Category Theory

Alexander Katovsky

March 2, 2013

Abstract

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

Contents

1 Category

theory *Category*
imports ~~/*src*/*HOL*/*Library*/*FuncSet*
begin

record ('*o*,'*m*) *Category* =
  Obj :: '*o set* (*obj*ı *70*)
  Mor :: '*m set* (*mor*ı *70*)
  Dom :: '*m* ⇒ '*o* (*dom*ı - [*80*] *70*)
  Cod :: '*m* ⇒ '*o* (*cod*ı - [*80*] *70*)

Id :: *'o* ⇒ *'m* (*id*ı - [*80*] *75*)
Comp :: *'m* ⇒ *'m* ⇒ *'m* (**infixl** ;;ı *70*)

definition
MapsTo :: (*'o*,*'m*,*'a*) *Category-scheme* ⇒ *'m* ⇒ *'o* ⇒ *'o* ⇒ *bool* (- *maps*ı - *to* - [*60*, *60*, *60*] *65*) **where**
MapsTo CC f X Y ≡ *f* ∈ *Mor CC* ∧ *Dom CC f* = *X* ∧ *Cod CC f* = *Y*

definition
CompDefined :: (*'o*,*'m*,*'a*) *Category-scheme* ⇒ *'m* ⇒ *'m* ⇒ *bool* (**infixl** ≈>ı *65*)
where
CompDefined CC f g ≡ *f* ∈ *Mor CC* ∧ *g* ∈ *Mor CC* ∧ *Cod CC f* = *Dom CC g*

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

locale *Category* = *ExtCategory* +
assumes *Cdom* : *f* ∈ *mor* ⟹ *dom f* ∈ *obj*
and *Ccod* : *f* ∈ *mor* ⟹ *cod f* ∈ *obj*
and *Cidm* [*dest*]: *X* ∈ *obj* ⟹ (*id X*) *maps X to X*
and *Cidl* : *f* ∈ *mor* ⟹ *id* (*dom f*) ;; *f* = *f*
and *Cidr* : *f* ∈ *mor* ⟹ *f* ;; *id* (*cod f*) = *f*
and *Cassoc* : ⟦*f* ≈> *g* ; *g* ≈> *h*⟧ ⟹ (*f* ;; *g*) ;; *h* = *f* ;; (*g* ;; *h*)
and *Ccompt* : ⟦*f maps X to Y* ; *g maps Y to Z*⟧ ⟹ (*f* ;; *g*) *maps X to Z*

definition
MakeCat :: (*'o*,*'m*,*'a*) *Category-scheme* ⇒ (*'o*,*'m*,*'a*) *Category-scheme* **where**
MakeCat C ≡ ⦇
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 = λ *f g* . (*restrict* (*split* (*Comp C*)) ({(*f*,*g*) | *f g* . $f \approx>_C g$})) (*f*,*g*),
... = *Category.more C*
⦈

lemma *MakeCatMapsTo*: $f\ maps_C\ X\ to\ Y \Longrightarrow f\ maps_{MakeCat\ C}\ X\ to\ Y$
⟨*proof*⟩

lemma *MakeCatComp*: $f \approx>_C g \Longrightarrow f\ ;;_{MakeCat\ C}\ g = f\ ;;_C\ g$
⟨*proof*⟩

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

lemma *MakeCatObj*: $obj_{MakeCat\ C} = obj_C$
⟨*proof*⟩

lemma *MakeCatMor*: $mor_{MakeCat\ C} = mor_C$
⟨*proof*⟩

lemma *MakeCatDom*: $f \in mor_C \Longrightarrow dom_C\ f = dom_{MakeCat\ C}\ f$
⟨*proof*⟩

lemma *MakeCatCod*: $f \in mor_C \Longrightarrow cod_C\ f = cod_{MakeCat\ C}\ f$
⟨*proof*⟩

lemma *MakeCatCompDef*: $f \approx>_{MakeCat\ C} g = f \approx>_C g$
⟨*proof*⟩

lemma *MakeCatComp2*: $f \approx>_{MakeCat\ C} g \Longrightarrow f\ ;;_{MakeCat\ C}\ g = f\ ;;_C\ g$
⟨*proof*⟩

lemma *ExtCategoryMakeCat*: *ExtCategory* (*MakeCat C*)
⟨*proof*⟩

lemma *MakeCat*: *Category-axioms C* $\Longrightarrow$ *Category* (*MakeCat C*)
⟨*proof*⟩

lemma *MapsToE*[*elim*]: $[\![f\ maps_C\ X\ to\ Y\ ;\ [\![f \in mor_C\ ;\ dom_C\ f = X\ ;\ cod_C\ f = Y]\!] \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *MapsToI*[*intro*]: $[\![f \in mor_C\ ;\ dom_C\ f = X\ ;\ cod_C\ f = Y]\!] \Longrightarrow f\ maps_C\ X\ to\ Y$
⟨*proof*⟩

lemma *CompDefinedE*[*elim*]: $[\![f \approx>_C g\ ;\ [\![f \in mor_C\ ;\ g \in mor_C\ ;\ cod_C\ f = dom_C\ g]\!] \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *CompDefinedI*[*intro*]: $[\![f \in mor_C\ ;\ g \in mor_C\ ;\ cod_C\ f = dom_C\ g]\!] \Longrightarrow f \approx>_C g$
⟨*proof*⟩

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

lemma *MapsToCompDef*:
assumes $f\ maps_C\ X\ to\ Y$ **and** $g\ maps_C\ Y\ to\ Z$

shows $f \approx>_C g$
$\langle proof \rangle$

lemma (**in** *Category*) *MapsToMorDomCod*:
assumes $f \approx> g$
shows f ;; $g \in mor$ **and** $dom\ (f\ ;;\ g) = dom\ f$ **and** $cod\ (f\ ;;\ g) = cod\ g$
$\langle proof \rangle$

lemma (**in** *Category*) *MapsToObj*:
assumes f *maps* X *to* Y
shows $X \in obj$ **and** $Y \in obj$
$\langle proof \rangle$

lemma (**in** *Category*) *IdInj*:
assumes $X \in obj$ **and** $Y \in obj$ **and** $id\ X = id\ Y$
shows $X = Y$
$\langle proof \rangle$

lemma (**in** *Category*) *CompDefComp*:
assumes $f \approx> g$ **and** $g \approx> h$
shows $f \approx> (g\ ;;\ h)$ **and** $(f\ ;;\ g) \approx> h$
$\langle proof \rangle$

lemma (**in** *Category*) *CatIdInMor*: $X \in obj \Longrightarrow id\ X \in mor$
$\langle proof \rangle$

lemma (**in** *Category*) *MapsToId*: **assumes** $X \in obj$ **shows** $id\ X \approx> id\ X$
$\langle proof \rangle$

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

lemma (**in** *Category*) *LeftRightInvUniq*:
assumes *0*: $h \approx> f$ **and** *z*: $f \approx> g$
assumes *1*: f ;; $g = id\ (dom\ f)$
and *2*: h ;; $f = id\ (cod\ f)$
shows $h = g$
$\langle proof \rangle$

lemma (**in** *Category*) *CatIdDomCod*:
assumes $X \in obj$
shows $dom\ (id\ X) = X$ **and** $cod\ (id\ X) = X$
$\langle proof \rangle$

