtor-def*)
moreover have $((\text{MakeFtor } F) \@\@ X) = F \@\@ X$ **and** $((\text{MakeFtor } F) \@\@ Y) = F \@\@ Y$
proof-
have *Category (CatDom F)* **using** *assms* **by** (*simp add: FunctorM-def PreFunctor-def*)
hence $X \in \text{obj}_{\text{CatDom } F}$ **and** $Y \in \text{obj}_{\text{CatDom } F}$
using *aa* **by** (*auto simp add: Category.MapsToObj MakeFtor-def*)
moreover have *PreFunctor F* **using** *assms(1)* **by** (*simp add: FunctorM-def*)
ultimately show $((\text{MakeFtor } F) \@\@ X) = F \@\@ X$ **and** $((\text{MakeFtor } F) \@\@ Y) = F \@\@ Y$
by (*simp add: MakeFtorObj*)
qed
moreover have $F \#\# f \text{ maps}_{\text{CatCod}} F (F \@\@ X) \text{ to } (F \@\@ Y)$ **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 \Rightarrow ('o,'o,'m,'m,'a,'a) Functor (FId' - [70]) **where**
IdentityFunctor' C \equiv ($\text{CatDom} = C$, $\text{CatCod} = C$, $\text{MapM} = (\lambda f . f)$)

definition

IdentityFunctor (FId - [70]) **where**

$IdentityFunctor\ C \equiv MakeFtor(IdentityFunctor'\ C)$

lemma $IdFtor'PreFunctor$: $Category\ C \implies PreFunctor\ (FId'\ C)$
by(*auto simp add: PreFunctor-def IdentityFunctor'-def*)

lemma $IdFtor'Obj$:

assumes $Category\ C$ **and** $X \in obj_{CatDom}\ (FId'\ C)$

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 $X \in obj_{CatCod}\ (FId'\ C)$ **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 $X \in obj_{CatDom}\ (FId'\ C)$ **and** $Y \in obj_{CatDom}\ (FId'\ C)$

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 $IdFtorFtor$: $Category\ C \implies Functor\ (FId'\ C)$

by (*auto simp add: IdentityFunctor-def IdFtor'FtorM intro: MakeFtor*)

definition

$ConstFunctor' :: ('o1, 'm1, 'a)\ Category\ scheme \implies$

$('o2, 'm2, 'b)\ Category\ scheme \implies 'o2 \implies ('o1, 'o2, 'm1, 'm2, 'a, 'b)\ Functor$

where

$ConstFunctor'\ A\ B\ b \equiv \langle$

$CatDom = A$,

$CatCod = B$,

$MapM = (\lambda f . (Id\ B)\ b)$

\rangle

definition $ConstFunctor\ A\ B\ b \equiv MakeFtor(ConstFunctor'\ A\ B\ b)$

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 Fun-*
ctorM-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 = λ *f* . (*MapM G*)((*MapM F*) *f*)

)

definition *FunctorComp* (**infixl** $::;$ 71) **where** $\text{FunctorComp } F \ G \equiv \text{MakeFtor } (\text{FunctorComp}' \ F \ G)$

lemma *FtorCompComp'*:

assumes $f \approx_{>} \text{CatDom } F \ g$

and $F \approx_{>} ::; G$

shows $G \ ## \ (F \ ## \ (f \ ;_{\text{CatDom } F} \ g)) = (G \ ## \ (F \ ## \ f)) \ ;_{\text{CatCod } G} \ (G \ ## \ (F \ ## \ g))$

proof–

have [*simp*]: $\text{PreFunctor } G \wedge \text{PreFunctor } F$ **using** *assms* **by** *auto*

have [*simp*]: $(F \ ## \ f) \approx_{>} \text{CatDom } G \ (F \ ## \ g)$ **using** *assms* **by** (*auto simp add: Functor.FunctorCompDef*[of $F \ f \ g$])

have $F \ ## \ (f \ ;_{\text{CatDom } F} \ g) = (F \ ## \ f) \ ;_{\text{CatCod } F} \ (F \ ## \ g)$ **using** *assms* **by** (*auto simp add: PreFunctor.FunctorComp*)

hence $G \ ## \ (F \ ## \ (f \ ;_{\text{CatDom } F} \ g)) = G \ ## \ ((F \ ## \ f) \ ;_{\text{CatCod } F} \ (F \ ## \ g))$ **by** *simp*

also have $\dots = G \ ## \ ((F \ ## \ f) \ ;_{\text{CatDom } G} \ (F \ ## \ g))$ **using** *assms* **by** *auto*

also have $\dots = (G \ ## \ (F \ ## \ f)) \ ;_{\text{CatCod } G} \ (G \ ## \ (F \ ## \ g))$

by (*simp add: PreFunctor.FunctorComp*[of $G \ (F \ ## \ f) \ (F \ ## \ g)$])

finally show *?thesis* **by** *simp*

qed

lemma *FtorCompId*:

assumes $a: X \in (\text{Obj } (\text{CatDom } F))$

and $F \approx_{>} ::; G$

shows $G \ ## \ (F \ ## \ (\text{id}_{\text{CatDom } F} \ X)) = \text{id}_{\text{CatCod } G} (G \ @\@ \ (F \ @\@ \ X)) \wedge G \ @\@ \ (F \ @\@ \ X) \in (\text{Obj } (\text{CatCod } G))$

proof–

have [*simp*]: $\text{PreFunctor } G \wedge \text{PreFunctor } F$ **using** *assms* **by** *auto*

have $b: (F \ @\@ \ X) \in \text{obj}_{\text{CatDom } G}$ **using** *assms*

by (*auto simp add: PreFunctor.FunctorId2*)

have $G \ ## \ F \ ## \ (\text{id}_{\text{CatDom } F} \ X) = G \ ## \ (\text{id}_{\text{CatCod } F} (F \ @\@ \ X))$ **using** $b \ a$ **by** (*simp add: PreFunctor.FunctorId2*[of $F \ X$])

also have $\dots = G \ ## \ (\text{id}_{\text{CatDom } G} (F \ @\@ \ X))$ **using** *assms* **by** *auto*

also have $\dots = \text{id}_{\text{CatCod } G} (G \ @\@ \ (F \ @\@ \ X)) \wedge G \ @\@ \ (F \ @\@ \ X) \in (\text{Obj } (\text{CatCod } G))$ **using** b

by (*simp add: PreFunctor.FunctorId2*[of $G \ (F \ @\@ \ X)$])

finally show *?thesis* **by** *simp*

qed

lemma *FtorCompIdDef*:

assumes $a: X \in (\text{Obj } (\text{CatDom } F))$ **and** $b: \text{PreFunctor } (F \ ;; G)$

and $F \approx_{>} ::; G$

shows $(F \ ;; G) \ @\@ \ X = (G \ @\@ \ (F \ @\@ \ X))$

proof–

have $(F \ ;; G) \ ## \ (\text{id}_{\text{CatDom } (F \ ;; G)} (X)) = G \ ## \ (F \ ## \ (\text{id}_{\text{CatDom } F} (X)))$

using *assms*

by (simp add: FunctorComp'-def)
 also have $\dots = id_{CatCod\ G}(G\ @\@ (F\ @\@ (X)))$ 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: FunctorComp'-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: FunctorComp'-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 $(F \;;\; G) \text{ @@ } (\text{dom}_{\text{CatDom}}(F \;;\; G) f) = (G \text{ @@ } (F \text{ @@ } (\text{dom}_{\text{CatDom}}(F \;;\; G) f))) \wedge$
 $(F \;;\; G) \text{ @@ } (\text{cod}_{\text{CatDom}}(F \;;\; G) f) = (G \text{ @@ } (F \text{ @@ } (\text{cod}_{\text{CatDom}}(F \;;\; G) f)))$
by (*auto simp add: FtorCompIdDef[of - F G] 1 assms*)
ultimately show *?thesis* **by auto**
qed

lemma *FunctorCompMapsTo3*:

assumes *f maps* $\text{CatDom}(F \;;\; G)$ *X to Y*
and $F \approx > \;;\; G$
and *PreFunctor* $(F \;;\; G)$
shows $F \;;\; G \text{ ## } f \text{ maps } \text{CatCod}(F \;;\; G) F \;;\; G \text{ @@ } X \text{ to } F \;;\; G \text{ @@ } Y$

proof–

have $f \in \text{mor}_{\text{CatDom}}(F \;;\; G)$
and $\text{dom}_{\text{CatDom}}(F \;;\; G) f = X$
and $\text{cod}_{\text{CatDom}}(F \;;\; G) f = Y$ **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 $1: \text{PreFunctor } G \wedge \text{PreFunctor } F$ **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* $(\text{CatDom } F)$ **and** *Category* $(\text{CatCod } G)$ **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 $1: \text{PreFunctor}(F \;;\; G)$ **using** *assms* **by** (*rule FtorCompPreFtor*)

{
fix $X Y f$ **assume** *a: f maps* $\text{CatDom}(F \;;\; G)$ *X to Y*
have $F \;;\; G \text{ ## } f \text{ maps } \text{CatCod}(F \;;\; G) F \;;\; G \text{ @@ } X \text{ to } F \;;\; G \text{ @@ } Y$
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  $\text{Functor } (F ; ; G)$ 
proof-
  have  $\text{FunctorM } (F ; ; G)$  using assms by (rule FtorCompM)
  thus ?thesis by (simp add: FunctorComp-def MakeFtor)
qed

lemma (in Functor) FunctorPreservesIso:
  assumes  $\text{ciso } \text{CatDom } F \ k$ 
  shows  $\text{ciso } \text{CatCod } F \ (F \ \#\# \ k)$ 
proof-
  have [simp]:  $k \in \text{mor } \text{CatDom } F$  using assms by (simp add: Category.IsoIsMor)
  have  $\text{cinv } \text{CatCod } F \ (F \ \#\# \ k) \ (F \ \#\# \ (\text{Cinv } \text{CatDom } F \ k))$ 
  proof(rule Category.Inverse-relI)
    show Category (CatCod F) by simp
    show  $(F \ \#\# \ k) \approx > \text{CatCod } F \ (F \ \#\# \ (\text{Cinv } \text{CatDom } F \ k))$ 
      by (rule FunctorCompDef, simp add: Category.IsoCompInv assms)
    show  $(F \ \#\# \ k) ; ; \text{CatCod } F \ (F \ \#\# \ (\text{Cinv } \text{CatDom } F \ k)) = \text{id}_{\text{CatCod } F}$ 
      ( $\text{dom } \text{CatCod } F \ (F \ \#\# \ k)$ )
    proof-
      have  $(F \ \#\# \ k) ; ; \text{CatCod } F \ (F \ \#\# \ (\text{Cinv } \text{CatDom } F \ k)) = F \ \#\# \ (k$ 
      ; ;  $\text{CatDom } F \ (\text{Cinv } \text{CatDom } F \ k))$  using assms
      by(auto simp add: FunctorComp Category.IsoCompInv)
      also have  $\dots = F \ \#\# \ (\text{id}_{\text{CatDom } F} \ (\text{dom } \text{CatDom } F \ k))$  using assms by
      (simp add: Category.IsoInvId2)
      also have  $\dots = \text{id}_{\text{CatCod } F} \ (\text{dom } \text{CatCod } F \ (F \ \#\# \ k))$  by (simp add:
      FunctorId3Dom)
      finally show ?thesis by simp
    qed
    show  $(F \ \#\# \ (\text{Cinv } \text{CatDom } F \ k)) ; ; \text{CatCod } F \ (F \ \#\# \ k) = \text{id}_{\text{CatCod } F}$ 
      ( $\text{cod } \text{CatCod } F \ (F \ \#\# \ k)$ )
    proof-
      have  $(F \ \#\# \ (\text{Cinv } \text{CatDom } F \ k)) ; ; \text{CatCod } F \ (F \ \#\# \ k) = F \ \#\#$ 
      ( $(\text{Cinv } \text{CatDom } F \ k) ; ; \text{CatDom } F \ k)$  using assms
      by(auto simp add: FunctorComp Category.InvCompIso)
      also have  $\dots = F \ \#\# \ (\text{id}_{\text{CatDom } F} \ (\text{cod } \text{CatDom } F \ k))$  using assms by
      (simp add: Category.IsoInvId1)
      also have  $\dots = \text{id}_{\text{CatCod } F} \ (\text{cod } \text{CatCod } F \ (F \ \#\# \ k))$  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]:  $\text{Functor } F \implies \text{FunctorM } F$ 

```

by (simp add: Functor-def)

```

locale Equivalence = Functor +
  assumes Full:  $\llbracket A \in \text{Obj} (\text{CatDom } F) ; B \in \text{Obj} (\text{CatDom } F) ;$ 
     $h \text{ maps}_{\text{CatCod } F} (F \text{ @@ } A) \text{ to } (F \text{ @@ } B) \rrbracket \implies$ 
     $\exists f . (f \text{ maps}_{\text{CatDom } F} A \text{ to } B) \wedge (F \text{ ## } f = h)$ 
  and Faithful:  $\llbracket f \text{ maps}_{\text{CatDom } F} A \text{ to } B ; g \text{ maps}_{\text{CatDom } F} A \text{ to } B ; F \text{ ## } f =$ 
 $F \text{ ## } g \rrbracket \implies f = g$ 
  and IsoDense:  $C \in \text{Obj} (\text{CatCod } F) \implies \exists A \in \text{Obj} (\text{CatDom } F) . \text{ObjIso}$ 
 $(\text{CatCod } F) (F \text{ @@ } A) C$ 
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  $\eta$  X  $\equiv$  (NatTransMap  $\eta$ ) X

```

definition NTCatDom $\eta \equiv \text{CatDom} (\text{NTDom } \eta)$

definition NTCatCod $\eta \equiv \text{CatCod} (\text{NTCod } \eta)$

locale NatTransExt =

```

  fixes  $\eta :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{NatTrans}$  (structure)
  assumes NTExt : NatTransMap  $\eta \in \text{extensional} (\text{Obj} (\text{NTCatDom } \eta))$ 

```

locale NatTransP =