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

lemma (**in** *Category*) *CatIdUniqR*:
assumes *iota*: ι *maps X to X*
and *rid*: $\forall\ f\ .\ f \approx> \iota \longrightarrow f\ ;;\ \iota = f$
shows $id\ X = \iota$
$\langle proof \rangle$

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

lemma (**in** *Category*) *Inverse-relI*: $[\![f \approx> g\ ;\ f\ ;;\ g = id\ (dom\ f)\ ;\ g\ ;;\ f = id\ (cod\ f)]\!] \Longrightarrow (cinv\ f\ g)$
$\langle proof \rangle$

lemma (**in** *Category*) *Inverse-relE*[*elim*]: $[\![cinv\ f\ g\ ;\ [\![f \approx> g\ ;\ f\ ;;\ g = id\ (dom\ f)\ ;\ g\ ;;\ f = id\ (cod\ f)]\!] \Longrightarrow P]\!] \Longrightarrow P$
$\langle proof \rangle$

lemma (**in** *Category*) *Inverse-relSym*:
assumes *cinv f g*
shows *cinv g f*
$\langle proof \rangle$

lemma (**in** *Category*) *InverseUnique*:
assumes *1*: *cinv f g*
and *2*: *cinv f h*
shows $g = h$
$\langle proof \rangle$

lemma (**in** *Category*) *InvId*: **assumes** $X \in obj$ **shows** $(cinv\ (id\ X)\ (id\ X))$
$\langle proof \rangle$

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

lemma (**in** *Category*) *inv2Inv*:
assumes *cinv f g*
shows *ciso f* **and** $Cinv\ f = g$
$\langle proof \rangle$

lemma (**in** *Category*) *iso2Inv*:
assumes *ciso f*
shows *cinv f* (*Cinv f*)
⟨*proof*⟩

lemma (**in** *Category*) *InvInv*:
assumes *ciso f*
shows *ciso* (*Cinv f*) **and** (*Cinv* (*Cinv f*)) = *f*
⟨*proof*⟩

lemma (**in** *Category*) *InvIsMor*: (*cinv f g*) $\Longrightarrow$ ($f \in mor \wedge g \in mor$)
⟨*proof*⟩

lemma (**in** *Category*) *IsoIsMor*: *ciso f* $\Longrightarrow$ $f \in mor$
⟨*proof*⟩

lemma (**in** *Category*) *InvDomCod*:
assumes *ciso f*
shows *dom* (*Cinv f*) = *cod f* **and** *cod* (*Cinv f*) = *dom f* **and** *Cinv f* $\in$ *mor*
⟨*proof*⟩

lemma (**in** *Category*) *IsoCompInv*: *ciso f* $\Longrightarrow$ *f* ≈> *Cinv f*
⟨*proof*⟩

lemma (**in** *Category*) *InvCompIso*: *ciso f* $\Longrightarrow$ *Cinv f* ≈> *f*
⟨*proof*⟩

lemma (**in** *Category*) *IsoInvId1* : *ciso f* $\Longrightarrow$ (*Cinv f*) ;; *f* = (*id* (*cod f*))
⟨*proof*⟩

lemma (**in** *Category*) *IsoInvId2* : *ciso f* $\Longrightarrow$ *f* ;; (*Cinv f*) = (*id* (*dom f*))
⟨*proof*⟩

lemma (**in** *Category*) *IsoCompDef*:
assumes *1*: *f* ≈> *g* **and** *2*: *ciso f* **and** *3*: *ciso g*
shows (*Cinv g*) ≈> (*Cinv f*)
⟨*proof*⟩

lemma (**in** *Category*) *IsoCompose*:
assumes *1*: *f* ≈> *g* **and** *2*: *ciso f* **and** *3*: *ciso g*
shows *ciso* (*f* ;; *g*) **and** *Cinv* (*f* ;; *g*) = (*Cinv g*) ;; (*Cinv f*)
⟨*proof*⟩

definition *ObjIso C A B* $\equiv \exists\ k$. ($k\ maps_C\ A\ to\ B$) $\wedge$ $ciso_C\ k$

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*)
$\langle proof \rangle$

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

lemma *OpCatOpCat*: $Op\ (Op\ C) = C$
$\langle proof \rangle$

lemma *OpCatCatAx*: *Category-axioms* $C \Longrightarrow$ *Category-axioms* $(Op\ C)$
$\langle proof \rangle$

lemma *OpCatCatExt*: *ExtCategory* $C \Longrightarrow$ *ExtCategory* $(Op\ C)$
$\langle proof \rangle$

lemma *OpCatCat*: *Category* $C \Longrightarrow$ *Category* $(Op\ C)$
$\langle proof \rangle$

lemma *MapsToOp*: $f\ maps_C\ X\ to\ Y \Longrightarrow f\ maps_{Op\ C}\ Y\ to\ X$
$\langle proof \rangle$

lemma *MapsToOpOp*: $f\ maps_{Op\ C}\ X\ to\ Y \Longrightarrow f\ maps_C\ Y\ to\ X$
$\langle proof \rangle$

lemma *CompDefOp*: $f \approx>_C g \Longrightarrow g \approx>_{Op\ C} f$
$\langle proof \rangle$

end

2 Universe

theory *Universe*
imports ~~/*src*/*HOL*/*ZF*/*MainZF*
begin

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

lemma *ElemLambdaFun* : ($\bigwedge$ *x* .*Elem x u* $\Longrightarrow$ *Elem* (*f x*) *U*) $\Longrightarrow$ *Elem* (*Lambda u f*) (*Fun u U*)
⟨*proof*⟩

lemma *RangeRepl*: *Range* (*Lambda A f*) = *Repl A f*
⟨*proof*⟩

lemma (**in** *Universe*) *Utrans*: $[\![$*Elem a U* ; *Elem b a*$]\!]$ $\Longrightarrow$ *Elem b U*
⟨*proof*⟩

lemma *ReplId*: *Repl A id* = *A*
⟨*proof*⟩

lemma (**in** *Universe*) *UniverseSum* : *Elem u U* $\Longrightarrow$ *Elem* (*Sum u*) *U*
⟨*proof*⟩

lemma (**in** *Universe*) *UniverseUnion*:
  assumes *Elem u U* **and** *Elem v U*
  shows *Elem* (*union u v*) *U*
⟨*proof*⟩

lemma *UPairSingleton*: *Upair u v* = *union* (*Singleton u*) (*Singleton v*)
⟨*proof*⟩

lemma (**in** *Universe*) *UniverseUPair*: $[\![$*Elem u U* ; *Elem v U*$]\!]$ $\Longrightarrow$ *Elem* (*Upair u v*) *U*
⟨*proof*⟩

lemma (**in** *Universe*) *UniversePair*: $[\![$*Elem u U* ; *Elem v U*$]\!]$ $\Longrightarrow$ *Elem* (*Opair u v*) *U*
⟨*proof*⟩

lemma (**in** *Universe*) $[\![$*Elem u U* ; *Elem v U*$]\!]$ $\Longrightarrow$ *Elem* (*Sum* (*Repl u* (%*x* .