```

  fixes  $\eta :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{NatTrans}$  (structure)
  assumes NatTransFtor: Functor (NTDom  $\eta$ )
  and NatTransFtor2: Functor (NTCod  $\eta$ )
  and NatTransFtorDom: NTCatDom  $\eta = \text{CatDom} (\text{NTCod } \eta)$ 
  and NatTransFtorCod: NTCatCod  $\eta = \text{CatCod} (\text{NTDom } \eta)$ 
  and NatTransMapsTo:  $X \in \text{obj}_{\text{NTCatDom } \eta} \implies$ 
     $(\eta \text{ $$ } X) \text{ maps}_{\text{NTCatCod } \eta} ((\text{NTDom } \eta) \text{ @@ } X) \text{ to } ((\text{NTCod}$ 
 $\eta) \text{ @@ } X)$ 
  and NatTrans:  $f \text{ maps}_{\text{NTCatDom } \eta} X \text{ to } Y \implies$ 
     $((\text{NTDom } \eta) \text{ ## } f) ;;_{\text{NTCatCod } \eta} (\eta \text{ $$ } Y) = (\eta \text{ $$ } X) ;;_{\text{NTCatCod } \eta}$ 
 $((\text{NTCod } \eta) \text{ ## } f)$ 

```

locale $NatTrans = NatTransP + NatTransExt$

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

definition $MakeNT :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans \implies ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans$ **where**

$MakeNT \eta \equiv$ (\langle
 $NTDom = NTDom \eta$,
 $NTCod = NTCod \eta$,
 $NatTransMap = restrict (NatTransMap \eta) (Obj (NTCatDom \eta))$
 \rangle)

definition

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

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

lemma $MakeNT$: $NatTransP \eta \implies NatTrans (MakeNT \eta)$

by ($auto$ $simp$ add : $NatTransP-def$ $NatTrans-def$ $MakeNT-def$ $NTCatDom-def$ $NTCod-def$ $Category.MapsToObj$ $NatTransExt-def$)

lemma $MakeNT-comp$: $X \in Obj (NTCatDom f) \implies (MakeNT f) \$\$ X = f \$\$ X$
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 $MakeNTApp$: $X \in Obj (NTCatDom (MakeNT f)) \implies f \$\$ X = (MakeNT f) \$\$ X$

by ($simp$ add : $MakeNT-def$ $NTCatDom-def$)

lemma $NatTransMapsTo$:

assumes $NT \eta : F \implies G$ **and** $X \in Obj (CatDom F)$

shows $\eta \$\$ X$ $maps_{CatCod} G (F @@@ X)$ $to (G @@@ X)$

proof–

have NTP : $NatTransP \eta$ **using** $assms$ **by** $auto$

have NTC : $NTCatCod \eta = CatCod G$ **using** $assms$ **by** ($auto$ $simp$ add : $NTCatCod-def$)

have NTD : $NTCatDom \eta = CatDom F$ **using** $assms$ **by** ($auto$ $simp$ add : $NTCatDom-def$)

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

definition

$NTCompDefined :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) \ NatTrans$
 $\Rightarrow ('o1, 'o2, 'm1, 'm2, 'a, 'b) \ NatTrans \Rightarrow bool$ (**infixl** $\approx>\cdot$ 65)

where

$NTCompDefined \ \eta1 \ \eta2 \equiv NatTrans \ \eta1 \wedge NatTrans \ \eta2 \wedge NTCatDom \ \eta2 = NTCatDom \ \eta1 \wedge$

$NTCatCod \ \eta2 = NTCatCod \ \eta1 \wedge NTCod \ \eta1 = NTDom \ \eta2$

lemma $NTCompDefinedE[elim]: \llbracket \eta1 \approx>\cdot \eta2 ; \llbracket NatTrans \ \eta1 ; NatTrans \ \eta2 ; NTCatDom \ \eta2 = NTCatDom \ \eta1 ;$

$NTCatCod \ \eta2 = NTCatCod \ \eta1 ; NTCod \ \eta1 = NTDom \ \eta2 \rrbracket$

$\implies R \rrbracket \implies R$

by (*simp add: NTCompDefined-def*)

lemma $NTCompDefinedI: \llbracket NatTrans \ \eta1 ; NatTrans \ \eta2 ; NTCatDom \ \eta2 = NTCatDom \ \eta1 ;$

$NTCatCod \ \eta2 = NTCatCod \ \eta1 ; NTCod \ \eta1 = NTDom \ \eta2 \rrbracket$

$\implies \eta1 \approx>\cdot \eta2$

by (*simp add: NTCompDefined-def*)

lemma $NatTransExt0:$

assumes $NTDom \ \eta1 = NTDom \ \eta2$ **and** $NTCod \ \eta1 = NTCod \ \eta2$

and $\bigwedge X . X \in Obj (NTCatDom \ \eta1) \implies \eta1 \ \$\$ X = \eta2 \ \$\$ X$

and $NatTransMap \ \eta1 \in extensional (Obj (NTCatDom \ \eta1))$

and $NatTransMap \ \eta2 \in extensional (Obj (NTCatDom \ \eta2))$

shows $\eta1 = \eta2$

proof–

have $NatTransMap \ \eta1 = NatTransMap \ \eta2$

proof(*rule extensionalityI [of NatTransMap $\eta1$ Obj (NTCatDom $\eta1$)*])

show $NatTransMap \ \eta1 \in extensional (Obj (NTCatDom \ \eta1))$ **using** *assms by simp*

have $NTCatDom \ \eta1 = NTCatDom \ \eta2$ **using** *assms by (simp add: NTCatDom-def)*

moreover **have** $NatTransMap \ \eta2 \in extensional (Obj (NTCatDom \ \eta2))$ **using** *assms by simp*

ultimately **show** $NatTransMap \ \eta2 \in extensional (Obj (NTCatDom \ \eta1))$ **by** *simp*

{fix X **assume** $X \in Obj (NTCatDom \ \eta1)$ **thus** $\eta1 \ \$\$ X = \eta2 \ \$\$ X$ **using** *assms by simp***}**

qed

thus *?thesis using assms by (simp)*

qed

lemma *NatTransExt'*:

assumes $NTDom \eta1' = NTDom \eta2'$ **and** $NTCod \eta1' = NTCod \eta2'$

and $\bigwedge X . X \in Obj (NTCatDom \eta1') \implies \eta1' \$\$ X = \eta2' \$\$ X$

shows $MakeNT \eta1' = MakeNT \eta2'$

proof(*rule NatTransExt0*)

show $NatTransMap (MakeNT \eta1') \in extensional (Obj (NTCatDom (MakeNT \eta1')))$ **and**

$NatTransMap (MakeNT \eta2') \in extensional (Obj (NTCatDom (MakeNT \eta2')))$

using *assms*

by(*simp add: MakeNT-def NTCatDom-def NTCatCod-def NatTransExt-def*)**+**

show $NTDom (MakeNT \eta1') = NTDom (MakeNT \eta2')$ **and**

$NTCod (MakeNT \eta1') = NTCod (MakeNT \eta2')$ **using** *assms* **by** (*simp add: MakeNT-def*)**+**

{

fix X **assume** $1: X \in Obj (NTCatDom (MakeNT \eta1'))$

show $(MakeNT \eta1') \$\$ X = (MakeNT \eta2') \$\$ X$

proof–

have $NTCatDom (MakeNT \eta1') = NTCatDom (MakeNT \eta2')$ **using** *assms*
by(*simp add: NTCatDom-def MakeNT-def*)

hence $2: X \in Obj (NTCatDom (MakeNT \eta2'))$ **using** 1 **by** *simp*

have $(NTCatDom \eta1') = (NTCatDom (MakeNT \eta1'))$ **by** (*rule MakeNT-dom*)

hence $X \in Obj (NTCatDom \eta1')$ **using** 1 *assms* **by** *simp*

hence $\eta1' \$\$ X = \eta2' \$\$ X$ **using** *assms* **by** *simp*

moreover **have** $\eta1' \$\$ X = (MakeNT \eta1') \$\$ X$ **using** 1 *assms* **by** (*simp add: MakeNTApp*)

moreover **have** $\eta2' \$\$ X = (MakeNT \eta2') \$\$ X$ **using** 2 *assms* **by** (*simp add: MakeNTApp*)

ultimately **have** $(MakeNT \eta1') \$\$ X = (MakeNT \eta2') \$\$ X$ **by** *simp*

thus *?thesis* **using** *assms* **by** *simp*

qed

}

qed

lemma *NatTransExt*:

assumes $NatTrans \eta1$ **and** $NatTrans \eta2$ **and** $NTDom \eta1 = NTDom \eta2$ **and**
 $NTCod \eta1 = NTCod \eta2$

and $\bigwedge X . X \in Obj (NTCatDom \eta1) \implies \eta1 \$\$ X = \eta2 \$\$ X$

shows $\eta1 = \eta2$

proof–

have $NatTransMap \eta1 \in extensional (Obj (NTCatDom \eta1))$ **and**

$NatTransMap \eta2 \in extensional (Obj (NTCatDom \eta2))$ **using** *assms*

by(*simp only: NatTransExt-def NatTrans-def*)**+**

thus *?thesis* **using** *assms* **by** (*simp add: NatTransExt0*)

qed

definition

$IdNatTrans' :: ('o1, 'o2, 'm1, 'm2, 'a1, 'a2) Functor \Rightarrow ('o1, 'o2, 'm1, 'm2, 'a1,$

'a2) *NatTrans* **where**
 $IdNatTrans' F \equiv ()$
 $NTDom = F$,
 $NTCod = F$,
 $NatTransMap = \lambda X . id_{CatCod F} (F @@ X)$
 $\})$

definition $IdNatTrans F \equiv MakeNT(IdNatTrans' F)$

lemma $IdNatTrans-map: X \in obj_{CatDom F} \implies (IdNatTrans F) \$\$ X = id_{CatCod F} (F @@ X)$

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 $IdNatTransNatTrans': Functor F \implies NatTransP(IdNatTrans' F)$

proof(*auto simp add: NatTransP-def IdNatTrans'-def NTCatDom-def NTCatCod-def Category.Simps*)

PreFunctor.FunctorId2 functor-simps Functor.FunctorMapsTo)
 $\{$
 $\text{fix } f X Y$
 $\text{assume } a: \text{Functor } F \text{ and } b: f \text{ maps}_{CatDom F} X \text{ to } Y$
 $\text{show } (F \#\# f) \::_{CatCod F} (id_{CatCod F} (F @@ Y)) = (id_{CatCod F} (F @@ X))$
 $\::_{CatCod F} (F \#\# f)$
proof-
 $\text{have } 1: \text{Category } (CatCod F) \text{ using } a \text{ by simp}$
 $\text{have } F \#\# f \text{ maps}_{CatCod F} (F @@ X) \text{ to } (F @@ Y) \text{ using } a b \text{ by (auto simp add: Functor.FunctorMapsTo)}$
 $\text{hence } 2: F \#\# f \in mor_{CatCod F} \text{ and } 3: cod_{CatCod F} (F \#\# f) = (F @@ Y)$
 $\text{and } 4: dom_{CatCod F} (F \#\# f) = (F @@ X) \text{ by auto}$
 $\text{have } (F \#\# f) \::_{CatCod F} (id_{CatCod F} (F @@ Y)) = (F \#\# f) \::_{CatCod F} (id_{CatCod F} (cod_{CatCod F} (F \#\# f)))$
 $\text{using } 3 \text{ by simp}$
 $\text{also have } \dots = F \#\# f \text{ using } 1 2 \text{ by (auto simp add: Category.Cidr)}$
 $\text{also have } \dots = (id_{CatCod F} (dom_{CatCod F} (F \#\# f))) \::_{CatCod F} (F \#\# f)$
 $\text{using } 1 2 \text{ by (auto simp add: Category.Cidl)}$
 $\text{also have } \dots = (id_{CatCod F} (F @@ X)) \::_{CatCod F} (F \#\# f) \text{ using } 4 \text{ by simp}$
 $\text{finally show ?thesis .}$
 qed
 $\}$
qed

lemma $IdNatTransNatTrans: Functor F \implies NatTrans (IdNatTrans F)$

by (*simp add: IdNatTransNatTrans' IdNatTrans-def MakeNT*)

definition