Singleton (*Opair x v*)))) *U*
⟨*proof*⟩

lemma *SumRepl*: *Sum* (*Repl A* (*Singleton o f*)) = *Repl A f*
⟨*proof*⟩

lemma (**in** *Universe*) *UniverseProd*:
assumes *Elem u U* **and** *Elem v U*
shows *Elem* (*CartProd u v*) *U*
⟨*proof*⟩

lemma (**in** *Universe*) *UniverseSubset*: ⟦*subset u v* ; *Elem v U*⟧ ⟹ *Elem u U*
⟨*proof*⟩

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

lemma *SepSubset*: *subset* (*Sep A p*) *A*
⟨*proof*⟩

lemma *SubsetSmall*:
assumes *subset A′ A* **and** *subset A B* **shows** *subset A′ B*
⟨*proof*⟩

lemma *SubsetTrans*:
assumes (*subset a b*) **and** (*subset b c*)
shows (*subset a c*)
⟨*proof*⟩

lemma *SubsetSepTrans*: *subset A B* ⟹ *subset* (*Sep A p*) *B*
⟨*proof*⟩

lemma *ProductSubset*: *subset* (*Product u*) (*Power* (*CartProd u* (*Sum u*)))
⟨*proof*⟩

lemma (**in** *Universe*) *UniverseProduct*: *Elem u U* ⟹ *Elem* (*Product u*) *U*
⟨*proof*⟩

lemma *ZFImageRangeExplode*: *x* ∈ *range explode* ⟹ *f* ‘ *x* ∈ *range explode*
⟨*proof*⟩

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

lemma *subsetFn*: ⟦ *Y* ≠ {} ; *Y* ⊆ *X* ⟧ ⟹ (*subsetFn X Y*) ‘ *X* = *Y*
⟨*proof*⟩

lemma *ZFSubsetRangeExplode*: $[\![X \in \mathit{range\ explode}\ ;\ Y \subseteq X]\!] \Longrightarrow Y \in \mathit{range}$ $\mathit{explode}$
⟨*proof*⟩

lemma *ZFUnionRangeExplode*:
assumes $\bigwedge\ x\ .\ x \in A \Longrightarrow f\,x \in \mathit{range\ explode}$ **and** $A \in \mathit{range\ explode}$
shows $(\bigcup\ x \in A\ .\ f\,x) \in \mathit{range\ explode}$
⟨*proof*⟩

lemma *ZFUnionNatInRangeExplode*: $(\bigwedge\ (n :: \mathit{nat})\ .\ f\,n \in \mathit{range\ explode}) \Longrightarrow (\bigcup$ $n\ .\ f\,n) \in \mathit{range\ explode}$
⟨*proof*⟩

lemma *ZFProdFnInRangeExplode*: $[\![A \in \mathit{range\ explode}\ ;\ B \in \mathit{range\ explode}]\!] \Longrightarrow$ $f\ \text{`}\ (A \times B) \in \mathit{range\ explode}$
⟨*proof*⟩

lemma *ZFUnionInRangeExplode*: $[\![A \in \mathit{range\ explode}\ ;\ B \in \mathit{range\ explode}]\!] \Longrightarrow A$ $\cup\ B \in \mathit{range\ explode}$
⟨*proof*⟩

lemma *SingletonInRangeExplode*: $\{x\} \in \mathit{range\ explode}$
⟨*proof*⟩

definition *ZFTriple* :: $[\mathit{ZF},\mathit{ZF},\mathit{ZF}] \Rightarrow \mathit{ZF}$ **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*
⟨*proof*⟩
lemma *ZFTSnd*: *ZFTSnd (ZFTriple a b c) = b*
⟨*proof*⟩
lemma *ZFTThd*: *ZFTThd (ZFTriple a b c) = c*
⟨*proof*⟩

lemma *ZFTriple*: $\mathit{ZFTriple}\ a\ b\ c = \mathit{ZFTriple}\ a'\ b'\ c' \Longrightarrow (a = a' \wedge b = b' \wedge c =$ $c')$
⟨*proof*⟩

lemma *ZFSucZero*: *Nat2nat (SucNat Empty) = 1*
⟨*proof*⟩

lemma *ZFZero*: *Nat2nat Empty = 0*
⟨*proof*⟩

lemma *ZFSucNeq0*: $\mathit{Elem}\ x\ \mathit{Nat} \Longrightarrow \mathit{Nat2nat}\ (\mathit{SucNat}\ x) \neq 0$

⟨*proof*⟩

end

3 Monadic Equational Theory

theory *MonadicEquationalTheory*
imports *Category Universe*
begin

record $('t,'f)$ *Signature* =
  BaseTypes :: *'t set* (*Ty*ı)
  BaseFunctions :: *'f set* (*Fn*ı)
  SigDom :: $'f \Rightarrow 't$ (*sDom*ı)
  SigCod :: $'f \Rightarrow 't$ (*sCod*ı)

locale *Signature* =
  fixes S :: $('t,'f)$ *Signature* (**structure**)
  assumes *Domt*: $f \in Fn \Longrightarrow sDom\ f \in Ty$
  and *Codt*: $f \in Fn \Longrightarrow sCod\ f \in Ty$

definition *funsignature-abbrev* (- $\in$ *Sig* - : - $\to$ -) **where**
  $f \in Sig\ S : A \to B \equiv f \in (BaseFunctions\ S) \land A \in (BaseTypes\ S) \land B \in (BaseTypes\ S) \land$
  $(SigDom\ S\ f) = A \land (SigCod\ S\ f) = B \land Signature\ S$

lemma *funsignature-abbrevE*[*elim*]:
$[\![f \in Sig\ S : A \to B\ ;\ [\![f \in (BaseFunctions\ S)\ ;\ A \in (BaseTypes\ S)\ ;\ B \in (BaseTypes\ S)\ ;$
$(SigDom\ S\ f) = A\ ;\ (SigCod\ S\ f) = B\ ;\ Signature\ S]\!] \Longrightarrow R]\!]$
$\Longrightarrow R$
⟨*proof*⟩

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: $A \in BaseTypes\ S \Longrightarrow Sig\ S \triangleright \vdash A\ Type$
  | *WellDefinedVar*: $Sig\ S \triangleright \vdash A\ Type \Longrightarrow Sig\ S \triangleright (Vx : A \vdash Vx : A)$
  | *WellDefinedFn*: $[\![Sig\ S \triangleright (Vx : A \vdash e : B)\ ;\ f \in Sig\ S : B \to C]\!] \Longrightarrow Sig\ S \triangleright (Vx : A \vdash (f\ E@\ e) : C)$
  | *WellDefinedEq*: $[\![Sig\ S \triangleright (Vx : A \vdash e1 : B)\ ;\ Sig\ S \triangleright (Vx : A \vdash e2 : B)]\!] \Longrightarrow Sig\ S \triangleright (Vx : A \vdash e1 \equiv e2 : B)$

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

lemma *SigId*: $Sig\ S \triangleright (Vx : A \ \vdash Vx : B) \Longrightarrow A = B$
⟨*proof*⟩

lemma *SigTyId*: $Sig\ S \triangleright (Vx : A \ \vdash Vx : A) \Longrightarrow A \in BaseTypes\ S$
⟨*proof*⟩

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

datatype $('o,'m)\ IType = IObj\ 'o \mid IMor\ 'm \mid IBool\ bool$

record $('t,'f,'o,'m)\ Interpretation =$
  $ISignature :: ('t,'f)\ Signature\ (iS_1)$
  $ICategory :: ('o,'m)\ Category\ (iC_1)$
  $ITypes :: 't \Rightarrow 'o\ (Ty[\![\text{-}]\!]_1)$
  $IFunctions :: 'f \Rightarrow 'm\ (Fn[\![\text{-}]\!]_1)$