```

NatTransComp' :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans =>
  ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans =>
  ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans (infixl .1 75) where
NatTransComp' η1 η2 = ()
  NTDom = NTDom η1 ,
  NTCod = NTCod η2 ,
  NatTransMap = λ X . (η1 $$ X) ;;NTCatCod η1 (η2 $$ X)
)

```

definition *NatTransComp* (infixl . 75) where $\eta1 \cdot \eta2 \equiv \text{MakeNT}(\eta1 \cdot 1 \eta2)$

lemma *NatTransComp-Comp1*: $\llbracket x \in \text{Obj}(\text{NTCatDom } f) ; f \approx \triangleright \cdot g \rrbracket \implies (f \cdot g) \text{ $$ } x = (f \text{ $$ } x) ;; \text{NTCatCod } g (g \text{ $$ } x)$

by(*auto simp add: NatTransComp-def NatTransComp'-def MakeNT-def NTCatCod-def NTCatDom-def*)

lemma *NatTransComp-Comp2*: $\llbracket x \in \text{Obj}(\text{NTCatDom } f) ; f \approx \triangleright \cdot g \rrbracket \implies (f \cdot g) \text{ $$ } x = (f \text{ $$ } x) ;; \text{NTCatCod } f (g \text{ $$ } 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*]: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NatTrans } \eta1$ **by** *auto*

lemma [*simp*]: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NatTrans } \eta2$ **by** *auto*

lemma *NTCatDom*: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NTCatDom } \eta1 = \text{NTCatDom } \eta2$ **by** *auto*

lemma *NTCatCod*: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NTCatCod } \eta1 = \text{NTCatCod } \eta2$ **by** *auto*

lemma [*simp*]: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NTCatDom } (\eta1 \cdot 1 \eta2) = \text{NTCatDom } \eta1$ **by** (*auto simp add: NatTransComp'-def NTCatDom-def*)

lemma [*simp*]: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NTCatCod } (\eta1 \cdot 1 \eta2) = \text{NTCatCod } \eta1$ **by** (*auto simp add: NatTransComp'-def NTCatCod-def*)

lemma [*simp*]: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NTCatDom } (\eta1 \cdot \eta2) = \text{NTCatDom } \eta1$ **by** (*auto simp add: NatTransComp-defs*)

lemma [*simp*]: $\eta1 \approx \triangleright \cdot \eta2 \implies \text{NTCatCod } (\eta1 \cdot \eta2) = \text{NTCatCod } \eta1$ **by** (*auto simp add: NatTransComp-defs*)

lemma [*simp*]: $\text{NatTrans } \eta \implies \text{Category}(\text{NTCatDom } \eta)$ **by** (*simp add: NatTransP.NatTransFtor NTCatDom-def*)

lemma [*simp*]: $\text{NatTrans } \eta \implies \text{Category}(\text{NTCatCod } \eta)$ **by** (*simp add: NatTransP.NatTransFtor2 NTCatCod-def*)

lemma *DDDC*: **assumes** *NatTrans f* **shows** $\text{CatDom } (\text{NTDom } f) = \text{CatDom } (\text{NTCod } f)$

proof–

have $\text{CatDom } (\text{NTDom } f) = \text{NTCatDom } f$ **by** (*simp add: NTCatDom-def*)

thus *?thesis* **using** *assms* **by** (*simp add: NatTransP.NatTransFtorDom*)

qed
lemma *CCCD*: **assumes** $\text{NatTrans } f$ **shows** $\text{CatCod } (\text{NTCod } f) = \text{CatCod } (\text{NTDom } f)$
proof–
have $\text{CatCod } (\text{NTCod } f) = \text{NTCatCod } f$ **by** (*simp add: NTCatCod-def*)
thus *?thesis* **using** *assms* **by** (*simp add: NatTransP.NatTransFtorCod*)
qed

lemma *IdNatTransCompDefDom*: $\text{NatTrans } f \implies (\text{IdNatTrans } (\text{NTDom } f)) \approx_{>\cdot} f$
apply(*rule NTCompDefinedI*)
apply(*simp-all add: IdNatTransNatTrans NatTransP.NatTransFtor*)
apply(*simp-all add: IdNatTrans-defs CCCD*)
done

lemma *IdNatTransCompDefCod*: $\text{NatTrans } f \implies f \approx_{>\cdot} (\text{IdNatTrans } (\text{NTCod } f))$
apply(*rule NTCompDefinedI*)
apply(*simp-all add: IdNatTransNatTrans NatTransP.NatTransFtor2*)
apply(*simp-all add: IdNatTrans-defs DDDC*)
done

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

lemma *NatTransCompDefDom*:
assumes $\text{NatTrans } \eta$ **and** f **maps** $\text{NTCatDom } \eta$ X **to** Y
shows $(\text{NTDom } \eta \ \#\# f) \approx_{>\text{NTCatCod } \eta} (\eta \ \$\$ Y)$
proof(*rule CompDefinedI*)
have $b: X \in \text{obj}_{\text{NTCatDom } \eta}$ **and** $c: Y \in \text{obj}_{\text{NTCatDom } \eta}$ **using** *assms* **by** (*auto simp add: Category.MapsToObj*)

have $d: (\eta \ \$\$ \ Y) \text{ maps}_{NTCatCod \ \eta} ((NTDom \ \eta) \ @\@ \ Y) \text{ to } ((NTCod \ \eta) \ @\@ \ Y)$
using $assms \ c$
by $(simp \ add: \ NatTransP.NatTransMapsTo)$
thus $\eta \ \$\$ \ Y \in \text{mor}_{NTCatCod \ \eta}$ **by** $auto$
have $f \text{ maps}_{CatDom \ (NTDom \ \eta)} \ X \text{ to } Y$ **using** $assms$ **by** $(simp \ add: \ NTCatDom-def)$
hence $e: NTDom \ \eta \ \#\# \ f \text{ maps}_{CatCod \ (NTDom \ \eta)} (NTDom \ \eta \ @\@ \ X) \text{ to } (NTDom \ \eta \ @\@ \ Y)$ **using** $assms$
by $(simp \ add: \ FunctorM.FunctorCompM \ NatTransP.NatTransFtor)$
thus $NTDom \ \eta \ \#\# \ f \in \text{mor}_{NTCatCod \ \eta}$ **using** $assms$ **by** $(auto \ simp \ add: \ NatTransP.NatTransFtorCod)$
have $dom_{NTCatCod \ \eta} (\eta \ \$\$ \ Y) = (NTDom \ \eta \ @\@ \ Y)$ **using** d **by** $auto$
also have $\dots = cod_{CatCod \ (NTDom \ \eta)} (NTDom \ \eta \ \#\# \ f)$ **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 $\eta1 \ \approx\> \cdot \ \eta2$ **and** $X \in \text{obj}_{NTCatDom \ \eta1}$
shows $(\eta1 \ \$\$ \ X) \ \approx\>_{NTCatCod \ \eta1} (\eta2 \ \$\$ \ X)$
proof $(rule \ CompDefinedI)$
have $1: (\eta1 \ \$\$ \ X) \text{ maps}_{NTCatCod \ \eta1} ((NTDom \ \eta1) \ @\@ \ X) \text{ to } ((NTCod \ \eta1) \ @\@ \ X)$ **using** $assms$
by $(simp \ add: \ NatTransP.NatTransMapsTo)$
have $NTCatCod \ \eta1 = NTCatCod \ \eta2$ **using** $assms$ **by** $auto$
hence $2: (\eta2 \ \$\$ \ X) \text{ maps}_{NTCatCod \ \eta1} ((NTDom \ \eta2) \ @\@ \ X) \text{ to } ((NTCod \ \eta2) \ @\@ \ X)$ **using** $assms$
by $(simp \ add: \ NatTransP.NatTransMapsTo \ NTCatDom)$
show $\eta1 \ \$\$ \ X \in \text{mor}_{NTCatCod \ \eta1}$
and $\eta2 \ \$\$ \ X \in \text{mor}_{NTCatCod \ \eta1}$ **using** $1 \ 2$ **by** $auto$
have $cod_{NTCatCod \ \eta1} (\eta1 \ \$\$ \ X) = ((NTCod \ \eta1) \ @\@ \ X)$ **using** 1 **by** $auto$
also have $\dots = ((NTDom \ \eta2) \ @\@ \ X)$ **using** $assms$ **by** $auto$
finally show $cod_{NTCatCod \ \eta1} (\eta1 \ \$\$ \ X) = dom_{NTCatCod \ \eta1} (\eta2 \ \$\$ \ X)$ **using** 2 **by** $auto$
qed

lemma $NatTransCompNatTrans':$

assumes $\eta1 \ \approx\> \cdot \ \eta2$
shows $NatTransP \ (\eta1 \ \cdot 1 \ \eta2)$
proof $(auto \ simp \ add: \ NatTransP-def)$
show $Functor \ (NTDom \ (\eta1 \ \cdot 1 \ \eta2))$ **and** $Functor \ (NTCod \ (\eta1 \ \cdot 1 \ \eta2))$ **using** $assms$
by $(auto \ simp \ add: \ NatTransComp'-def \ NatTransP.NatTransFtor \ NatTransP.NatTransFtor2)$
show $NTCatDom \ (\eta1 \ \cdot 1 \ \eta2) = CatDom \ (NTCod \ (\eta1 \ \cdot 1 \ \eta2))$ **and**
 $NTCatCod \ (\eta1 \ \cdot 1 \ \eta2) = CatCod \ (NTDom \ (\eta1 \ \cdot 1 \ \eta2))$
proof $(auto \ simp \ add: \ NatTransComp'-def \ NTCatCod-def \ NTCatDom-def)$
have $CatDom \ (NTDom \ \eta1) = NTCatDom \ \eta1$ **by** $(simp \ add: \ NTCatDom-def)$
thus $CatDom \ (NTDom \ \eta1) = CatDom \ (NTCod \ \eta2)$ **using** $assms$ **by** $(auto$

simp add: NatTransP.NatTransFtorDom)
have $CatCod (NTCod \eta 2) = NTCatCod \eta 2$ **by** (*simp add: NTCatCod-def*)
thus $CatCod (NTCod \eta 2) = CatCod (NTDom \eta 1)$ **using** *assms* **by** (*auto simp add: NatTransP.NatTransFtorCod*)
qed
{
fix X **assume** $aa: X \in obj_{NTCatDom} (\eta 1 \cdot 1 \eta 2)$
show $(\eta 1 \cdot 1 \eta 2) \text{ \#\# } X \text{ maps}_{NTCatCod} (\eta 1 \cdot 1 \eta 2) NTDom (\eta 1 \cdot 1 \eta 2) @@ X$ **to**
 $NTCod (\eta 1 \cdot 1 \eta 2) @@ X$
proof-
have $X \in obj_{NTCatDom} \eta 1$ **and** $NatTrans \eta 1$ **using** *assms* **by** *simp+*
hence $(\eta 1 \text{ \#\# } X) \text{ maps}_{NTCatCod} \eta 1 ((NTDom \eta 1) @@ X)$ **to** $((NTCod \eta 1) @@ X)$
by (*simp add: NatTransP.NatTransMapsTo*)
moreover **have** $(\eta 2 \text{ \#\# } X) \text{ maps}_{NTCatCod} \eta 1 ((NTCod \eta 1) @@ X)$ **to** $((NTCod \eta 2) @@ X)$
proof-
have $X \in obj_{NTCatDom} \eta 2$ **and** $NatTrans \eta 2$ **using** *assms* **by** *auto*
hence $(\eta 2 \text{ \#\# } X) \text{ maps}_{NTCatCod} \eta 2 ((NTDom \eta 2) @@ X)$ **to** $((NTCod \eta 2) @@ X)$
by (*simp add: NatTransP.NatTransMapsTo*)
thus *?thesis* **using** *assms* **by** *auto*
qed
ultimately **have** $(\eta 1 \text{ \#\# } X) \text{ ;}_{NTCatCod} \eta 1 (\eta 2 \text{ \#\# } X) \text{ maps}_{NTCatCod} \eta 1$
 $((NTDom \eta 1) @@ X)$ **to** $((NTCod \eta 2) @@ X)$
using *assms* **by** (*simp add: Category.Ccompt*)
thus *?thesis* **using** *assms* **by** (*auto simp add: NatTransComp'-def NTCatCod-def*)
qed
}
{
fix $f X Y$ **assume** $a: f \text{ maps}_{(NTCatDom (\eta 1 \cdot 1 \eta 2))} X \text{ to } Y$
show $(NTDom (\eta 1 \cdot 1 \eta 2) \text{ \#\# } f) \text{ ;}_{NTCatCod} (\eta 1 \cdot 1 \eta 2) (\eta 1 \cdot 1 \eta 2 \text{ \#\# } Y) =$
 $((\eta 1 \cdot 1 \eta 2) \text{ \#\# } X) \text{ ;}_{NTCatCod} (\eta 1 \cdot 1 \eta 2) (NTCod (\eta 1 \cdot 1 \eta 2) \text{ \#\# } f)$
proof-
have $b: X \in obj_{NTCatDom} \eta 1$ **and** $c: Y \in obj_{NTCatDom} \eta 1$ **using** *assms* **by** *auto*
by (*auto simp add: Category.MapsToObj*)
have $((NTDom \eta 1) \text{ \#\# } f) \text{ ;}_{NTCatCod} \eta 1 ((\eta 1 \text{ \#\# } Y) \text{ ;}_{NTCatCod} \eta 1 (\eta 2 \text{ \#\# } Y)) =$
 $((NTDom \eta 1) \text{ \#\# } f) \text{ ;}_{NTCatCod} \eta 1 (\eta 1 \text{ \#\# } Y) \text{ ;}_{NTCatCod} \eta 1 (\eta 2 \text{ \#\# } Y)$
proof-
have $((NTDom \eta 1) \text{ \#\# } f) \approx_{NTCatCod} \eta 1 (\eta 1 \text{ \#\# } Y)$ **using** *assms* **by**
(*auto simp add: NatTransCompDefDom*)
moreover **have** $(\eta 1 \text{ \#\# } Y) \approx_{NTCatCod} \eta 1 (\eta 2 \text{ \#\# } Y)$ **using** *assms* **by**
(*simp add: NatTransCompCompDef c*)
ultimately **show** *?thesis* **using** *assms* **by** (*simp add: Category.Cassoc*)

qed
also have ... = ((η_1 \$\$ X) ;; $NTCatCod$ η_1 (($NTDom$ η_2) ## f)) ;; $NTCatCod$ η_1 (η_2 \$\$ Y)
using *assms a* **by** (*auto simp add: NatTransP.NatTrans*)
also have ... = (η_1 \$\$ X) ;; $NTCatCod$ η_1 ((($NTDom$ η_2) ## f) ;; $NTCatCod$ η_1 (η_2 \$\$ Y))
proof-
have (η_1 \$\$ X) $\approx_{>NTCatCod \eta_1}$ (($NTCod$ η_1) ## f) **using** *assms a* **by** (*simp add: NatTransCompDefCod*)
moreover have (($NTDom$ η_2) ## f) $\approx_{>NTCatCod \eta_1}$ (η_2 \$\$ Y) **using** *assms a*
by (*simp add: NatTransCompDefDom NTCatDom NTCatCod*)
ultimately show *?thesis* **using** *assms* **by** (*auto simp add: Category.Cassoc*)
qed
also have ... = (η_1 \$\$ X) ;; $NTCatCod$ η_1 ((η_2 \$\$ X) ;; $NTCatCod$ η_1 (($NTCod$ η_2) ## f))
using *assms a* **by** (*simp add: NatTransP.NatTrans NTCatDom NTCatCod*)
also have ... = (η_1 \$\$ X) ;; $NTCatCod$ η_1 (η_2 \$\$ X) ;; $NTCatCod$ η_1 (($NTCod$ η_2) ## f)
proof-
have (η_1 \$\$ X) $\approx_{>NTCatCod \eta_1}$ (η_2 \$\$ X) **using** *assms* **by** (*simp add: NatTransCompCompDef b*)
moreover have (η_2 \$\$ X) $\approx_{>NTCatCod \eta_1}$ (($NTCod$ η_2) ## 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*: $\eta_1 \approx_{> \cdot} \eta_2 \implies NatTrans (\eta_1 \cdot \eta_2)$
by (*simp add: NatTransCompNatTrans' NatTransComp-def MakeNT*)

definition

$CatExp' :: ('o1, 'm1, 'a) \text{Category-scheme} \Rightarrow ('o2, 'm2, 'b) \text{Category-scheme} \Rightarrow$
 $((('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{Functor},$
 $('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{NatTrans}) \text{Category} \text{ where}$

$CatExp' A B \equiv (\langle$
 $\text{Category.Obj} = \{F . Ftor F : A \longrightarrow B\} ,$
 $\text{Category.Mor} = \{\eta . NatTrans \eta \wedge NTCatDom \eta = A \wedge NTCatCod \eta = B\}$

\rangle
 $\text{Category.Dom} = NTDom ,$
 $\text{Category.Cod} = NTCod ,$
 $\text{Category.Id} = IdNatTrans ,$
 $\text{Category.Comp} = \lambda f g. (f \cdot g)$

\rangle

definition $CatExp\ A\ B \equiv MakeCat(CatExp'\ A\ B)$

lemma $IdNatTransMapL$:

assumes $NT: NatTrans\ f$
shows $IdNatTrans\ (NTDom\ f) \cdot f = f$
proof(*rule NatTransExt*)
show $NatTrans\ f$ **using** *assms* .
show $NatTrans\ (IdNatTrans\ (NTDom\ f) \cdot f)$ **using** NT
by (*simp add: NatTransP.NatTransFtor IdNatTransNatTrans IdNatTransCompDefDom NatTransCompNatTrans*)
show $NTDom\ (IdNatTrans\ (NTDom\ f) \cdot f) = NTDom\ f$ **and**
 $NTCod\ (IdNatTrans\ (NTDom\ f) \cdot f) = NTCod\ f$ **by** (*simp add: IdNatTrans-defs NatTransComp-defs*)
{
fix x **assume** $aa: x \in Obj\ (NTCatDom\ (IdNatTrans\ (NTDom\ f) \cdot f))$
show $(IdNatTrans\ (NTDom\ f) \cdot f)\ \$\$ x = f\ \$\$ x$
proof-
have $XObj: x \in Obj(NTCatDom\ f)$ **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 $(IdNatTrans\ (NTDom\ f) \cdot f)\ \$\$ x = (IdNatTrans\ (NTDom\ f)\ \$\$ x)$
 $::_{NTCatCod\ f}\ (f\ \$\$ x)$
proof(*rule NatTransComp-Comp1*)
show $x \in obj_{NTCatDom}\ (IdNatTrans\ (NTDom\ f))$ **using** $XObj$ **by** (*simp add: IdNatTrans-defs*)
show $IdNatTrans\ (NTDom\ f) \approx_{>\cdot} f$ **using** NT **by** (*simp add: IdNatTransCompDefDom*)
qed
also have $\dots = id_{NTCatCod\ f}\ (dom_{NTCatCod\ f}\ (f\ \$\$ x)) ::_{NTCatCod\ f}\ (f\ \$\$ x)$
using $XObj\ NT\ fMap$ **by** (*auto simp add: IdNatTrans-map NTCatDom-def CCCD NTCatCod-def*)
also have $\dots = f\ \$\$ x$
proof-
have $f\ \$\$ x \in mor_{NTCatCod\ f}$ **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 $f \cdot IdNatTrans\ (NTCod\ f) = f$
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 $$ x mapsNTCatCod f NTDom f @@ x to NTCod f @@ x using
    NT XObj
    by (simp add: NatTransP.NatTransMapsTo)
    have (f · IdNatTrans (NTCod f)) $$ x = (f $$ x) ;;NTCatCod f (IdNatTrans (NTCod f)) $$ x
    using XObj NT by (auto simp add: NatTransComp-Comp2 IdNatTransCompDefCod)
    also have ... = (f $$ x) ;;NTCatCod f (idNTCatCod f (codNTCatCod f (f $$ x)))
    proof–
    have x ∈ objCatDom (NTCod f) using XObj NT by (simp add: IdNatTrans-defs DDDC)
    moreover have (codNTCatCod f (f $$ x)) = (NTCod f) @@ x using fMap
    by auto
    ultimately have (IdNatTrans (NTCod f)) $$ x = (idNTCatCod f (codNTCatCod f (f $$ x)))
    by (simp add: IdNatTrans-map NTCatCod-def)
    thus ?thesis by simp
    qed
    also have ... = f $$ x
    proof–
    have f $$ x ∈ morNTCatCod f using fMap by (auto)
    thus ?thesis using NT by (simp add: Category.Cidr)
    qed
    finally show ?thesis .
  }
qed

```

```

lemma NatTransCompDefined:
  assumes f ≈>· g and g ≈>· h
  shows (f · g) ≈>· h and f ≈>· (g · h)
proof–
  show (f · g) ≈>· h
  proof(rule NTCompDefinedI)
  show NatTrans (f · 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: Nat-
TransComp-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: Nat-
TransComp-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  $f \cdot g \approx_{>} \cdot h$  using assms by (simp add: NatTransCompDefined)
qed
also have  $\dots = ((f \ \$\$ \ x) \ ; ; NTCatCod\ h\ (g \ \$\$ \ x)) \ ; ; NTCatCod\ h\ (h \ \$\$ \ x)$  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  $x \in \text{obj}_{NTCatDom} f$  using obj1 assms by (simp add: NTCatDom)
    show  $f \approx_{>} \cdot g \cdot h$  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  $f \in \text{mor}_{CatExp' A B}$ 
    thus  $\text{dom}_{CatExp' A B} f \in \text{obj}_{CatExp' A B}$  and
       $\text{cod}_{CatExp' A B} f \in \text{obj}_{CatExp' A B}$ 
    by(auto simp add: CatExp'-def NatTransP.NatTransFtor
      NatTransP.NatTransFtor2 NatTransP.NatTransFtorDom NatTransP.NatTransFtorCod
      DDDC CCCD functor-abbrev-def)
  }
  {
    fix F assume  $a: F \in \text{obj}_{CatExp' A B}$ 
    show  $\text{id}_{CatExp' A B} F \text{ maps } CatExp' A B \text{ } F \text{ to } F$ 
    proof(rule MapsToI)
      have  $\text{Ftor } F : A \longrightarrow B$  using a by (simp add: CatExp'-def)
      thus  $\text{id}_{CatExp' A B} F \in \text{mor}_{CatExp' A B}$ 
      apply(simp add: CatExp'-def NTCatDom-def NTCatCod-def IdNatTransNat-
Trans functor-abbrev-def)
      apply(simp add: IdNatTrans-defs)
    qed
  }