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

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

lemmas *Interp.intros* [*intro*]
inductive-cases *InterpTyE* [*elim*!]: $L[\![\vdash A\ Type]\!]_I \to i$
inductive-cases *InterpVarE* [*elim*!]: $L[\![Vx : A \ \vdash Vx : A]\!]_I \to i$

inductive-cases *InterpFnE* [*elim*!]: $L[\![Vx : A \vdash (f\ E@\ e) : C]\!]_I \to i$
inductive-cases *InterpEqE* [*elim*!]: $L[\![Vx : A \vdash e1 \equiv e2 : B]\!]_I \to i$

lemma (**in** *Interpretation*) *InterpEqEq*[*intro*]:
$[\![L[\![Vx : A \vdash e1 : B]\!] \to (IMor\ g)\ ;\ L[\![Vx : A \vdash e2 : B]\!] \to (IMor\ g)]\!] \Longrightarrow L[\![Vx : A \vdash e1 \equiv e2 : B]\!] \to (IBool\ True)$
⟨*proof*⟩

lemma (**in** *Interpretation*) *InterpExprWellDefined*:
$L[\![Vx : A \vdash e : B]\!] \to i \Longrightarrow Sig\ iS \rhd Vx : A \vdash e : B$
⟨*proof*⟩

lemma (**in** *Interpretation*) *WellDefined*: $L[\![\varphi]\!] \to i \Longrightarrow Sig\ iS \rhd \varphi$
⟨*proof*⟩

lemma (**in** *Interpretation*) *Bool*: $L[\![\varphi]\!] \to (IBool\ i) \Longrightarrow \exists\ A\ B\ e\ d\ .\ \varphi = (Vx : A \vdash e \equiv d : B)$
⟨*proof*⟩

lemma (**in** *Interpretation*) *FunctionalExpr*:
$\bigwedge i\ j\ A\ B.[\![L[\![Vx : A \vdash e : B]\!] \to i;\ L[\![Vx : A \vdash e : B]\!] \to j]\!] \Longrightarrow i = j$
⟨*proof*⟩

lemma (**in** *Interpretation*) *Functional*: $[\![L[\![\varphi]\!] \to i1\ ;\ L[\![\varphi]\!] \to i2]\!] \Longrightarrow i1 = i2$
⟨*proof*⟩

lemma (**in** *Interpretation*) *MorphismsPreserved*:
$\bigwedge\ B\ i\ .\ L[\![Vx : A \vdash e : B]\!] \to i \Longrightarrow \exists\ g\ .\ i = (IMor\ g) \wedge (g\ maps_{iC}\ Ty[\![A]\!]\ to\ Ty[\![B]\!])$
⟨*proof*⟩

lemma (**in** *Interpretation*) *Expr2Mor*: $L[\![Vx : A \vdash e : B]\!] \to (IMor\ g) \Longrightarrow (g\ maps_{iC}\ Ty[\![A]\!]\ to\ Ty[\![B]\!])$
⟨*proof*⟩

lemma (**in** *Interpretation*) *WellDefinedExprInterp*: $\bigwedge\ B\ .\ (Sig\ iS \rhd Vx : A \vdash e : B) \Longrightarrow (\exists\ i\ .\ L[\![Vx : A \vdash e : B]\!] \to i)$
⟨*proof*⟩

lemma (**in** *Interpretation*) *Sig2Mor*: **assumes** $(Sig\ iS \rhd Vx : A \vdash e : B)$ **shows** $\exists\ g\ .\ L[\![Vx : A \vdash e : B]\!] \to (IMor\ g)$
⟨*proof*⟩

record $('t,'f)$ *Axioms* =
  aAxioms :: $('t,'f)$ *Language set*
  aSignature :: $('t,'f)$ *Signature* (aS_1)

locale *Axioms* =
  fixes *Ax* :: $('t,'f)$ *Axioms* (**structure**)

assumes AxT: $(aAxioms\ Ax) \subseteq \{(Vx : A \vdash e1 \equiv e2 : B) \mid A\ B\ e1\ e2\ .\ Sig\ (aSignature\ Ax) \triangleright (Vx : A \vdash e1 \equiv e2 : B)\}$
assumes $AxSig$: $Signature\ (aSignature\ Ax)$

primrec $Subst :: ('t,'f)\ Expression \Rightarrow ('t,'f)\ Expression \Rightarrow ('t,'f)\ Expression\ (sub\ \text{-}\ in\ \text{-}\ [81,81]\ 81)$ **where**
$(sub\ e\ in\ Vx) = e \mid sub\ e\ in\ (f\ E@\ d) = (f\ E@\ (sub\ e\ in\ d))$

lemma $SubstXinE$: $(sub\ Vx\ in\ e) = e$
$\langle proof \rangle$

lemma $SubstAssoc$: $sub\ a\ in\ (sub\ b\ in\ c) = sub\ (sub\ a\ in\ b)\ in\ c$
$\langle proof \rangle$

lemma $SubstWellDefined$: $\bigwedge C\ .\ [\![Sig\ S \triangleright (Vx : A \vdash e : B);\ Sig\ S \triangleright (Vx : B \vdash d : C)]\!]$
$\Longrightarrow Sig\ S \triangleright (Vx : A \vdash (sub\ e\ in\ d) : C)$
$\langle proof \rangle$

inductive-set (**in** $Axioms$) $Theory$ **where**
Ax: $A \in (aAxioms\ Ax) \Longrightarrow A \in Theory$
$\mid Refl$: $Sig\ (aSignature\ Ax) \triangleright (Vx : A \vdash e : B) \Longrightarrow (Vx : A \vdash e \equiv e : B) \in Theory$
$\mid Symm$: $(Vx : A \vdash e1 \equiv e2 : B) \in Theory \Longrightarrow (Vx : A \vdash e2 \equiv e1 : B) \in Theory$

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

lemma (**in** $Axioms$) $Equiv2WellDefined$: $\varphi \in Theory \Longrightarrow Sig\ aS \triangleright \varphi$
$\langle proof \rangle$

lemma (**in** $Axioms$) $Subst'$:
$\bigwedge C\ .\ [\![Sig\ aS \triangleright Vx : B \vdash d : C\ ;\ (Vx : A \vdash e1 \equiv e2 : B) \in Theory]\!] \Longrightarrow$
$(Vx : A \vdash (sub\ e1\ in\ d) \equiv (sub\ e2\ in\ d) : C) \in Theory$
$\langle proof \rangle$

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

lemma (**in** *Interpretation*) *Equiv*:
  assumes $L[\![Vx : A \vdash e1 \equiv e2 : B]\!] \rightarrow (IBool\ True)$
  shows $\exists\ g\ .\ (L[\![Vx : A \vdash e1 : B]\!] \rightarrow (IMor\ g)) \wedge (L[\![Vx : A \vdash e2 : B]\!] \rightarrow (IMor\ g))$
$\langle proof \rangle$

lemma (**in** *Interpretation*) *SubstComp*: $\bigwedge h\ C\ .\ [\![(L[\![Vx : A \vdash e : B]\!] \rightarrow (IMor\ g))\ ;\ (L[\![Vx : B \vdash d : C]\!] \rightarrow (IMor\ h))]\!] \Longrightarrow$
  $(L[\![Vx : A \vdash (sub\ e\ in\ d) : C]\!] \rightarrow (IMor\ (g\ ;;_{iC}\ h)))$
$\langle proof \rangle$

lemma (**in** *Model*) *Sound*: $\varphi \in Theory \Longrightarrow\ L[\![\varphi]\!] \rightarrow (IBool\ True)$
$\langle proof \rangle$