```

```

done
show domCatExp' A B (idCatExp' A B F) = F by (simp add: CatExp'-def
IdNatTrans-defs)
show codCatExp' A B (idCatExp' A B F) = F by (simp add: CatExp'-def
IdNatTrans-defs)
qed
}
{
fix f assume a: f ∈ morCatExp' A B
show (idCatExp' A B (domCatExp' A B f)) ∩CatExp' A B f = f and
f ∩CatExp' A B (idCatExp' A B (codCatExp' 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) · f = f using NT by (simp add: IdNatTransMapL)
show f · IdNatTrans (NTCod f) = f using NT by (simp add: IdNatTransMapR)
qed
}
}
{
fix f g h assume aa: f ≈CatExp' A B g and bb: g ≈CatExp' A B h
{
fix f g assume f ≈CatExp' A B g hence f ≈CatExp' A B · g
apply (simp only: NTCompDefined-def)
by (auto simp add: CatExp'-def)
}
hence f ≈CatExp' A B · g and g ≈CatExp' A B · 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 mapsCatExp' A B X to Y and b: g mapsCatExp' A B
Y to Z
show f ∩CatExp' A B g mapsCatExp' 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 ≈CatExp' A B · g by (auto intro: NTCompDefinedI)
thus NatTrans (f · g) by (simp add: NatTransCompNatTrans)
show NTCatDom (f · g) = A using Comp cd1 by (simp add: NTCatDom)
show NTCatCod (f · g) = B using Comp cc2 by (simp add: NTCatCod)
show NTDom (f · g) = X using d1 by (simp add: NatTransComp-defs)
show NTCod (f · g) = Z using c2 by (simp add: NatTransComp-defs)
qed
}
}
qed

```

lemma *CatExpCat*: $\llbracket \text{Category } A ; \text{Category } B \rrbracket \implies \text{Category } (\text{CatExp } A \ B)$
by (*simp add: CatExpCatAx CatExp-def MakeCat*)

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

lemma *CatExpDom*: $f \in \text{Mor } (\text{CatExp } A \ B) \implies \text{dom}_{\text{CatExp } A \ B} f = \text{NTDom } f$
by (*simp add: CatExp-defs*)

lemma *CatExpCod*: $f \in \text{Mor } (\text{CatExp } A \ B) \implies \text{cod}_{\text{CatExp } A \ B} f = \text{NTCod } f$
by (*simp add: CatExp-defs*)

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

lemma *CatExpNatTransCompDef*: **assumes** $f \approx_{>} \text{CatExp } A \ B \ g$ **shows** $f \approx_{>} \cdot g$
proof–

have $1: f \approx_{>} \text{CatExp}' \ A \ B \ g$ **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* $g = \text{NTCatDom } f$ **using** 1 **by** (*auto simp add: CatExp'-def*)

show *NTCatCod* $g = \text{NTCatCod } f$ **using** 1 **by** (*auto simp add: CatExp'-def*)

show *NTCod* $f = \text{NTDom } g$ **using** 1 **by** (*auto simp add: CatExp'-def*)

qed

qed

lemma *CatExpDist*:

assumes $X \in \text{Obj } A$ **and** $f \approx_{>} \text{CatExp } A \ B \ g$

shows $(f \ ;_{\text{CatExp } A \ B} g) \ \$$ \ X = (f \ \$\$ X) \ ;_B (g \ \$\$ X)$

proof–

have $f \in \text{Mor } (\text{CatExp}' \ A \ B)$ **using** *assms* **by** (*auto simp add: CatExp-def MakeCatMor*)

hence $1: \text{NTCatDom } f = A$ **and** $2: \text{NTCatCod } f = B$ **by** (*simp add: CatExp'-def*)**+**

hence $4: X \in \text{Obj } (\text{NTCatDom } f)$ **using** *assms* **by** *simp*

have $3: f \approx_{>} \cdot g$ **using** *assms*(2) **by** (*simp add: CatExpNatTransCompDef*)

have $(f \ ;_{\text{CatExp } A \ B} g) \ \$$ \ X = (f \ ;_{\text{CatExp}' \ A \ B} g) \ \$$ \ X$ **using** *assms*(2) **by** (*simp add: CatExp-def MakeCatComp2*)

also have $\dots = (f \cdot g) \ \$$ \ X$ **by** (*simp add: CatExp'-def*)

also have $\dots = (f \ \$\$ X) \ ;_B (g \ \$\$ X)$ **using** $4 \ 2 \ 3$ **by** (*simp add: NatTransComp-Comp2[of X f g]*)

finally show *?thesis* .

qed

lemma *CatExpMorNT*: $f \in \text{Mor } (\text{CatExp } A \ B) \implies \text{NatTrans } f$

by (simp add: CatExp-defs)

end

6 The Category of Sets

theory SetCat
imports Functors Universe
begin

notation Elem (infixl $|\in|$ 70)

notation HOLZF.subset (infixl $|\subseteq|$ 71)

notation CartProd (infixl $|\times|$ 75)

definition

ZFfun :: ZF \Rightarrow ZF \Rightarrow (ZF \Rightarrow ZF) \Rightarrow ZF **where**
ZFfun d r f \equiv Opair (Opair d r) (Lambda d f)

definition

ZFfunDom :: ZF \Rightarrow ZF ($|dom|$ - [72] 72) **where**
ZFfunDom f \equiv Fst (Fst f)

definition

ZFfunCod :: ZF \Rightarrow ZF ($|cod|$ - [72] 72) **where**
ZFfunCod f \equiv Snd (Fst f)

definition

ZFfunApp :: ZF \Rightarrow ZF \Rightarrow ZF (infixl $|\@|$ 73) **where**
ZFfunApp f x \equiv app (Snd f) x

definition

ZFfunComp :: ZF \Rightarrow ZF \Rightarrow ZF (infixl $|o|$ 72) **where**
ZFfunComp f g \equiv ZFfun ($|dom|$ f) ($|cod|$ g) ($\lambda x. g |\@| (f |\@| x)$)

definition

isZFfun :: ZF \Rightarrow bool **where**
isZFfun drf \equiv let f = Snd drf in
isOpair drf \wedge isOpair (Fst drf) \wedge isFun f \wedge (f $|\subseteq|$ (Domain f) $|\times|$
(Range f))
 \wedge (Domain f = $|dom|$ drf) \wedge (Range f $|\subseteq|$ $|cod|$ drf)

lemma isZFfunE[elim]: \llbracket isZFfun f ;

\llbracket isOpair f ; isOpair (Fst f) ; isFun (Snd f) ;
 \llbracket (Snd f) $|\subseteq|$ (Domain (Snd f)) $|\times|$ (Range (Snd f)) \rrbracket ;
(Domain (Snd f) = $|dom|$ f) \wedge (Range (Snd f) $|\subseteq|$ $|cod|$ f) $\rrbracket \Longrightarrow R \Longrightarrow R$
by (auto simp add: isZFfun-def Let-def)

definition

SET' :: (ZF, ZF) Category **where**

```

SET' ≡ (|
  Category.Obj = {x . True} ,
  Category.Mor = {f . isZFfun f} ,
  Category.Dom = ZFfunDom ,
  Category.Cod = ZFfunCod ,
  Category.Id = λx. ZFfun x x (λx . 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 ∀ x . x |∈| A ⟶ (f x) |∈| 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 |∈| Range (Lambda A f) ⟹ x |∈| 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 |∈| (Lambda A f) ⟹ x |∈| A |×| Range (Lambda A f)
      by(auto simp add: CartProd Lambda-def Repl Range)
  }
  thus (Lambda A f) |⊆| (A |×| Range (Lambda A f))
    by (auto simp add: HOLZF.subset-def)
qed