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

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

lemma [*simp*]: *ZFAxioms T* $\Longrightarrow$ *Axioms T* $\langle proof \rangle$

primrec *Expr2ZF* :: (*ZF*,*ZF*) *Expression* $\Rightarrow$ *ZF* **where**
    Expr2ZFVx: *Expr2ZF Vx* = *ZFTriple* (*nat2Nat 0*) (*nat2Nat 0*) *Empty*
  | *Expr2ZFfe*: *Expr2ZF* (*f E@ e*) = *ZFTriple* (*SucNat* (*ZFTFst* (*Expr2ZF e*)))
                (*nat2Nat 1*)
                (*Opair f* (*Expr2ZF e*))

definition *ZF2Expr* :: *ZF* $\Rightarrow$ (*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* $\Longrightarrow$ *e* = *Vx*
$\langle proof \rangle$

lemma *ZFDepthInNat*: *Elem* (*ZFTFst* (*Expr2ZF e*)) *Nat*
$\langle proof \rangle$

lemma *Expr2ZFType1*: *ZFType* (*Expr2ZF e*) = *1* $\Longrightarrow$
  $\exists\ f\ e'\ .\ e = (f\ E@\ e') \wedge (Suc\ (ZFDepth\ (Expr2ZF\ e'))) = (ZFDepth\ (Expr2ZF\ e))$
$\langle proof \rangle$

lemma *Expr2ZFDepth0*: $ZFDepth\ (Expr2ZF\ e) = 0 \Longrightarrow ZFType\ (Expr2ZF\ e) = 0$
⟨*proof*⟩

lemma *Expr2ZFDepthSuc*: $ZFDepth\ (Expr2ZF\ e) = Suc\ n \Longrightarrow ZFType\ (Expr2ZF\ e) = 1$
⟨*proof*⟩

lemma *Expr2Data*: $ZFData\ (Expr2ZF\ (f\ E@\ e)) = Opair\ f\ (Expr2ZF\ e)$
⟨*proof*⟩

lemma *Expr2ZFinj*: *inj Expr2ZF*
⟨*proof*⟩

definition $TermEquivClGen\ T\ A\ e\ B \equiv \{e'\ .\ (Vx : A \vdash e' \equiv e : B) \in Axioms.Theory\ T\}$
definition $TermEquivCl'\ T\ A\ e\ B \equiv (\!|\ TDomain = A\ ,\ TExprSet = TermEquivClGen\ T\ A\ e\ B\ ,\ TCodomain = B |\!)$

definition *m2ZF* :: $(ZF,ZF)\ TermEquivClT \Rightarrow ZF$ **where**
$m2ZF\ t \equiv ZFTriple\ (TDomain\ t)\ (implode\ (Expr2ZF\ `\ (TExprSet\ t)))\ (TCodomain\ t)$

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

lemma *TDomain*: $TDomain\ (TermEquivCl'\ T\ A\ e\ B) = A$ ⟨*proof*⟩
lemma *TCodomain*: $TCodomain\ (TermEquivCl'\ T\ A\ e\ B) = B$ ⟨*proof*⟩

primrec *WellFormedToSet* :: $(ZF,ZF)\ Signature \Rightarrow nat \Rightarrow (ZF,ZF)\ Expression\ set$ **where**
$WFS0$: $WellFormedToSet\ S\ 0 = \{Vx\}$
| $WFSS$: $WellFormedToSet\ S\ (Suc\ n) = (WellFormedToSet\ S\ n) \cup$
$\{f\ E@\ e \mid f\ e\ .\ f \in BaseFunctions\ S \wedge e \in (WellFormedToSet\ S\ n)\}$

lemma *WellFormedToSetInRangeExplode*: $ZFAxioms\ T \Longrightarrow (Expr2ZF\ `\ (WellFormedToSet\ aS_T\ n)) \in range\ explode$
⟨*proof*⟩

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

lemma *TermSetInSet*: $ZFAxioms\ T \Longrightarrow Expr2ZF\ `\ (TermEquivClGen\ T\ A\ e\ B) \in range\ explode$
⟨*proof*⟩

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

$\langle proof \rangle$

lemma *ZF2m*: $ZFAxioms\ T \Longrightarrow ZF2m\ T\ (m2ZF\ (TermEquivCl'\ T\ A\ e\ B)) = (TermEquivCl'\ T\ A\ e\ B)$
$\langle proof \rangle$

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*: $ZFAxioms\ T \Longrightarrow CLDomain\ T\ [A,e,B]_T = A$ $\langle proof \rangle$
lemma *CLCodomain*: $ZFAxioms\ T \Longrightarrow CLCodomain\ T\ [A,e,B]_T = B$ $\langle proof \rangle$

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

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

lemma *CanonicalCompWellDefined*:
assumes *zaxt*: $ZFAxioms\ T$ **and** $Sig\ aS_T \triangleright (Vx : A \vdash d : B)$ **and** $Sig\ aS_T \triangleright (Vx : B \vdash d' : C)$
shows $CanonicalComp\ T\ [A,d,B]_T\ [B,d',C]_T = [A, sub\ d\ in\ d', C]_T$
$\langle proof \rangle$

definition $CanonicalCat'\ T \equiv (\!|$
$Obj = BaseTypes\ (aS_T),$
$Mor = \{[A,e,B]_T \mid A\ e\ B\ .\ Sig\ aS_T \triangleright (Vx : A \vdash e : B)\},$
$Dom = CLDomain\ T,$
$Cod = CLCodomain\ T,$
$Id = (\lambda\ A\ .\ [A,Vx,A]_T),$
$Comp = CanonicalComp\ T$
$|\!)$

definition $CanonicalCat\ T \equiv MakeCat\ (CanonicalCat'\ T)$

lemma *CanonicalCat'MapsTo*:
assumes $f\ maps_{CanonicalCat'\ T}\ X\ to\ Y$ **and** *zx*: $ZFAxioms\ T$

shows $\exists$ *ef* . $f = [X,ef,Y]_T \wedge$ *Sig* (*aSignature* T) $\triangleright$ ($Vx : X \vdash ef : Y$)
⟨*proof*⟩

lemma *CanonicalCatCat'*: *ZFAxioms* $T \Longrightarrow$ *Category-axioms* (*CanonicalCat'* T)
⟨*proof*⟩

lemma *CanonicalCatCat*: *ZFAxioms* $T \Longrightarrow$ *Category* (*CanonicalCat* T)
⟨*proof*⟩

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

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

lemma *CIObj*: *Obj* (*CanonicalCat* T) = *BaseTypes* (*aSignature* T)
⟨*proof*⟩

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

lemma *CIDom*: $[\![$*ZFAxioms* T ; $[A,e,B]_T \in$ *Mor*(*CanonicalCat* T)$]\!] \Longrightarrow$ *Dom* (*CanonicalCat* T) $[A,e,B]_T = A$
⟨*proof*⟩

lemma *CICod*: $[\![$*ZFAxioms* T ; $[A,e,B]_T \in$ *Mor*(*CanonicalCat* T)$]\!] \Longrightarrow$ *Cod* (*CanonicalCat* T) $[A,e,B]_T = B$
⟨*proof*⟩

lemma *CIId*: $[\![A \in$ *BaseTypes* (*aSignature* T)$]\!] \Longrightarrow$ *Id* (*CanonicalCat* T) A = $[A,Vx,A]_T$
⟨*proof*⟩

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 $[A,e,B]_T ;;_{CanonicalCat\ T} [B,d,C]_T = [A,sub\ e\ in\ d,C]_T$
⟨*proof*⟩

lemma [*simp*]: *ZFAxioms* $T \Longrightarrow$ *Category* $iC_{CI\ T}$ ⟨*proof*⟩
lemma [*simp*]: *ZFAxioms* $T \Longrightarrow$ *Signature* $iS_{CI\ T}$ ⟨*proof*⟩

lemma *CIInterpretation*: *ZFAxioms* $T \Longrightarrow$ *Interpretation* (*CI* T)

$\langle proof \rangle$

lemma *CIInterp2Mor*: *ZFAxioms T* $\Longrightarrow$ ($\bigwedge$ *B* . *Sig iS* $_{CI\ T}$ $\triangleright$ (*Vx* : *A* $\vdash$ *e* : *B*) $\Longrightarrow$ $L[\![Vx : A \vdash e : B]\!]_{CI\ T} \rightarrow (IMor\ [A,\ e,\ B]_T))$
$\langle proof \rangle$

lemma *CIModel*: *ZFAxioms T* $\Longrightarrow$ *Model* (*CI T*) *T*
$\langle proof \rangle$

lemma *CIComplete*: **assumes** *ZFAxioms T* **and** $L[\![\varphi]\!]_{CI\ T} \rightarrow (IBool\ True)$ **shows** $\varphi \in$ *Axioms.Theory T*
$\langle proof \rangle$

lemma *Complete*:
assumes *ZFAxioms T*
and $\bigwedge$ (*I* :: (*ZF*,*ZF*,*ZF*,*ZF*) *Interpretation*) . *Model I T* $\Longrightarrow$ $(L[\![\varphi]\!]_I \rightarrow (IBool\ True))$
shows $\varphi \in$ *Axioms.Theory T*
$\langle proof \rangle$

end

4 Functor

theory *Functors*
imports *Category*
begin

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

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

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

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

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

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

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

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

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

lemma *PreFunctorFunctor*[*simp*]: *Functor F* $\Longrightarrow$ *PreFunctor F*
⟨*proof*⟩

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*]: ⟦*Ftor F* : $A \longrightarrow B$; ⟦(*Functor F*) ; (*CatDom F* = A) ; (*CatCod F* = B)⟧ $\Longrightarrow R$⟧ $\Longrightarrow R$
⟨*proof*⟩

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*]: ⟦$F \approx>$;;; G ; ⟦*Functor F* ; *Functor G* ; *CatDom G* = *CatCod F*⟧ $\Longrightarrow R$⟧ $\Longrightarrow R$

⟨proof⟩

lemma (**in** *Functor*) *FunctorMapsTo*:
assumes $f \in mor_{CatDom\ F}$
shows $F\ \#\#\ f\ maps_{CatCod\ F}\ (F\ @@\ (dom_{CatDom\ F}\ f))\ to\ (F\ @@\ (cod_{CatDom\ F}\ f))$
⟨proof⟩

lemma (**in** *Functor*) *FunctorCodDom*:
assumes $f \in mor_{CatDom\ F}$
shows $dom_{CatCod\ F}(F\ \#\#\ f) = F\ @@\ (dom_{CatDom\ F}\ f)$ **and** $cod_{CatCod\ F}(F\ \#\#\ f) = F\ @@\ (cod_{CatDom\ F}\ f)$
⟨proof⟩

lemma (**in** *Functor*) *FunctorCompPreserved*: $f \in mor_{CatDom\ F} \Longrightarrow F\ \#\#\ f \in mor_{CatCod\ F}$
⟨proof⟩

lemma (**in** *Functor*) *FunctorCompDef*:
assumes $f \approx>_{CatDom\ F} g$ **shows** $(F\ \#\#\ f) \approx>_{CatCod\ F} (F\ \#\#\ g)$
⟨proof⟩

lemma *FunctorComp*: $[\![Ftor\ F : A \longrightarrow B\ ;\ f \approx>_A g]\!] \Longrightarrow F\ \#\#\ (f\ ;;_A\ g) = (F\ \#\#\ f)\ ;;_B\ (F\ \#\#\ g)$
⟨proof⟩

lemma *FunctorCompDef*: $[\![Ftor\ F : A \longrightarrow B\ ;\ f \approx>_A g]\!] \Longrightarrow (F\ \#\#\ f) \approx>_B (F\ \#\#\ g)$
⟨proof⟩

lemma *FunctorMapsTo*:
assumes $Ftor\ F : A \longrightarrow B$ **and** $f\ maps_A\ X\ to\ Y$
shows $(F\ \#\#\ f)\ maps_B\ (F\ @@\ X)\ to\ (F\ @@\ Y)$
⟨proof⟩

lemma (**in** *PreFunctor*) *FunctorId2*:
assumes $X \in obj_{CatDom\ F}$
shows $F\ @@\ X \in obj_{CatCod\ F} \land F\ \#\#\ (id_{CatDom\ F}\ X) = id_{CatCod\ F}\ (F\ @@\ X)$
⟨proof⟩

lemma *FunctorId*:
assumes $Ftor\ F : C \longrightarrow D$ **and** $X \in Obj\ C$
shows $F\ \#\#\ (Id\ C\ X) = Id\ D\ (F\ @@\ X)$
⟨proof⟩

lemma (**in** *Functor*) *DomFunctor*: $f \in mor_{CatDom\ F} \Longrightarrow dom_{CatCod\ F}\ (F\ \#\#\ f) = F\ @@\ (dom_{CatDom\ F}\ f)$
⟨proof⟩

lemma (**in** *Functor*) *CodFunctor*: $f \in mor_{CatDom\ F} \Longrightarrow cod_{CatCod\ F}\ (F\ \#\#\ f) = F\ @@\ (cod_{CatDom\ F}\ f)$
⟨*proof*⟩

lemma (**in** *Functor*) *FunctorId3Dom*:
assumes $f \in mor_{CatDom\ F}$
shows $F\ \#\#\ (id_{CatDom\ F}\ (dom_{CatDom\ F}\ f)) = id_{CatCod\ F}\ (dom_{CatCod\ F}\ (F\ \#\#\ f))$
⟨*proof*⟩

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

lemma (**in** *PreFunctor*) *FmToFo*: $[\![X \in obj_{CatDom\ F}\ ;\ Y \in obj_{CatCod\ F}\ ;\ F\ \#\#\ (id_{CatDom\ F}\ X) = id_{CatCod\ F}\ Y]\!] \Longrightarrow F\ @@\ X = Y$
⟨*proof*⟩

lemma *MakeFtorPreFtor*:
assumes *PreFunctor F* **shows** *PreFunctor* (*MakeFtor F*)
⟨*proof*⟩

lemma *MakeFtorMor*: $f \in mor_{CatDom\ F} \Longrightarrow MakeFtor\ F\ \#\#\ f = F\ \#\#\ f$
⟨*proof*⟩

lemma *MakeFtorObj*:
assumes *PreFunctor F* **and** $X \in obj_{CatDom\ F}$
shows $MakeFtor\ F\ @@\ X = F\ @@\ X$
⟨*proof*⟩

lemma *MakeFtor*: **assumes** *FunctorM F* **shows** *Functor* (*MakeFtor F*)
⟨*proof*⟩