```

lemma $ZFCartProd:$

```

  assumes x |∈| A |×| B
  shows Fst x |∈| A ∧ Snd x |∈| B ∧ 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 $x \in | \text{Domain } f$

shows $\text{Opair } x (\text{app } f x) \in | f$

proof-

have $\exists! y . \text{Opair } x y \in | f$ using *assms* by (*auto simp add: unique-fun-value*)

thus $\text{Opair } x (\text{app } f x) \in | f$ by (*auto simp add: app-def intro: theI'*)

qed

lemma *ZFFunToLambda*:

assumes 1: *isFun* *f*

and 2: $f \subseteq | (\text{Domain } f) \times | (\text{Range } f)$

shows $f = \text{Lambda } (\text{Domain } f) (\lambda x. \text{app } f x)$

proof(*subst Ext, rule allI, rule iffI*)

{

fix *x* assume $a: x \in | f$ show $x \in | \text{Lambda } (\text{Domain } f) (\lambda x. \text{app } f x)$

proof(*simp add: Lambda-def Repl, rule exI[of - (Fst x)], rule conjI*)

have $b: \text{isOpair } x \wedge \text{Fst } x \in | \text{Domain } f$ using 2 *a* by (*auto simp add: subset-def ZFCartProd*)

thus $\text{Fst } x \in | \text{Domain } f$..

hence $\text{Opair } (\text{Fst } x) (\text{app } f (\text{Fst } x)) \in | f$ using 1 by (*simp add: ZFfunDomainOpair*)

moreover have $\text{Opair } (\text{Fst } x) (\text{Snd } x) \in | f$ using *a* 2 by (*auto simp add: FstSnd subset-def b*)

ultimately have $\text{Snd } x = (\text{app } f (\text{Fst } x))$ using 1 by (*auto simp add: isFun-def*)

hence $\text{Opair } (\text{Fst } x) (\text{app } f (\text{Fst } x)) = \text{Opair } (\text{Fst } x) (\text{Snd } x)$ by *simp*

also have $\dots = x$ using *b* by (*simp add: FstSnd*)

finally show $x = \text{Opair } (\text{Fst } x) (\text{app } f (\text{Fst } x))$..

qed

}

moreover

{

fix *x* assume $a: x \in | \text{Lambda } (\text{Domain } f) (\lambda x. \text{app } f x)$ show $x \in | f$

proof-

from *Lambda-def* obtain *a* where $a \in | \text{Domain } f \wedge x = \text{Opair } a (\text{app } f a)$

using *a* by (*auto simp add: Repl*)

thus *thesis* using *a* 1 by (*auto simp add: ZFfunDomainOpair*)

qed

}

qed

lemma *ZFfunApp*:

assumes $x \in | A$

shows $(\text{ZFfun } A B f) \text{ |@| } x = f x$

proof-

have $(\text{ZFfun } A B f) \text{ |@| } x = \text{app } (\text{Lambda } A f) x$ by (*simp add: ZFfun-def*)

ZFfunApp-def Snd)
also have $\dots = f x$ **using** *assms* **by** (*simp add: Lambda-app*)
finally show *?thesis* .
qed

lemma *ZFfun*:
assumes *isZFfun f*
shows $f = \text{ZFfun } (|dom| f) (|cod| f) (\lambda x. f |@| x)$
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 $f = \text{Opair } (\text{Opair } (\text{Fst } (\text{Fst } f)) (\text{Snd } (\text{Fst } f))) (\text{Snd } f)$ **by** (*simp add: FstSnd*)
hence $f = \text{Opair } (\text{Opair } (|dom| f) (|cod| f)) (\text{Snd } f)$ **using** *assms* **by** (*simp add: ZFfunDom-def ZFfunCod-def*)
moreover have $\text{Snd } f = \text{Lambda } (|dom| f) (\lambda x. f |@| x)$
proof-
have $|dom| f = \text{Domain } (\text{Snd } f)$ **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 $(\lambda x. f |@| x) = (\lambda x. \text{app } (\text{Snd } f) x)$ **by**(*simp add: ZFfunApp-def*)
moreover have $(\text{Snd } f) \subseteq (\text{Domain } (\text{Snd } f)) \times (\text{Range } (\text{Snd } f))$ **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 $f = \text{Opair } (\text{Opair } (|dom| f) (|cod| f)) (\text{Lambda } (|dom| f) (\lambda x. f |@| x))$ **by** *simp*
qed

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

lemma *ZFfunExt*:
assumes $|dom| f = |dom| g$ **and** $|cod| f = |cod| g$ **and** *funf: isZFfun f* **and** *fung: isZFfun g*
and $\bigwedge x. x \in (|dom| f) \implies f |@| x = g |@| x$
shows $f = g$
proof-
have $1: f = \text{ZFfun } (|dom| f) (|cod| f) (\lambda x. f |@| x)$ **using** *funf* **by** (*rule ZFfun*)
have $g = \text{ZFfun } (|dom| g) (|cod| g) (\lambda x. g |@| x)$ **using** *fung* **by** (*rule ZFfun*)
hence $2: g = \text{ZFfun } (|dom| f) (|cod| f) (\lambda x. g |@| x)$ **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 $x |∈| |dom| f$
shows $f |@| x |∈| |cod| f$
proof(*simp add: ZFfunApp-def*)
have $app (Snd f) x |∈| Range (Snd f)$ **using** *assms* **by** (*auto simp add: fun-value-in-range*)
thus $app (Snd f) x |∈| |cod| f$ **using** *assms* **by** (*auto simp add: HOLZF.subset-def*)
qed

lemma *ZFfunComp*:
assumes $\forall x. x |∈| A \longrightarrow f x |∈| B$
shows $(ZFfun A B f) |o| (ZFfun B C g) = ZFfun A C (g \circ f)$
proof (*simp add: ZFfunComp-def ZFfunDom ZFfunCod*)
{
fix x **assume** $a: x |∈| A$
have $ZFfun B C g |@| (ZFfun A B f |@| x) = (g \circ f) x$
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: $\forall x. x |∈| |dom| f \longrightarrow (\lambda x. f |@| x) x |∈| |cod| f$ **using** a **by** (*simp add: ZFfunDomAppCod*)
hence 4: $\forall x. x |∈| |dom| f \longrightarrow (\lambda x. g |@| (f |@| x)) x |∈| |cod| g$
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 ZFfun-Dom*)
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 ZFfun-Dom*)
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 ∈ Obj (SET)*

by (*simp add: SET-def SET'-def MakeCat-def*)

lemma *SETcod: isZFfun (ZFfun A B f) \implies cod_{SET} ZFfun A B f = B*

by(*simp add: SET-def MakeCat-def SET'-def ZFfunCod*)

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

lemma SETdom: $isZFun (ZFun A B f) \implies dom_{SET} ZFun A B f = A$
by(simp add: SET-def MakeCat-def SET'-def ZFunDom)

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 $isZFun(Id SET X)$ **by** (simp add: SETCategory Category.CatIdInMor SETmor)
moreover have $(Id SET X) = ZFun X X (\lambda x. x)$ **using** *assms* **by** (simp add: SET-def SET'-def MakeCat-def)
ultimately show *?thesis* **using** *assms* **by** (simp add: ZFunApp)
qed

lemma SETCompE[elim]: $\llbracket f \approx_{SET} g ; \llbracket isZFun f ; isZFun g ; |cod| f = |dom| g \rrbracket \implies R \rrbracket \implies R$
by (auto simp add: SET-def SET'-def MakeCat-def)

lemma SETmapsTo: $f maps_{SET} X \text{ to } Y \implies isZFun f \wedge |dom| f = X \wedge |cod| f = Y$
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 $\dots = f ;;_{SET'} g$ **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 $\dots = ZFun (|dom| f) (|cod| g) (\lambda x. g \ @ \ (f \ @ \ x))$ **using** *assms*
by (auto simp add: ZFunCompApp)
finally show *?thesis* **using** *assms* **by** (simp add: ZFunApp)
qed

lemma SETZFun:

assumes $f maps_{SET} X \text{ to } Y$ **shows** $f = ZFun X Y (\lambda x. f \ @ \ x)$

proof-

have $isZFun f$ **using** *assms* **by** (auto simp add: SETmor)
hence $f = ZFun (|dom| f) (|cod| f) (\lambda x. f \ @ \ x)$ **by** (simp add: ZFun)
moreover have $|dom| f = X$ **and** $|cod| f = Y$ **using** *assms* **by** (auto simp add: SETmor)

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: isZFun 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: ZFunDomAppCod*)
thus *?thesis* **using** *2* **by** *simp*
qed

record (*'o, 'm*) *LSCategory* = (*'o, 'm*) *Category* +
mor2ZF :: *'m* \Rightarrow *ZF* (*m2z1*- [70] 70)

definition
ZF2mor (*z2m1*- [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 (*Hom1* - - [65, 65] 65) **where**
HomSet *C X Y* \equiv *implode* (*HOMCollection* *C X Y*)

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

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

definition
HomFtorMap :: (*'o, 'm, 'a*) *LSCategory-scheme* \Rightarrow *'o* \Rightarrow *'m* \Rightarrow *ZF* (*Hom1*[-, -]
[65, 65] 65) **where**
HomFtorMap *C X g* \equiv *ZFun* (*Hom*_{*C*} *X* (*dom*_{*C*} *g*)) (*Hom*_{*C*} *X* (*cod*_{*C*} *g*)) ($\lambda f .$
*m2z*_{*C*} ((*z2m*_{*C*} *f*) ;; *C* *g*))

definition

$HomFtor' :: ('o, 'm, 'a) LSCategory-scheme \Rightarrow 'o \Rightarrow$
 $('o, ZF, 'm, ZF, (\{mor2ZF :: 'm \Rightarrow ZF, \dots :: 'a\}, unit) Functor (HomP1[-, -] [65]$
 65) **where**
 $HomFtor' C X \equiv ()$
 $CatDom = C,$
 $CatCod = SET,$
 $MapM = \lambda g . Hom_C[X, g]$
 $\})$

definition $HomFtor (Hom1[-, -] [65] 65)$ **where** $HomFtor C X \equiv MakeFtor (HomFtor' C X)$

lemma $[simp]: LSCategory C \Longrightarrow Category C$
by $(simp\ add: LSCategory-def)$

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

lemma $(in\ LSCategory)\ m2zz2mInv:$
assumes $f \in mor$
shows $z2m\ (m2z\ f) = f$
proof-
have $1: f \in mor \wedge m2z\ f = m2z\ f$ **using** $assms$ **by** $simp$
moreover $\exists! m . m \in mor \wedge m2z\ m = (m2z\ f)$
proof $(rule\ ex-ex1I)$
show $\exists m . m \in mor \wedge m2z\ m = (m2z\ f)$
by $(rule\ exI[of\ -\ f],\ insert\ 1,\ simp)$
 $\{$
fix $m\ y$ **assume** $m \in mor \wedge m2z\ m = (m2z\ f)$ **and** $y \in mor \wedge m2z\ y = (m2z\ f)$
 $\}$
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: $\exists m . m \text{ maps } X \text{ to } Y \wedge m2z m = f$
proof-
have *HOMCollection* $C X Y \in \text{range explode}$ **using** *assms* **by** (*simp add: HOMSetIsSet*)
moreover **have** $f \in$ *implode* (*HOMCollection* $C X Y$) **using** *assms*(β) **by** (*simp add: HomSet-def*)
ultimately **have** $f \in \text{HOMCollection } C X Y$ **by** (*simp add: HOLZF.Elem-implode*)
thus *?thesis* **by** (*auto simp add: HOMCollection-def*)
qed
have 2: $\exists! m . m \in \text{mor} \wedge m2z m = f$
proof(*rule ex-ex1I*)
show $\exists m . m \in \text{mor} \wedge m2z m = f$
proof-
from 1 **obtain** m **where** $m \in \text{mor} \wedge m2z m = f$ **by** *auto*
thus *?thesis* **by** *auto*
qed
{
fix $m y$ **assume** $m \in \text{mor} \wedge m2z m = f$ **and** $y \in \text{mor} \wedge m2z y = f$
thus $m = y$ **by**(*simp add: mor2ZFInj*)
}
qed
thus *?thesis*
proof-
from 1 **obtain** a **where** 3: $a \text{ maps } X \text{ to } Y \wedge m2z a = f$ **by** *auto*
have 4: $a \in \text{mor}$ **using** 3 **by** *auto*
have $z2m f = a$
apply (*auto simp add: 3 ZF2mor-def[of - f]*)
apply (*rule the1-equality[of $\lambda m . m \in \text{mor} \wedge m2z m = f a$]*)
apply (*auto simp add: 2 3 4*)
done
thus *?thesis* **by** (*simp add: 3*)
qed
qed

lemma *HomFtorMapLemma1:*

assumes $a: \text{LSCategory } C$ **and** $b: X \in \text{obj}_C$ **and** $c: f \in \text{mor}_C$ **and** $d: x \in$
 $(\text{Hom}_C X (\text{dom}_C f))$
shows $(m2z_C ((z2m_C x) ;;_C f)) \in (\text{Hom}_C X (\text{cod}_C f))$
proof-
have 1: $\text{dom}_C f \in \text{obj}_C$ **and** 2: $\text{cod}_C f \in \text{obj}_C$ **using** $a c$ **by** (*simp add: Category.Simps*)
have $z2m_C x \text{ maps}_C X \text{ to } (\text{dom}_C f)$ **using** $a b d 1$ **by** (*auto simp add: LSCategory.z2mm2z*)
hence $(z2m_C x) ;;_C f \text{ maps}_C X \text{ to } (\text{cod}_C f)$ **using** $a c$ **by** (*auto intro: Category.Ccompt*)
hence $(m2z_C ((z2m_C x) ;;_C f)) \in (\text{Hom}_C X (\text{cod}_C f))$ **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** $X \in \text{obj}_C$ **and** $f \in \text{mor}_C$

shows $\text{Hom}_C[X, f] \in \text{mor}_{\text{SET}'}$

proof(*simp add: HomFtorMap-def*)

{

fix x **assume** $x \in |(\text{Hom}_C X \text{ dom}_C f)$

hence $m2z_C ((z2m_C x) ;;_C f) \in |(\text{Hom}_C X \text{ cod}_C f)$ **using** *assms* **by** (*blast intro: HomFtorMapLemma1*)

}

hence $\forall x . x \in |(\text{Hom}_C X \text{ dom}_C f) \longrightarrow (m2z_C ((z2m_C x) ;;_C f)) \in |(\text{Hom}_C X \text{ cod}_C f)$ **by** (*simp*)

hence $\text{isZFFun} (\text{ZFFun} (\text{Hom}_C X \text{ dom}_C f) (\text{Hom}_C X \text{ cod}_C f) (\lambda x . m2z_C ((z2m_C x) ;;_C f)))$

by (*simp add: SETfun*)

thus $\text{ZFFun} (\text{Hom}_C X \text{ dom}_C f) (\text{Hom}_C X \text{ cod}_C f) (\lambda x . m2z_C ((z2m_C x) ;;_C f)) \in \text{mor}_{\text{SET}'}$

by (*simp add: SET'-def*)

qed

lemma *HomFtorMor'*:

assumes *LSCategory* C **and** $X \in \text{obj}_C$ **and** $f \in \text{mor}_C$

shows $\text{Hom}_C[X, f]$ *maps* $_{\text{SET}'}$ $\text{Hom}_C X (\text{dom}_C f)$ *to* $\text{Hom}_C X (\text{cod}_C f)$

proof–

have $\text{Hom}_C[X, f] \in \text{mor}_{\text{SET}'}$ **using** *assms* **by** (*simp add: HomFtorInMor'*)

moreover **have** $\text{dom}_{\text{SET}'} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{dom}_C f)$

by(*simp add: HomFtorMap-def SET'-def ZFFunDom*)

moreover **have** $\text{cod}_{\text{SET}'} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{cod}_C f)$

by(*simp add: HomFtorMap-def SET'-def ZFFunCod*)

ultimately show *?thesis* **by** (*auto simp add: SET-def*)

qed

lemma *HomFtorMapsTo*:

$\llbracket \text{LSCategory } C ; X \in \text{obj}_C ; f \in \text{mor}_C \rrbracket \implies \text{Hom}_C[X, f]$ *maps* $_{\text{SET}}$ $\text{Hom}_C X (\text{dom}_C f)$ *to* $\text{Hom}_C X (\text{cod}_C f)$

by (*simp add: HomFtorMor' SET-def MakeCatMapsTo*)

lemma *HomFtorMor*:

assumes *LSCategory* C **and** $X \in \text{obj}_C$ **and** $f \in \text{mor}_C$

shows $\text{Hom}_C[X, f] \in \text{Mor } \text{SET}$ **and** $\text{dom}_{\text{SET}} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{dom}_C f)$

and $\text{cod}_{\text{SET}} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{cod}_C f)$

proof–

have $\text{Hom}_C[X, f]$ *maps* $_{\text{SET}}$ $\text{Hom}_C X (\text{dom}_C f)$ *to* $\text{Hom}_C X (\text{cod}_C f)$ **using** *assms* **by** (*simp add: HomFtorMapsTo*)

thus $\text{Hom}_C[X, f] \in \text{Mor } \text{SET}$ **and** $\text{dom}_{\text{SET}} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{dom}_C f)$

and $\text{cod}_{\text{SET}} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{cod}_C f)$

by *auto*

qed

lemma *HomFtorCompDef'*:

assumes *LSCategory C* **and** $X \in \text{obj}_C$ **and** $f \approx_{>C} g$

shows $(\text{Hom}_C[X, f]) \approx_{>SET'} (\text{Hom}_C[X, g])$

proof(*rule CompDefinedI*)

have $a: f \in \text{mor}_C$ **and** $b: g \in \text{mor}_C$ **using** *assms(3)* **by** *auto*

thus $\text{Hom}_C[X, f] \in \text{mor}_{SET'}$ **and** $\text{Hom}_C[X, g] \in \text{mor}_{SET'}$ **using** *assms* **by** (*simp add: HomFtorInMor'*) $+$

have $(\text{Hom}_C[X, f]) \text{ maps}_{SET'} \text{Hom}_C X \text{ dom}_C f \text{ to } \text{Hom}_C X \text{ cod}_C f$

and $(\text{Hom}_C[X, g]) \text{ maps}_{SET'} \text{Hom}_C X \text{ dom}_C g \text{ to } \text{Hom}_C X \text{ cod}_C g$ **using** *assms* a b **by** (*simp add: HomFtorMor'*) $+$

hence $\text{cod}_{SET'} (\text{Hom}_C[X, f]) = \text{Hom}_C X (\text{cod}_C f)$

and $\text{dom}_{SET'} (\text{Hom}_C[X, g]) = \text{Hom}_C X (\text{dom}_C g)$ **by** *auto*

moreover $(\text{cod}_C f) = (\text{dom}_C g)$ **using** *assms(3)* **by** *auto*

ultimately show $\text{cod}_{SET'} (\text{Hom}_C[X, f]) = \text{dom}_{SET'} (\text{Hom}_C[X, g])$ **by** *simp*

qed

lemma *HomFtorDist'*:

assumes $a: \text{LSCategory } C$ **and** $b: X \in \text{obj}_C$ **and** $c: f \approx_{>C} g$

shows $(\text{Hom}_C[X, f]) ;;_{SET'} (\text{Hom}_C[X, g]) = \text{Hom}_C[X, f ;;_C g]$

proof–

let $?A = (\text{Hom}_C X \text{ dom}_C f)$

let $?B = (\text{Hom}_C X \text{ dom}_C g)$

let $?C = (\text{Hom}_C X \text{ cod}_C g)$

let $?f = (\lambda h. \text{m2z}_C ((\text{z2m}_C h) ;;_C f))$

let $?g = (\lambda f. \text{m2z}_C ((\text{z2m}_C f) ;;_C g))$

have $1: \text{cod}_C f = \text{dom}_C g$ **using** c **by** *auto*

have $2: \text{dom}_C (f ;;_C g) = \text{dom}_C f$ **and** $3: \text{cod}_C (f ;;_C g) = \text{cod}_C g$ **using** *assms*

by (*auto simp add: Category.MapsToMorDomCod*)

have $(\text{Hom}_C[X, f]) ;;_{SET'} (\text{Hom}_C[X, g]) = (\text{ZFfun } ?A (\text{Hom}_C X \text{ cod}_C f) ?f) |o| (\text{ZFfun } ?B ?C ?g)$

by (*simp add: HomFtorMap-def SET'-def*)

also have $\dots = (\text{ZFfun } ?A ?B ?f) |o| (\text{ZFfun } ?B ?C ?g)$ **using** 1 **by** *simp*

also have $\dots = \text{ZFfun } ?A ?C (?g o ?f)$

proof(*rule ZFfunComp, rule allI, rule impI*)

{

fix h **assume** $aa: h \in | \in | ?A$ **show** $?f h \in | \in | ?B$

proof–

have $f \in \text{mor}_C$ **using** *assms* **by** *auto*

hence $?f h \in | \in | (\text{Hom}_C X \text{ cod}_C f)$ **using** *assms* aa **by** (*simp add:*

HomFtorMapLemma1)

thus *thesis* **using** 1 **by** *simp*

qed

}

qed

also have $\dots = \text{ZFfun } ?A ?C (\lambda h. \text{m2z}_C ((\text{z2m}_C h) ;;_C (f ;;_C g)))$

proof(*rule ZFfun-ext, rule allI, rule impI, simp add: comp-def*)

```

{
  fix h assume aa: h |∈| ?A
  show m2zC ((z2mC (m2zC ((z2mC h) ;;C f))) ;;C g) = m2zC ((z2mC h) ;;C
(f ;;C g))
  proof-
    have bb: (z2mC h) ≈>C f
    proof(rule CompDefinedI)
      show f ∈ morC using c by auto
      hence domC f ∈ objC using a by (simp add: Category.Cdom)
      hence (z2mC h) mapsC X to domC f using assms aa by (simp add:
LSCategory.z2mm2z)
      thus (z2mC h) ∈ morC and codC (z2mC h) = domC f by auto
    qed
    hence (z2mC h) ;;C f ∈ morC using a by (simp add: Category.MapsToMorDomCod)
    hence z2mC (m2zC ((z2mC h) ;;C f)) = (z2mC h) ;;C f using a by (simp
add: LSCategory.m2zz2mInv)
    hence m2zC ((z2mC (m2zC ((z2mC h) ;;C f))) ;;C g) = m2zC (((z2mC h)
;;C f) ;;C g) by simp
    also have ... = m2zC ((z2mC h) ;;C (f ;;C g)) using bb c a by (simp add:
Category.Cassoc)
    finally show ?thesis .
  qed
}
qed
also have ... = ZFfun (HomC X domC (f ;;C g)) (HomC X codC (f ;;C g)) (λh.
m2zC ((z2mC h) ;;C (f ;;C g)))
  using 2 3 by simp
also have ... = HomC[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 ∈ objC and f ≈>C g
  shows (HomC[X,f]) ;;SET (HomC[X,g]) = HomC[X,f ;;C g]
  proof-
    have (HomC[X,f]) ;;SET' (HomC[X,g]) = HomC[X,f ;;C g] using assms by
(simp add: HomFtorDist')
    moreover have (HomC[X,f]) ≈>SET' (HomC[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 ∈ objC and c: Y ∈ objC
  shows HomC[X,idC Y] = idSET' (HomC X Y)
  proof-
    have (idC Y) mapsC Y to Y using a c by (simp add: Category.Simps)
    hence 1: (domC (idC Y)) = Y and 2: (codC (idC Y)) = Y by auto
    have HomC[X,idC Y] = ZFfun (HomC X (domC (idC Y))) (HomC X (codC

```

```

(idC Y))) (λ f . m2zC ((z2mC f) ;;C (idC Y)))
  by (simp add: HomFtorMap-def)
  also have ... = ZFfun (HomC X Y) (HomC X Y) (λ f . m2zC ((z2mC f) ;;C
(idC Y))) using 1 2 by simp
  also have ... = ZFfun (HomC X Y) (HomC X Y) (λ f . f)
  proof(rule ZFfun-ext, rule allI, rule impI)
    {
      fix h assume aa: h |∈| (HomC X Y) show m2zC ((z2mC h) ;;C (idC Y))
= h
      proof-
        have (z2mC h) mapsC X to Y and bb: m2zC (z2mC h) = h
          using assms aa by (simp add: LSCategory.z2mm2z)+
        hence (z2mC h) ;;C (idC Y) = (z2mC h) using a by (auto simp add:
Category.Simps)
        hence m2zC ((z2mC h) ;;C (idC Y)) = m2zC (z2mC 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 ∈ objC and Y ∈ objC
shows HomC[X, idC Y] = idSET (HomC X Y)
proof-
  have HomC[X, idC Y] = idSET' (HomC X Y) using assms by (simp add:
HomFtorId')
  moreover have (HomC X Y) ∈ objSET' by (simp add: SET'-def)
  ultimately show ?thesis by (simp add: MakeCatId SET-def)
qed

```

lemma *HomFtorObj'*:

```

assumes a: LSCategory C
and b: PreFunctor (HomPC[X, -]) and c: X ∈ objC and d: Y ∈ objC
shows (HomPC[X, -]) @@ Y = HomC X Y
proof-
  let ?F = (HomFtor' C X)
  have ?F ## (idCatDom ?F Y) = HomC[X, idC Y] by (simp add: HomFtor'-def)
  also have ... = idCatCod ?F (HomC X Y) using assms by (simp add: HomFtorId
HomFtor'-def)
  finally have ?F ## (idCatDom ?F Y) = idCatCod ?F (HomC X Y) by simp
  moreover have HomC X Y ∈ objCatCod ?F using assms
  by (simp add: HomFtorId HomFtor'-def SET-def SET'-def MakeCatObj)
  moreover have Y ∈ objCatDom ?F using d by (simp add: HomFtor'-def)
  ultimately show ?thesis using b by (simp add: PreFunctor.FmToFo[of ?F Y
HomC X Y])
qed

```

```

lemma HomFtorFtor':
  assumes a: LSCategory C
  and b:  $X \in \text{obj}_C$ 
  shows FunctorM (HomPC[X, -])
proof(intro-locales)
  show PF: PreFunctor (HomPC[X, -])
  proof(auto simp add: HomFtor'-def PreFunctor-def SETCategory a HomFtorDist
b)
  {
    fix Z assume aa:  $Z \in \text{obj}_C$ 
    show  $\exists Y \in \text{obj}_{SET} . \text{Hom}_C[X, \text{id}_C Z] = \text{id}_{SET} Y$ 
    proof(rule-tac x=Hom_C X Z in Set.rev-be1)
      show  $\text{Hom}_C X Z \in \text{obj}_{SET}$  by (simp add: SET-def SET'-def MakeCatObj)
      show  $\text{Hom}_C[X, \text{id}_C Z] = \text{id}_{SET} (\text{Hom}_C X Z)$  using assms aa by(simp
add:HomFtorId)
    qed
  }
  qed
  {
    fix f Z Y assume aa: f mapsC Z to Y
    have (HomPC[X, -]) ## f mapsSET ((HomPC[X, -]) @@ Z) to ((HomPC[X, -])
    @@ Y)
    proof-
      have bb:  $Z \in \text{obj}_C$  and cc:  $Y \in \text{obj}_C$  using aa a by (simp add: Cate-
gory.MapsToObj)+
      have dd:  $\text{dom}_C f = Z$  and ee:  $\text{cod}_C f = Y$  and ff: f  $\in \text{mor}_C$  using aa by
auto
      have (HomPC[X, -]) ## f = HomC[X, f] by (simp add: HomFtor'-def)
      moreover have (HomPC[X, -]) @@ Z = HomC X Z
      and (HomPC[X, -]) @@ Y = HomC X Y using assms bb cc PF by (simp
add: HomFtorObj')+
      moreover have HomC[X, f] mapsSET (HomC X (domC f)) to (HomC X
      (codC f))
      using assms ff by (simp add: HomFtorMapsTo)
      ultimately show ?thesis using dd ee by simp
    qed
  }
  thus FunctorM-axioms (HomPC[X, -]) using PF by (auto simp add: Func-
torM-axioms-def HomFtor'-def)
qed

```

```

lemma HomFtorFtor:
  assumes a: LSCategory C
  and b:  $X \in \text{obj}_C$ 
  shows Functor (HomC[X, -])
proof-
  have FunctorM (HomPC[X, -]) using assms by (rule HomFtorFtor')
  thus ?thesis by (simp add: HomFtor-def MakeFtor)

```

qed

lemma *HomFtorObj*:

assumes *LSCategory C*

and $X \in \text{obj}_C$ **and** $Y \in \text{obj}_C$

shows $(\text{Hom}_C[X, -]) \text{@@} Y = \text{Hom}_C X Y$

proof–

have *FunctorM* $(\text{Hom}P_C[X, -])$ **using** *assms* **by** (*simp add: HomFtorFtor'*)

hence *1: PreFunctor* $(\text{Hom}P_C[X, -])$ **by** (*simp add: FunctorM-def*)

moreover **have** *CatDom* $(\text{Hom}P_C[X, -]) = C$ **by** (*simp add: HomFtor'-def*)

ultimately **have** $(\text{Hom}_C[X, -]) \text{@@} Y = (\text{Hom}P_C[X, -]) \text{@@} Y$ **using** *assms*
by (*simp add: MakeFtorObj HomFtor-def*)

thus *?thesis* **using** *assms 1* **by** (*simp add: HomFtorObj'*)

qed

definition

HomFtorMapContra $:: ('o, 'm, 'a) \text{LSCategory-scheme} \Rightarrow 'm \Rightarrow 'o \Rightarrow \text{ZF} (\text{Hom}C_1[-, -]$
[65,65] 65) **where**

HomFtorMapContra $C g X \equiv \text{ZFfun} (\text{Hom}_C (\text{cod}_C g) X) (\text{Hom}_C (\text{dom}_C g) X)$
 $(\lambda f . m2z_C (g ;;_C (z2m_C f)))$

definition

HomFtorContra' $:: ('o, 'm, 'a) \text{LSCategory-scheme} \Rightarrow 'o \Rightarrow$

$('o, \text{ZF}, 'm, \text{ZF}, (\text{mor2ZF} :: 'm \Rightarrow \text{ZF}, \dots :: 'a), \text{unit}) \text{Functor} (\text{Hom}P_1[-, -]$ [65]
65) **where**

HomFtorContra' $C X \equiv ($

$\text{CatDom} = (\text{Op } C),$

$\text{CatCod} = \text{SET} ,$

$\text{MapM} = \lambda g . \text{Hom}C_C[g, X]$

$)$

definition *HomFtorContra* $(\text{Hom}_1[-, -]$ [65] 65) **where** *HomFtorContra* $C X \equiv$
MakeFtor $(\text{HomFtorContra}' C X)$

lemma *HomContraAt*: $x \in | (\text{Hom}_C (\text{cod}_C f) X) \implies (\text{Hom}C_C[f, X]) \text{@@} x =$
 $m2z_C (f ;;_C (z2m_C x))$

by (*simp add: HomFtorMapContra-def ZFfunApp*)

lemma *mor2ZF-Op*: $\text{mor2ZF} (\text{Op } C) = \text{mor2ZF } C$

apply (*cases C*)

apply (*simp add: OppositeCategory-def*)

done

lemma *mor-Op*: $\text{mor}_{\text{Op } C} = \text{mor}_C$ **by** (*simp add: OppositeCategory-def*)

lemma *obj-Op*: $\text{obj}_{\text{Op } C} = \text{obj}_C$ **by** (*simp add: OppositeCategory-def*)

lemma *ZF2mor-Op*: $\text{ZF2mor} (\text{Op } C) f = \text{ZF2mor } C f$

by (*simp add: ZF2mor-def mor2ZF-Op mor-Op*)

lemma *mapsTo-Op*: $f \text{ maps}_{Op\ C} Y \text{ to } X = f \text{ maps}_C X \text{ to } Y$
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 $\dots = (Hom_{Op C} X Y)$ **using** *assms(1) 1 2* **by** (*simp add: LSCategory-Op HomFtorObj*)

also have $\dots = (Hom_C Y X)$ **by** (*simp add: Hom-Op*)

finally show *?thesis* .

qed

lemma *HomCHomOp*: $Hom_C C[g, X] = Hom_{Op C}[X, g]$

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 $Hom_C C[f, X]$ *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 $Hom_C C[f, X] = Hom_{Op C}[X, f]$ **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** $X \in \text{obj}_C$ **and** $f \in \text{mor}_C$
shows $\text{Hom}_C C[f, X] \in \text{Mor SET}$ **and** $\text{dom}_{SET} (\text{Hom}_C C[f, X]) = \text{Hom}_C (\text{cod}_C f) X$
and $\text{cod}_{SET} (\text{Hom}_C C[f, X]) = \text{Hom}_C (\text{dom}_C f) X$
proof–
have $\text{Hom}_C C[f, X]$ *maps*_{SET} $\text{Hom}_C (\text{cod}_C f) X$ **to** $\text{Hom}_C (\text{dom}_C f) X$ **using**
assms **by** (*simp add: HomFtorContraMapsTo*)
thus $\text{Hom}_C C[f, X] \in \text{Mor SET}$ **and** $\text{dom}_{SET} (\text{Hom}_C C[f, X]) = \text{Hom}_C (\text{cod}_C f) X$
and $\text{cod}_{SET} (\text{Hom}_C C[f, X]) = \text{Hom}_C (\text{dom}_C f) X$
by *auto*
qed

lemma *HomContraMor*:

assumes *LSCategory C* **and** $f \in \text{Mor } C$
shows $(\text{Hom}_C [-, X]) \#\# f = \text{Hom}_C C[f, X]$
by (*simp add: HomFtorContra-def HomFtorContra'-def MakeFtor-def assms OppositeCategory-def*)

lemma *HomCHom*:

assumes *LSCategory C* **and** $f \in \text{Mor } C$ **and** $g \in \text{Mor } C$
shows $(\text{Hom}_C C[g, \text{dom}_C f]) \;\text{;;}\;_{SET} (\text{Hom}_C [\text{dom}_C g, f]) = (\text{Hom}_C [\text{cod}_C g, f]) \;\text{;;}\;_{SET} (\text{Hom}_C C[g, \text{cod}_C f])$
proof–
have *ObjDf*: $\text{dom}_C f \in \text{obj}_C$ **and** *ObjDg*: $\text{dom}_C g \in \text{obj}_C$ **using** *assms* **by** (*simp add: Category.Cdom*)
have *ObjCg*: $\text{cod}_C g \in \text{obj}_C$ **and** *ObjCf*: $\text{cod}_C f \in \text{obj}_C$ **using** *assms* **by** (*simp add: Category.Ccod*)
have $(\text{Hom}_C C[g, \text{dom}_C f]) \;\text{;;}\;_{SET} (\text{Hom}_C [\text{dom}_C g, f]) = (\text{Hom}_C C[g, \text{dom}_C f]) \;|\; o \;|$
 $(\text{Hom}_C [\text{dom}_C g, f])$
proof–
have $(\text{Hom}_C C[g, \text{dom}_C f]) \approx\;>_{SET} (\text{Hom}_C [\text{dom}_C g, f])$
proof(*rule CompDefinedI*)
show $\text{Hom}_C C[g, \text{dom}_C f] \in \text{Mor SET}$ **using** *assms* *ObjDg* **by** (*simp add: HomFtorMor*)
show $\text{Hom}_C C[g, \text{dom}_C f] \in \text{Mor SET}$ **using** *assms* *ObjDf* **by** (*simp add: HomFtorContraMor*)
show $\text{cod}_{SET} (\text{Hom}_C C[g, \text{dom}_C f]) = \text{dom}_{SET} (\text{Hom}_C [\text{dom}_C g, f])$ **using**
assms *ObjDg* *ObjDf*
by (*simp add: HomFtorMor HomFtorContraMor*)

```

qed
thus ?thesis by (simp add: SET-def SET'-def MakeCatComp2)
qed
also have ... = ZFfun (HomC (codC g) (domC f)) (HomC (domC g) (codC f))
  ((λ h . m2zC ((z2mC h) ;;C f)) o (λ h . m2zC (g ;;C (z2mC h))))
proof (simp add: HomFtorMapContra-def HomFtorMap-def, rule ZFfunComp,
rule allI, rule impI)
{
  fix x assume aa: x |∈| (HomC (codC g) (domC f))
  show (m2zC (g ;;C (z2mC x))) |∈| (HomC (domC g) (domC f))
  proof (rule LSCategory.m2zz2m, simp-all add: assms(1) ObjDg ObjDf)
    have g mapsC (domC g) to (codC g) using assms by auto
    moreover have (z2mC x) mapsC (codC g) to (domC f) using aa ObjCg
ObjDf assms(1)
    by (simp add: LSCategory.z2mm2z)
    ultimately show g ;;C (z2mC x) mapsC (domC g) to (domC f) using
assms(1)
    by (simp add: Category.Ccompt)
  }
qed
}
qed
also have ... = ZFfun (HomC (codC g) (domC f)) (HomC (domC g) (codC f))
  ((λ h . m2zC (g ;;C (z2mC h))) o (λ h . m2zC ((z2mC h) ;;C f)))
proof (rule ZFfun-ext, rule allI, rule impI)
{
  fix h assume aa: h |∈| (HomC (codC g) (domC f))
  show ((λ h . m2zC ((z2mC h) ;;C f)) o (λ h . m2zC (g ;;C (z2mC h)))) h =
  ((λ h . m2zC (g ;;C (z2mC h))) o (λ h . m2zC ((z2mC h) ;;C f))) h
  proof-
    have MapsTo1: (z2mC h) mapsC (codC g) to (domC f) using assms(1)
ObjCg ObjDf aa by (simp add: LSCategory.z2mm2z)
    have CompDef1: (z2mC h) ≈>C f
    proof (rule CompDefinedI)
      show f ∈ morC using assms by simp
      show (z2mC h) ∈ morC and codC (z2mC h) = domC f using MapsTo1
by auto
    qed
    have CompDef2: g ≈>C (z2mC h)
    proof (rule CompDefinedI)
      show g ∈ morC using assms by simp
      thus (z2mC h) ∈ morC and codC g = domC (z2mC h) using MapsTo1
by auto
    qed
    have c1: (z2mC h) ;;C f ∈ Mor C using assms CompDef1 by (simp add:
Category.MapsToMorDomCod)
    have c2: g ;;C (z2mC h) ∈ Mor C using assms CompDef2 by (simp add:
Category.MapsToMorDomCod)
    have g ;;C (z2mC (m2zC ((z2mC h) ;;C f))) = g ;;C ((z2mC h) ;;C f) using
assms(1) c1

```

```

    by (simp add: LSCategory.m2zz2mInv)
    also have ... = (g ;;C (z2mC h)) ;;C f using CompDef1 CompDef2 assms
by (simp add: Category.Cassoc)
    also have ... = (z2mC (m2zC (g ;;C (z2mC h)))) ;;C f using assms(1) c2
    by (simp add: LSCategory.m2zz2mInv)
    finally have g ;;C (z2mC (m2zC ((z2mC h) ;;C f))) = (z2mC (m2zC (g ;;C
(z2mC h)))) ;;C f .
    thus ?thesis by simp
  qed
}
qed
also have ... = (HomC[codC g, f]) |o| (HomC C[g, codC f])
proof(simp add: HomFtorMapContra-def HomFtorMap-def, rule ZFfunComp[THEN
sym], rule allI, rule impI)
{
  fix x assume aa: x |∈| (HomC codC g domC f)
  show m2zC ((z2mC x) ;;C f) |∈| (HomC codC g codC f)
  proof(rule LSCategory.m2zz2m, simp-all add: assms(1) ObjCg ObjCf)
    have f mapsC (domC f) to (codC f) using assms by auto
    moreover have (z2mC x) mapsC (codC g) to (domC f) using aa ObjCg
ObjDf assms(1)
    by (simp add: LSCategory.z2mm2z)
    ultimately show (z2mC x) ;;C f mapsC codC g to codC f using assms(1)
    by (simp add: Category.Ccompt)
  qed
}
qed
also have ... = (HomC[codC g, f]) ;;SET (HomC C[g, codC f])
proof-
  have (HomC[codC g, f]) ≈>SET (HomC C[g, codC f])
  proof(rule CompDefinedI)
    show HomC[codC g, f] ∈ Mor SET using assms ObjCg by (simp add:
HomFtorMor)
    show HomC C[g, codC f] ∈ Mor SET using assms ObjCf by (simp add:
HomFtorContraMor)
    show codSET (HomC[codC g, f]) = domSET (HomC C[g, codC 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 $YFtorNT' C f \equiv (\langle NTDom = Hom_C[-, dom_C f], NTCod = Hom_C[-, cod_C f] \rangle,$
 $NatTransMap = \lambda B . Hom_C[B, f])$

definition $YFtorNT C f \equiv MakeNT (YFtorNT' C f)$

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 $YFtorNTApp1: \text{assumes } X \in Obj (NTCatDom (YFtorNT C f)) \text{ shows}$
 $(YFtorNT C f) \$\$ X = Hom_C[X, f]$