definition
IdentityFunctor′ :: ('o,'m,'a) *Category-scheme* ⇒ ('o,'o,'m,'m,'a,'a) *Functor* (*FId′* - [70]) **where**
$IdentityFunctor'\ C \equiv (\!|CatDom = C\ ,\ CatCod = C\ ,\ MapM = (\lambda\ f\ .\ f)\ |\!)$

definition
IdentityFunctor (*FId* - [70]) **where**
$IdentityFunctor\ C \equiv MakeFtor(IdentityFunctor'\ C)$

lemma *IdFtor′PreFunctor*: $Category\ C \Longrightarrow PreFunctor\ (FId'\ C)$
⟨*proof*⟩

lemma *IdFtor′Obj*:

assumes *Category C* **and** $X \in obj_{CatDom\ (FId'\ C)}$
shows $(FId'\ C)$ @@ $X = X$
⟨*proof*⟩

lemma *IdFtor'FtorM*:
assumes *Category C* **shows** *FunctorM* $(FId'\ C)$
⟨*proof*⟩

lemma *IdFtorFtor*: *Category C* $\Longrightarrow$ *Functor* $(FId\ C)$
⟨*proof*⟩

definition
ConstFunctor' :: $('o1,'m1,'a)$ *Category-scheme* $\Rightarrow$
$('o2,'m2,'b)$ *Category-scheme* $\Rightarrow$ $'o2$ $\Rightarrow$ $('o1,'o2,'m1,'m2,'a,'b)$
Functor **where**
ConstFunctor' $A\ B\ b \equiv$ (|
CatDom $= A$,
CatCod $= B$,
MapM $= (\lambda\ f\ .\ (Id\ B)\ b)$
|)

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

lemma *ConstFtor'* :
assumes *Category A Category B* $b \in (Obj\ B)$
shows *PreFunctor* (*ConstFunctor'* $A\ B\ b$)
and *FunctorM* (*ConstFunctor'* $A\ B\ b$)
⟨*proof*⟩

lemma *ConstFtor*:
assumes *Category A Category B* $b \in (Obj\ B)$
shows *Functor* (*ConstFunctor* $A\ B\ b$)
⟨*proof*⟩

definition
UnitFunctor :: $('o,'m,'a)$ *Category-scheme* $\Rightarrow$ $('o,unit,'m,unit,'a,unit)$ *Functor*
where
UnitFunctor C $\equiv$ *ConstFunctor C UnitCategory* ()

lemma *UnitFtor*:
assumes *Category C*
shows *Functor*(*UnitFunctor C*)
⟨*proof*⟩

definition
FunctorComp' :: $('o1,'o2,'m1,'m2,'a1,'a2)$ *Functor* $\Rightarrow$ $('o2,'o3,'m2,'m3,'b1,'b2)$
Functor
$\Rightarrow$ $('o1,'o3,'m1,'m3,'a1,'b2)$ *Functor* (**infixl** ;;: *71*) **where**
FunctorComp' $F\ G \equiv$ (|

$CatDom = CatDom\ F$,
$CatCod = CatCod\ G$,
$MapM\ \ = \lambda\ f\ .\ (MapM\ G)((MapM\ F)\ f)$
⦈

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

lemma *FtorCompComp'*:
assumes $f \approx>_{CatDom\ F}\ g$
and $F \approx>;;;\ G$
shows $G\ \#\#\ (F\ \#\#\ (f\ ;;_{CatDom\ F}\ g)) = (G\ \#\#\ (F\ \#\#\ f))\ ;;_{CatCod\ G}\ (G\ \#\#\ (F\ \#\#\ g))$
⟨*proof*⟩

lemma *FtorCompId*:
assumes a: $X \in (Obj\ (CatDom\ F))$
and $F \approx>;;;\ G$
shows $G\ \#\#\ (F\ \#\#\ (id_{CatDom\ F}\ X)) = id_{CatCod\ G}(G\ @@\ (F\ @@\ X)) \wedge G\ @@\ (F\ @@\ X) \in (Obj\ (CatCod\ G))$
⟨*proof*⟩

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

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*⟩

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*⟩

lemma *FunctorCompMapsTo3*:
assumes $f\ maps_{CatDom\ (F\ ;;;\ G)}\ X\ to\ Y$
and $F \approx>;;;\ G$
and $PreFunctor\ (F\ ;;;\ G)$
shows $F\ ;;;\ G\ \#\#\ f\ maps_{CatCod\ (F\ ;;;\ G)}\ F\ ;;;\ G\ @@\ X\ to\ F\ ;;;\ G\ @@\ Y$

⟨*proof*⟩

lemma *FtorCompPreFtor*:
assumes $F \approx\!>;;; G$
shows *PreFunctor* (F ;;: G)
⟨*proof*⟩

lemma *FtorCompM* :
assumes $F \approx\!>;;; G$
shows *FunctorM* (F ;;: G)
⟨*proof*⟩

lemma *FtorComp*:
assumes $F \approx\!>;;; G$
shows *Functor* (F ;;; G)
⟨*proof*⟩

lemma (**in** *Functor*) *FunctorPreservesIso*:
assumes $ciso_{CatDom\ F}\ k$
shows $ciso_{CatCod\ F}\ (F\ \#\#\ k)$
⟨*proof*⟩

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

lemma *FunctorMFunctor*[*simp*]: *Functor F* $\Longrightarrow$ *FunctorM F*
⟨*proof*⟩

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

end

5 Natural Transformation

theory *NatTrans*
imports *Functors*
begin

record ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *NatTrans* =
NTDom :: ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *Functor*
NTCod :: ('*o1*, '*o2*, '*m1*, '*m2*, '*a*, '*b*) *Functor*
NatTransMap :: '*o1* ⇒ '*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 $NTCatDom\ \eta \equiv CatDom\ (NTDom\ \eta)$
definition $NTCatCod\ \eta \equiv CatCod\ (NTCod\ \eta)$

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

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

locale $NatTrans = NatTransP + NatTransExt$

lemma [*simp*]: $NatTrans\ \eta \Longrightarrow NatTransP\ \eta$
⟨*proof*⟩

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

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

lemma *nt-abbrevE*[*elim*]: $[\![NT\ f : F \Longrightarrow G\ ;\ [\![(NatTrans\ f)\ ;\ (NTDom\ f = F)\ ;\ (NTCod\ f = G)]\!] \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *MakeNT*: $NatTransP\ \eta \Longrightarrow NatTrans\ (MakeNT\ \eta)$
⟨*proof*⟩

lemma *MakeNT-comp*: $X \in Obj\ (NTCatDom\ f) \Longrightarrow (MakeNT\ f)\ \$\$\ X = f\ \$\$\ X$
⟨*proof*⟩

lemma *MakeNT-dom*: $NTCatDom\ f = NTCatDom\ (MakeNT\ f)$
⟨*proof*⟩

lemma *MakeNT-cod*: $NTCatCod\ f = NTCatCod\ (MakeNT\ f)$
⟨*proof*⟩

lemma *MakeNTApp*: $X \in Obj\ (NTCatDom\ (MakeNT\ f)) \Longrightarrow f\ \$\$\ X = (MakeNT\ f)\ \$\$\ X$
⟨*proof*⟩

lemma *NatTransMapsTo*:
assumes $NT\ \eta : F \Longrightarrow G$ **and** $X \in Obj\ (CatDom\ F)$
shows $\eta\ \$\$\ X\ maps_{CatCod\ G}\ (F\ @@\ X)\ to\ (G\ @@\ X)$
⟨*proof*⟩