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

$YFtor' C \equiv (\langle$
 $CatDom = C,$
 $CatCod = CatExp (Op C) SET,$
 $MapM = \lambda f . YFtorNT C f$
 \rangle

definition $YFtor C \equiv MakeFtor(YFtor' C)$

lemmas $YFtor-defs = YFtor'-def YFtor-def MakeFtor-def$

lemma $YFtorNTNatTrans'$:

assumes *LSCategory C* **and** $f \in Mor C$

shows $NatTransP (YFtorNT' C f)$

proof(*auto simp only: NatTransP-def*)

have $Fd: Ftor (NTDom (YFtorNT' C f)) : (Op C) \longrightarrow SET$ **using** *assms*

by (*simp add: HomFtorContraFtor Category.Cdom YFtorNT'-def*)

have $Fc: Ftor (NTCod (YFtorNT' C f)) : (Op C) \longrightarrow SET$ **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) $$ X mapsNTCatCod (YFtorNT' C f) (NTDom (YFtorNT'
    C f) @@ X) to (NTCod (YFtorNT' C f) @@ X)
    proof-
      have Obj: X ∈ Obj C using a by (simp add: NTCatDom-def YFtorNT'-def
      HomFtorContraDom OppositeCategory-def)
      have H1: (HomC[-, domC f]) @@ X = HomC X domC f using assms Obj
    by(simp add: HomFtorOpObj Category.Cdom)
      have H2: (HomC[-, codC f]) @@ X = HomC X codC f using assms Obj
    by(simp add: HomFtorOpObj Category.Ccod)
      have HomC[X, f] mapsSET (HomC X domC f) to (HomC X codC f) using
    assms Obj by (simp add: HomFtorMapsTo)
      thus ?thesis using H1 H2 by(simp add: YFtorNT'-def NTCatCod-def NT-
    CatDom-def HomFtorContraCod)
    qed
  }
  {
    fix g X Y assume a: g mapsNTCatDom (YFtorNT' C f) X to Y
    show ((NTDom (YFtorNT' C f)) ## g) ;;NTCatCod (YFtorNT' C f) (YFtorNT'
    C f $$ Y) =
      ((YFtorNT' C f) $$ X) ;;NTCatCod (YFtorNT' C f) (NTCod (YFtorNT' C
    f) ## g)
    proof-
      have M1: g mapsOp C X to Y using a by (auto simp add: NTCatDom-def
      YFtorNT'-def HomFtorContraDom)
      have D1: domC g = Y and C1: codC 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[g, domC f]) = (HomC[-, domC f]) ## g
      and H2: (HomC[g, codC f]) = (HomC[-, codC f]) ## g using M1
      by (auto simp add: HomFtorContra-def HomFtorContra'-def MakeFtor-def)
      have (HomC[g, domC f]) ;;SET (HomC[domC g, f]) = (HomC[codC g, f])
    ;;SET (HomC[g, codC f]) using assms morf morg
      by (simp add: HomCHom)
      hence ((HomC[-, domC f]) ## g) ;;SET (HomC[Y, f]) = (HomC[X, f]) ;;SET
    ((HomC[-, codC 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 $f \in \text{Mor } C$
 shows *NatTrans* (*YFtorNT C f*)

by (simp add: assms YFtorNTNatTrans' YFtorNT-def MakeNT)

lemma YFtorNTMor:

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

lemma YFtorNTCompDef:

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

lemma *PreSheafCat*: $LSCategory\ C \implies Category\ (CatExp\ (Op\ C)\ SET)$
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 $\dots = dom_{CatExp\ (Op\ C)\ SET}\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-, X]))$
using *CAT HomObj*
by (*simp add: Category.CatIdDomCod*)
also have $\dots = NTDom\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-, X]))$ **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 $\dots = cod_{CatExp\ (Op\ C)\ SET}\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-, X]))$
using *CAT HomObj*
by (*simp add: Category.CatIdDomCod*)
also have $\dots = NTCod\ (Id\ (CatExp\ (Op\ C)\ SET)\ (Hom_C[-, X]))$ **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 ( $YFtor' C$ ) ## ( $f \mathbin{::}_{CatDom (YFtor' C)} g$ ) = (( $YFtor' C$ ) ## f)
     $\mathbin{::}_{CatCod (YFtor' C)} ((YFtor' C) ## g)$ 
    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:  $YFtorNT C f \mathbin{::}_{CatExp (Op C) SET} YFtorNT C g \in Mor (CatExp
      (Op C) SET)$  using CAT CD2
      by (simp add:  $Category.MapsToMorDomCod$ )
      show  $NatTrans (YFtorNT C (f \mathbin{::}_C g))$  using assms by (simp add:  $Cate-
      gory.MapsToMorDomCod CD YFtorNTNatTrans$ )
      show  $NatTrans (YFtorNT C f \mathbin{::}_{CatExp (Op C) SET} YFtorNT C g)$  using
      Mor1 by (simp add:  $CatExpMorNT$ )
      show  $NTDom (YFtorNT C (f \mathbin{::}_C g)) = NTDom (YFtorNT C f \mathbin{::}_{CatExp (Op C) SET}$ 

```


$YFtorNT\ C\ g)$
proof-
have 1: $YFtorNT\ C\ f \in mor_{CatExp\ (Op\ C)\ SET}$ **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 $\dots = Hom_C[-, dom_C\ f]$ **using** $CD\ assms$ **by** (*simp* *add: Category.MapsToMorDomCod*)
also have $\dots = NTDom\ (YFtorNT\ C\ f)$ **by** (*simp* *add: YFtorNT-defs*)
also have $\dots = dom_{CatExp\ (Op\ C)\ SET}\ (YFtorNT\ C\ f)$ **using** 1 **by** (*simp* *add: CatExpDom*)
also have $\dots = dom_{CatExp\ (Op\ C)\ SET}\ (YFtorNT\ C\ f\ ;;_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ g)$ **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: $YFtorNT\ C\ g \in mor_{CatExp\ (Op\ C)\ SET}$ **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 $\dots = Hom_C[-, cod_C\ g]$ **using** $CD\ assms$ **by** (*simp* *add: Category.MapsToMorDomCod*)
also have $\dots = NTCod\ (YFtorNT\ C\ g)$ **by** (*simp* *add: YFtorNT-defs*)
also have $\dots = cod_{CatExp\ (Op\ C)\ SET}\ (YFtorNT\ C\ g)$ **using** 1 **by** (*simp* *add: CatExpCod*)
also have $\dots = cod_{CatExp\ (Op\ C)\ SET}\ (YFtorNT\ C\ f\ ;;_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ g)$ **using** $CD2\ CAT$
by (*simp* *add: Category.MapsToMorDomCod*)
finally show *?thesis* **using** $Mor1$ **by** (*simp* *add: CatExpCod*)
qed
{
fix X **assume** $a: X \in Obj\ (NTCatDom\ (YFtorNT\ C\ (f\ ;;_C\ g)))$
show $YFtorNT\ C\ (f\ ;;_C\ g)\ \$\$ X = (YFtorNT\ C\ f\ ;;_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ g)\ \$\$ X$
proof-
have $Obj: X \in Obj\ C$ **and** $ObjOp: X \in Obj\ (Op\ C)$ **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 $\dots = (Hom_C[X, f])\ ;;_{SET}\ (Hom_C[X, g])$ **using** $CD\ assms\ Obj$
by (*simp* *add: HomFtorDist*)
also have $\dots = ((YFtorNT\ C\ f)\ \$\$ X)\ ;;_{SET}\ ((YFtorNT\ C\ g)\ \$\$ X)$ **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=HomC [-,X] in Set.rev-bezI)
    have X ∈ Obj C using a by(simp add: YFtor'-def)
    thus HomC [-,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)) (HomC [-,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 = HomC [-,X]
  proof(rule PreFunctor.FmToFo, simp-all add: assms YFtor'Obj1 YFtorPreFtor)
    have X ∈ Obj C using assms by(simp add: YFtor'-def)
    thus HomC [-,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 mapsCatDom (YFtor' C) X to Y
      show YFtor' C ## f mapsCatCod (YFtor' C) YFtor' C @@ X to YFtor' C
@@ Y
    }
    proof-
      have Mor1: f mapsC 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)
    qed
  qed

```

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** $Ftor (YFtor C) : C \longrightarrow (CatExp (Op C) SET)$
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** $X \in Obj C$
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** $X \in Obj C$ **and** $Y \in Obj C$
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 $\dots = ((YFtor C @@ Y) @@ X)$ **using** *assms* **by** (*simp add: YFtorObj*)
finally show *?thesis* **by** *simp*
qed

lemma *YFtorMor*: $\llbracket LSCategory C ; f \in Mor C \rrbracket \implies (YFtor C) \#\# f = YFtorNT C f$
by (*simp add: YFtor-defs MakeFtorMor*)

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

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 $\dots = CatDom (Hom_C[-, X])$ **using** $assms$ **by** $(simp\ add: YFtorObj\ YMapInv'-def)$
also have $\dots = Op\ C$ **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 $\dots = (YMapInv'\ C\ X\ F\ x) \ \$\$ B$ **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 $(YMap\ C\ X\ \eta) \mid \in \mid (F \ @\@ X)$
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 $\eta \ \$\$ X$ $maps_{SET} ((YFtor\ C \ @\@ X) \ @\@ X)$ $to (F \ @\@ X)$ **by**
 $simp$
moreover have $(m2z_C (Id\ C\ X)) \mid \in \mid ((YFtor\ C \ @\@ X) \ @\@ X)$
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 $((\eta\ \$\$ X)\ |\@|\ (m2z_C\ (Id\ C\ X)))\ |\in|\ (F\ @\@ X)$ **by** *(simp add: SETfunDomAppCod)*
qed

lemma *YMapInvNatTransP:*

assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $xobj: X \in Obj\ C$
and $xinF: x\ |\in|\ (F\ @\@ X)$
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 $hf: Ftor\ (YFtor\ C\ @\@ X) : (Op\ C) \longrightarrow SET$ **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** $yobja: Y \in Obj\ (CatDom\ ((YFtor\ C)\ @\@ X))$
show $ZFfun\ (Hom_C\ Y\ X)\ (F\ @\@ Y)\ (\lambda f. (F\ \#\#\ (z2m_C\ f))\ |\@|\ x)$ *maps* $CatCod\ F$
((YFtor C @@ X) @@ Y) to (F @@ Y)
proof*(simp add: cf, rule MapsToI)*
have $yobj: Y \in Obj\ C$ **using** $yobja$ **by** *(simp add: OppositeCategory-def)*
have $zffun: isZFfun\ (ZFfun\ (Hom_C\ Y\ X)\ (F\ @\@ Y)\ (\lambda f. (F\ \#\#\ z2m_C\ f))\ |\@|\ x))$
proof*(rule SETfun, rule allI, rule impI)*
{
fix y **assume** $yhom: y\ |\in|\ (Hom_C\ Y\ X)$ **show** $(F\ \#\#\ (z2m_C\ y))\ |\@|\ x\ |\in|\ (F\ @\@ Y)$
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 (HomC Y X) (F @@ Y) (λf. (F ## z2mCf) |@| x) ∈ morSET
using zffun
  by(simp add: SETmor)
  show codSET ZFfun (HomC Y X) (F @@ Y) (λf. (F ## z2mCf) |@| x) =
F @@ Y using zffun
  by(simp add: SETcod)
  have (HomC Y X) = (YFtor C @@ X) @@ Y using assms yobj by (simp
add: YFtorObj2)
  thus domSET ZFfun (HomC Y X) (F @@ Y) (λf. (F ## z2mCf) |@| x) =
(YFtor C @@ X) @@ Y using zffun
  by(simp add: SETdom)
  qed
}
{
  fix f Z Y assume fmaps: f mapsCatDom ((YFtor C) @@ X) Z to Y
  have fmapsa: f mapsOp C Z to Y using fmaps dy by simp
  hence fmapsb: f mapsC Y to Z by (rule MapsToOpOp)
  hence fmor: f ∈ Mor C and fdom: domC f = Y and fcod: codC f = Z by
(auto simp add: OppositeCategory-def)
  hence hc: (HomC[-,X]) ## f = (HomCC[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) mapsSET (F @@ Z) to (F @@ Y) using assms fmapsa
by (simp add: FunctorMapsTo)
  have Fzmaps: ∧ h A B . [h |∈| (HomC A B) ; A ∈ Obj C ; B ∈ Obj C] ⇒
(F ## (z2mC h)) mapsSET (F @@ B) to (F @@ A)
  proof-
    fix h A B assume h: h |∈| (HomC A B) and oA: A ∈ Obj C and oB: B ∈
Obj C
    have (z2mC h) mapsC A to B using assms h oA oB by (simp add: LSCate-
gory.z2mm2z)
    hence (z2mC h) mapsOp C B to A by (rule MapsToOp)
    thus (F ## (z2mC h)) mapsSET (F @@ B) to (F @@ A) using assms by
(simp add: FunctorMapsTo)
  qed
  have hHomF: ∧h. h |∈| (HomC Z X) ⇒ (F ## (z2mC h)) |@| x |∈| (F @@
Z) using xobj zobj xinF
  proof-
    fix h assume h: h |∈| (HomC Z X)
    have (F ## (z2mC h)) mapsSET (F @@ X) to (F @@ Z) using xobj zobj
h by (simp add: Fzmaps)
    thus (F ## (z2mC h)) |@| x |∈| (F @@ Z) using assms by (simp add:
SETfunDomAppCod)
  qed
  have Ff: F ## f = ZFfun (F @@ Z) (F @@ Y) (λh. (F ## f) |@| h) using
Ffmaps by (simp add: SETZFfun)
  have compdefa: ZFfun (HomC Z X) (HomC Y X) (λh. m2zC (f ;;C (z2mC h)))

```

```

≈> SET
  ZFfun (HomC Y X) (F @@ Y) (λh. (F ## (z2mC h)) |@| x)
proof(rule CompDefinedI, simp-all add: SETmor[THEN sym])
show isZFfun (ZFfun (HomC Z X) (HomC Y X) (λh. m2zC (f ;;C (z2mC h))))
proof(rule SETfun, rule allI, rule impI)
  fix h assume h: h |∈| (HomC Z X)
    have (z2mC h) mapsC Z to X using assms h xobj zobj by (simp add:
LSCategory.z2mm2z)
    hence f ;;C (z2mC h) mapsC Y to X using fmapsb assms(1) by (simp add:
Category.Ccompt)
    thus (m2zC (f ;;C (z2mC h))) |∈| (HomC Y X) using assms by (simp add:
LSCategory.m2zz2m)
  qed
  moreover show isZFfun (ZFfun (HomC Y X) (F @@ Y) (λh. (F ##
(z2mC h)) |@| x))
proof(rule SETfun, rule allI, rule impI)
  fix h assume h: h |∈| (HomC Y X)
    have (F ## (z2mC h)) mapsSET (F @@ X) to (F @@ Y) using xobj yobj
h by (simp add: Fzmaps)
    thus (F ## (z2mC h)) |@| x |∈| (F @@ Y) using assms by (simp add:
SETfunDomAppCod)
  qed
  ultimately show codSET(ZFfun (HomC Z X) (HomC Y X) (λh. m2zC (f
;;C (z2mC h)))) =
    domSET(ZFfun (HomC Y X) (F @@ Y) (λh. (F ## (z2mC h)) |@| x)) by
(simp add: SETcod SETdom)
  qed
  have compdefb: ZFfun (HomC Z X) (F @@ Z) (λh. (F ## (z2mC h)) |@| x)
≈> SET
  ZFfun (F @@ Z) (F @@ Y) (λh. (F ## f) |@| h)
proof(rule CompDefinedI, simp-all add: SETmor[THEN sym])
  show isZFfun (ZFfun (HomC Z X) (F @@ Z) (λh. (F ## (z2mC h)) |@|
x)) using hHomF by (simp add: SETfun)
  moreover show isZFfun (ZFfun (F @@ Z) (F @@ Y) (λh. (F ## f) |@|
h))
proof(rule SETfun, rule allI, rule impI)
  fix h assume h: h |∈| (F @@ Z)
    have F ## f mapsSET F @@ Z to F @@ Y using Ffmaps .
    thus (F ## f) |@| h |∈| (F @@ Y) using h by (simp add: SETfunDomAp-
pCod)
  qed
  ultimately show codSET (ZFfun (HomC Z X) (F @@ Z) (λh. (F ##
(z2mC h)) |@| x)) =
    domSET (ZFfun (F @@ Z) (F @@ Y) (λh. (F ## f) |@| h)) by (simp
add: SETcod SETdom)
  qed
  have ZFfun (HomC Z X) (HomC Y X) (λh. m2zC (f ;;C (z2mC h))) ;;SET
ZFfun (HomC Y X) (F @@ Y) (λh. (F ## (z2mC h)) |@| x) =
  ZFfun (HomC Z X) (HomC Y X) (λh. m2zC (f ;;C (z2mC 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** $h: h | \in | (Hom_C Z X)$
show $(m2z_C (f ;;_C (z2m_C h))) | \in | (Hom_C Y X)$
proof-
have $Z \in Obj C$ **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.Ccomp*)
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** $h: h | \in | (Hom_C Z X)$
have $zObj: Z \in Obj C$ **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 $(z2m_C h) \in Mor C$ **and** $dom_C (z2m_C h) = cod_C f$ **using** *fcod* **by** *auto*
hence *CompDef-hf*: $f \approx>_C (z2m_C h)$ **using** *fmor* **by** *auto*
hence *CompDef-hfOp*: $(z2m_C h) \approx>_{Op C} f$ **by** (*simp add: CompDefOp*)
hence *CompDef-FhfOp*: $(F ## (z2m_C h)) \approx>_{SET} (F ## f)$ **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 $xin: x | \in | |dom|(F ## (z2m_C h))$ **using** *assms* **by** (*simp add: SETmapsTo*)
have $(f ;;_C (z2m_C h)) \in Mor C$ **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 ... = $(F ## ((z2m_C h) ;;_{Op C} f)) |@| x$ **by** (*simp add: Opposite-Category-def*)
also have ... = $((F ## (z2m_C h)) ;;_{SET} (F ## f)) |@| x$ **using** *assms* *CompDef-hfOp* **by** (*simp add: FunctorComp*)
also have ... = $(F ## f) |@| ((F ## (z2m_C h)) |@| x)$ **using** *CompDef-FhfOp*

xin **by**(*rule SETCompAt*)
finally show ($F \#\# (z2m_C (m2z_C (f \; ;_C (z2m_C h))))$) $|\@| x = (F \#\# f)$
 $|\@| ((F \#\# (z2m_C h)) \; |\@| x)$.
}
qed
also have ... = $ZFfun (Hom_C Z X) (F \@\@ Z) (\lambda h. (F \#\# (z2m_C h)) \; |\@| x)$
 $|\@|$
 $ZFfun (F \@\@ Z) (F \@\@ Y) (\lambda h. (F \#\# f) \; |\@| h)$
by(*rule ZFfunComp[THEN sym], rule allI, rule impI, simp add: hHomF*)
also have ... = $ZFfun (Hom_C Z X) (F \@\@ Z) (\lambda h. (F \#\# (z2m_C h)) \; |\@| x)$
 $;\;_{SET} (F \#\# f)$
using *Ff compdefb by (simp add: SETComp)*
finally show ($((YFtor C) \@\@ X) \#\# f$) $;\;_{CatCod F} ZFfun (Hom_C Y X) (F$
 $\@\@ Y) (\lambda f. (F \#\# (z2m_C f)) \; |\@| x) =$
 $ZFfun (Hom_C Z X) (F \@\@ Z) (\lambda f. (F \#\# (z2m_C f)) \; |\@| x) \; ;\;_{CatCod F}$
 $(F \#\# f)$
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*
 $|\in| (F \@\@ X)$
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 $x \; |\in| (F \@\@ X)$
shows $NT (YMapInv C X F x) : (YFtor C \@\@ X) \Longrightarrow F$
proof(*auto simp only: nt-abbrev-def*)
show *NatTrans (YMapInv C X F x) using assms by (simp add: YMapInvNat-*
Trans)
show $NTDom (YMapInv C X F x) = YFtor C \@\@ X$ **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 Ftor: Ftor F : (Op C) \longrightarrow SET and XObj:*
 $X \in Obj C$
and $NT \; \eta : (YFtor C \@\@ X) \Longrightarrow F$
shows $YMapInv C X F (YMap C X \eta) = \eta$
proof(*rule NatTransExt*)
have ($YMap C X \eta$) $|\in| (F \@\@ X)$ **using** *assms by (simp add: YMapImage)*
hence $1: NT (YMapInv C X F (YMap C X \eta)) : (YFtor C \@\@ X) \Longrightarrow F$ **using**
assms by (simp add: YMapInvImage)
thus $NatTrans (YMapInv C X F (YMap C X \eta))$ **by** *auto*

show $NatTrans \eta$ **using** $assms$ **by** $auto$
have $NTDYI$: $NTDom (YMapInv C X F (YMap C X \eta)) = (YFtor C @@ X)$
using 1 **by** $auto$
moreover **have** $NTDeta$: $NTDom \eta = (YFtor C @@ X)$ **using** $assms$ **by** $auto$
ultimately **show** $NTDom (YMapInv C X F (YMap C X \eta)) = NTDom \eta$ **by**
 $simp$
have $NTCod (YMapInv C X F (YMap C X \eta)) = F$ **using** 1 **by** $auto$
moreover **have** $NTCeta$: $NTCod \eta = F$ **using** $assms$ **by** $auto$
ultimately **show** $NTCod (YMapInv C X F (YMap C X \eta)) = NTCod \eta$ **by**
 $simp$
 $\{$
fix Y **assume** $Yobja$: $Y \in Obj (NTCatDom (YMapInv C X F (YMap C X \eta)))$
have CCF : $CatCod F = SET$ **using** $assms$ **by** $auto$
have $Ftor (Hom_C[-, X]) : (Op C) \longrightarrow SET$ **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 $NTCatDom (YMapInv C X F (YMap C X \eta)) = Op C$ **using** $LSCat$
 $XObj NTDYI CDH$ **by** ($simp$ add : $NTCatDom-def$)
hence $YObjOp$: $Y \in Obj (Op C)$ **using** $Yobja$ **by** $simp$
hence $YObj$: $Y \in Obj C$ **and** $XObjOp$: $X \in Obj (Op C)$ **using** $XObj$ **by** ($simp$
 add : $OppositeCategory-def$)
have $yinv-mapsTo$: $((YMapInv C X F (YMap C X \eta)) \$\$ Y) maps_{SET} (Hom_C Y$
 $X) to (F @@ Y)$
proof-
have $((YMapInv C X F (YMap C X \eta)) \$\$ Y) maps_{SET} ((YFtor C @@ X)$
 $@@ Y) to (F @@ Y)$
using $1 CCF CDYF YObjOp NatTransMapsTo[of (YMapInv C X F (YMap$
 $C X \eta)) (YFtor C @@ X) F Y]$ **by** $simp$
thus $?thesis$ **using** $LSCat XObj YObj$ **by** ($simp$ add : $YFtorObj2$)
qed
have $eta-mapsTo$: $(\eta \$\$ Y) maps_{SET} (Hom_C Y X) to (F @@ Y)$
proof-
have $(\eta \$\$ Y) maps_{SET} ((YFtor C @@ X) @@ Y) to (F @@ Y)$
using $NT CDYF CCF YObjOp NatTransMapsTo[of \eta (YFtor C @@ X) F$
 $Y]$ **by** $simp$
thus $?thesis$ **using** $LSCat XObj YObj$ **by** ($simp$ add : $YFtorObj2$)
qed
show $(YMapInv C X F (YMap C X \eta)) \$\$ Y = \eta \$\$ Y$
proof($rule ZFfunExt$)
show $|dom|(YMapInv C X F (YMap C X \eta)) \$\$ Y = |dom|(\eta \$\$ Y)$
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 |∈| |dom|(YMapInv C X F (YMap C X η) $$ Y)
  have fHom: f |∈| (HomC Y X) using yinv-mapsTo fdomYinv by (simp
add: SETmapsTo)
  hence fMapsTo: (z2mC f) mapsC Y to X using assms YObj by (simp add:
LSCategory.z2mm2z)
  hence fCod: (codC (z2mC f)) = X and fDom: (domC (z2mC f)) = Y and
fMor: (z2mC f) ∈ Mor C by auto
  have (YMapInv C X F (YMap C X η) $$ Y) |@| f =
(F ## (z2mC f)) |@| ((η $$ X) |@| (m2zC (Id C X)))
  using fHom assms YObj by (simp add: ZFfunApp YMapInvApp YMap-def)
  also have ... = ((η $$ X) ;;SET (F ## (z2mC f))) |@| (m2zC (Id C X))
  proof-
  have aa: (η $$ X) mapsSET ((YFtor C @@ X) @@ X) to (F @@ X)
  using NT CDYF CCF YObjOp XObjOp NatTransMapsTo[of η (YFtor
C @@ X) F X] by simp
  have bb: (F ## (z2mC f)) mapsSET (F @@ X) to (F @@ Y)
  using fMapsTo Ftor by (simp add: MapsToOp FunctorMapsTo)
  have (η $$ X) ≈SET (F ## (z2mC f)) using aa bb by (simp add:
MapsToCompDef)
  moreover have (m2zC (Id C X)) |∈| |dom| (η $$ X) using assms aa
  by (simp add: SETmapsTo YFtorObj2 Category.Cidm LSCategory.m2zz2m)
  ultimately show ?thesis by (simp add: SETCompAt)
  qed
  also have ... = ((HomC C[z2mC f, X]) ;;SET (η $$ Y)) |@| (m2zC (Id C
X))
  proof-
  have NTDom η = (HomC[-, X]) using NTDetta assms by (simp add:
YFtorObj)
  moreover hence NTCatDom η = Op C by (simp add: NTCatDom-def
HomFtorContraDom)
  moreover have NTCatCod η = SET using assms by (auto simp add:
NTCatCod-def)
  moreover have NatTrans η and NTCod η = F using assms by auto
  moreover have (z2mC f) mapsOp C X to Y
  using fMapsTo MapsToOp[where ?f = (z2mC f) and ?X = Y and ?Y
= X and ?C = C] by simp
  ultimately have (η $$ X) ;;SET (F ## (z2mC f)) = ((HomC[-, X]) ##
(z2mC f)) ;;SET (η $$ Y)
  using NatTransP.NatTrans[of η (z2mC f) X Y] by simp
  moreover have ((HomC[-, X]) ## (z2mC f)) = (HomC C[(z2mC f), X])
using assms fMor by (simp add: HomContraMor)
  ultimately show ?thesis by simp
  qed
  also have ... = (η $$ Y) |@| ((HomC C[z2mC f, X]) |@| (m2zC (Id C X)))
  proof-
  have (HomC C[z2mC f, X]) ≈SET (η $$ Y)
  using fCod fDom XObj LSCat fMor HomFtorContraMapsTo[of C X
z2mC f] eta-mapsTo by (simp add: MapsToCompDef)

```

moreover have $|dom| (Hom_C [z2m_C f, X]) = (Hom_C (cod_C (z2m_C f)))$
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 $... = (\eta \ \$\$ Y) |@| (m2z_C ((z2m_C f) ;;_C (z2m_C (m2z_C (Id C X))))$
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 $... = (\eta \ \$\$ Y) |@| (m2z_C ((z2m_C f) ;;_C (Id C X)))$
using *assms by (simp add: LSCategory.m2zz2mInv Category.CatIdInMor)*
also have $... = (\eta \ \$\$ Y) |@| (m2z_C (z2m_C f))$ **using** *assms(1) fCod fMor Category.Cidr[of C (z2m_C f)] by (simp)*
also have $... = (\eta \ \$\$ Y) |@| f$ **using** *assms YObj fHom by (simp add: LSCategory.z2mm2z)*
finally show $(YMapInv C X F (YMap C X \eta) \ \$\$ Y) |@| f = (\eta \ \$\$ Y) |@| f$.
}
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 $(?η \ \$\$ X) = ZFfun (Hom_C X X) (F @@@ X) (\lambda f . (F \#\# (z2m_C f)) |@| x)$ **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: Category.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 $\dots = x$ **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** $ObjZ1: Z \in Obj (NTCatDom (YFtorNT' C f))$
have $ObjZ2: Z \in Obj C$ **using** $ObjZ1$ **by** (*simp add: YFtorNT'-def NTCat-Dom-def OppositeCategory-def HomFtorContraDom*)
moreover have $ObjX: X \in Obj C$ **and** $ObjY: Y \in Obj C$ **using** *assms* **by** (*simp add: Category.MapsToObj*)+
moreover
{
fix x **assume** $x: x | \in | (Hom_C Z X)$
have $m2z_C ((z2m_C x) ;;_C f) = ((Hom_C[-, Y]) \#\# (z2m_C x)) |@| (m2z_C f)$
proof-
have $morf: f \in Mor C$ **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 $morf: (z2m_C x) \in Mor C$ **by** *auto*
hence $(Hom_C[-, Y]) \#\# (z2m_C x) = (Hom_C C[(z2m_C x), Y])$ **using** *assms*
by (*simp add: HomContraMor*)
moreover have $(Hom_C C[(z2m_C x), Y]) |@| (m2z_C f) = m2z_C ((z2m_C x) ;;_C (m2z_C (m2z_C f)))$
proof (*rule HomContraAt*)
have $cod_C (z2m_C x) = X$ **using** $mapsx$ **by** *auto*
thus $(m2z_C f) | \in | (Hom_C (cod_C (z2m_C x)) Y)$ **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 $(YFtorNT' C f) \text{ \#\# } Z = (YMapInv' C X (Hom_C[-, Y])$
 $(m2z_C f)) \text{ \#\# } Z$ **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 $YFtor C \text{ \#\# } f = YMapInv C X (YFtor C @@ Y) (m2z_C f)$
proof-
have $f \in Mor C$ **using** *assms* **by** *auto*
moreover have $Y \in Obj C$ **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** $X \in Obj C$ **and** $Y \in Obj C$
and $NT \eta : (YFtor C @@ X) \implies (YFtor C @@ Y)$
shows $YFtor C \text{ \#\# } (z2m_C (YMap C X \eta)) = \eta$
and $z2m_C (YMap C X \eta)$ *maps_C X to Y*
proof-
have $ftor: Ftor (YFtor C @@ Y) : (Op C) \longrightarrow SET$ **using** *assms* **by** (*simp*
add: YFtorObj HomFtorContraFtor)
hence $(YMap C X \eta) \in | \in | ((YFtor C @@ Y) @@ X)$ **using** *assms* **by** (*simp add:*
YMapImage)
hence $yh: (YMap C X \eta) \in | \in | (Hom_C X Y)$ **using** *assms* **by** (*simp add:*
YFtorObj2)
thus $(z2m_C (YMap C X \eta))$ *maps_C X to Y* **using** *assms* **by** (*simp add: LSCat-*
egory.z2mm2z)
hence $YFtor C \text{ \#\# } (z2m_C (YMap C X \eta)) = YMapInv C X (YFtor C @@ Y)$
 $(m2z_C (z2m_C (YMap C X \eta)))$
using *assms* **by** (*simp add: YMapYoneda*)
also have $\dots = YMapInv C X (YFtor C @@ Y) (YMap C X \eta)$
using *assms* **by** (*simp add: LSCategory.z2mm2z*)
finally show $YFtor C \text{ \#\# } (z2m_C (YMap C X \eta)) = \eta$ **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 $YFtor C \text{ \#\# } f = YFtor C \text{ \#\# } g$
shows $f = g$
proof-
have *ObjX: X ∈ Obj C* **and** *ObjY: Y ∈ Obj C* **using** *assms* **by** (*simp add:*
Category.MapsToObj)**+**

have $M2Zf: (m2z_C f) \mid \in \mid ((YFtor C \ @\@ Y) \ @\@ X)$ **and** $M2Zg: (m2z_C g) \mid \in \mid ((YFtor C \ @\@ Y) \ @\@ X)$
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** $(m2z_C (Id C A)) \mid \in \mid (Hom_C A A)$ **using** *assms* **by** (*simp add: Category.Cidm LSCategory.m2zz2m*)
ultimately **have** $(m2z_C (Id C A)) \mid \in \mid (Hom_C A B)$ **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.ucas.ac.cn/~apk32/Isabelle/Category/Cat.pdf>.