definition
NTCompDefined :: $('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans$
$\Rightarrow ('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans \Rightarrow bool$ (**infixl** $\approx\!>\!\cdot\ 65$)
where
$NTCompDefined\ \eta 1\ \eta 2 \equiv NatTrans\ \eta 1 \wedge NatTrans\ \eta 2 \wedge NTCatDom\ \eta 2 = NTCatDom\ \eta 1 \wedge$
$NTCatCod\ \eta 2 = NTCatCod\ \eta 1 \wedge NTCod\ \eta 1 = NTDom\ \eta 2$

lemma *NTCompDefinedE*[*elim*]: $[\![\eta 1 \approx\!>\!\cdot\ \eta 2\ ;\ [\![NatTrans\ \eta 1\ ;\ NatTrans\ \eta 2\ ;\ NTCatDom\ \eta 2 = NTCatDom\ \eta 1\ ;$
$NTCatCod\ \eta 2 = NTCatCod\ \eta 1\ ;\ NTCod\ \eta 1 = NTDom\ \eta 2]\!]$
$\Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *NTCompDefinedI*: $[\![NatTrans\ \eta 1\ ;\ NatTrans\ \eta 2\ ;\ NTCatDom\ \eta 2 = NTCatDom\ \eta 1\ ;$
$NTCatCod\ \eta 2 = NTCatCod\ \eta 1\ ;\ NTCod\ \eta 1 = NTDom\ \eta 2]\!]$
$\Longrightarrow \eta 1 \approx\!>\!\cdot\ \eta 2$
⟨*proof*⟩

lemma *NatTransExt0*:
assumes $NTDom\ \eta 1 = NTDom\ \eta 2$ **and** $NTCod\ \eta 1 = NTCod\ \eta 2$
and $\bigwedge X\ .\ X \in Obj\ (NTCatDom\ \eta 1) \Longrightarrow \eta 1\ \$\$\ X = \eta 2\ \$\$\ X$
and $NatTransMap\ \eta 1 \in extensional\ (Obj\ (NTCatDom\ \eta 1))$
and $NatTransMap\ \eta 2 \in extensional\ (Obj\ (NTCatDom\ \eta 2))$
shows $\eta 1 = \eta 2$
⟨*proof*⟩

lemma *NatTransExt'*:

assumes $NTDom\ \eta 1' = NTDom\ \eta 2'$ **and** $NTCod\ \eta 1' = NTCod\ \eta 2'$
and $\bigwedge X\ .\ X \in Obj\ (NTCatDom\ \eta 1') \Longrightarrow \eta 1'\ \$\$\ X = \eta 2'\ \$\$\ X$
shows $MakeNT\ \eta 1' = MakeNT\ \eta 2'$
⟨*proof*⟩

lemma *NatTransExt*:
assumes $NatTrans\ \eta 1$ **and** $NatTrans\ \eta 2$ **and** $NTDom\ \eta 1 = NTDom\ \eta 2$ **and** $NTCod\ \eta 1 = NTCod\ \eta 2$
and $\bigwedge X\ .\ X \in Obj\ (NTCatDom\ \eta 1) \Longrightarrow \eta 1\ \$\$\ X = \eta 2\ \$\$\ X$
shows $\eta 1 = \eta 2$
⟨*proof*⟩

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*: $X \in obj_{CatDom\ F} \Longrightarrow (IdNatTrans\ F)\ \$\$\ X = id_{CatCod\ F}\ (F\ @@\ X)$
⟨*proof*⟩

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

lemma *IdNatTransNatTrans'*: $Functor\ F \Longrightarrow NatTransP(IdNatTrans'\ F)$
⟨*proof*⟩

lemma *IdNatTransNatTrans*: $Functor\ F \Longrightarrow NatTrans\ (IdNatTrans\ F)$
⟨*proof*⟩

definition
$NatTransComp' :: ('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans \Rightarrow$
$('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans \Rightarrow$
$('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans$ (**infixl** $\cdot 1\ 75$) **where**
$NatTransComp'\ \eta 1\ \eta 2 = ($|
$NTDom = NTDom\ \eta 1$,
$NTCod = NTCod\ \eta 2$,
$NatTransMap = \lambda\ X\ .\ (\eta 1\ \$\$\ X)\ ;;_{NTCatCod\ \eta 1}\ (\eta 2\ \$\$\ X)$
|$)$

definition *NatTransComp* (**infixl** $\cdot$ *75*) **where** $\eta 1 \cdot \eta 2 \equiv MakeNT(\eta 1\ \cdot 1\ \eta 2)$

lemma *NatTransComp-Comp1*: $[\![x \in Obj\ (NTCatDom\ f)\ ;\ f \approx\!>\!\cdot\ g]\!] \Longrightarrow (f\ \cdot\ g)$ $\$\$\ x = (f\ \$\$\ x)\ ;;_{NTCatCod\ g}\ (g\ \$\$\ x)$
⟨*proof*⟩

lemma *NatTransComp-Comp2*: $[\![x \in Obj\ (NTCatDom\ f)\ ;\ f \approx\!>\!\cdot\ g]\!] \Longrightarrow (f\ \cdot\ g)$ $\$\$\ x = (f\ \$\$\ x)\ ;;_{NTCatCod\ f}\ (g\ \$\$\ x)$
⟨*proof*⟩

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

NatTransComp-Comp1 NTCatCod-def NTCatDom-def

lemma [*simp*]: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NatTrans\ \eta 1$ ⟨*proof*⟩
lemma [*simp*]: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NatTrans\ \eta 2$ ⟨*proof*⟩
lemma *NTCatDom*: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NTCatDom\ \eta 1 = NTCatDom\ \eta 2$
⟨*proof*⟩
lemma *NTCatCod*: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NTCatCod\ \eta 1 = NTCatCod\ \eta 2$ ⟨*proof*⟩
lemma [*simp*]: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NTCatDom\ (\eta 1\ \cdot 1\ \eta 2) = NTCatDom\ \eta 1$ ⟨*proof*⟩
lemma [*simp*]: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NTCatCod\ (\eta 1\ \cdot 1\ \eta 2) = NTCatCod\ \eta 1$ ⟨*proof*⟩
lemma [*simp*]: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NTCatDom\ (\eta 1\ \cdot\ \eta 2) = NTCatDom\ \eta 1$ ⟨*proof*⟩
lemma [*simp*]: $\eta 1 \approx\!>\!\cdot\ \eta 2 \Longrightarrow NTCatCod\ (\eta 1\ \cdot\ \eta 2) = NTCatCod\ \eta 1$ ⟨*proof*⟩
lemma [*simp*]: $NatTrans\ \eta \Longrightarrow Category(NTCatDom\ \eta)$ ⟨*proof*⟩
lemma [*simp*]: $NatTrans\ \eta \Longrightarrow Category(NTCatCod\ \eta)$ ⟨*proof*⟩
lemma *DDDC*: **assumes** *NatTrans f* **shows** $CatDom\ (NTDom\ f) = CatDom\ (NTCod\ f)$
⟨*proof*⟩
lemma *CCCD*: **assumes** *NatTrans f* **shows** $CatCod\ (NTCod\ f) = CatCod\ (NTDom\ f)$
⟨*proof*⟩

lemma *IdNatTransCompDefDom*: $NatTrans\ f \Longrightarrow (IdNatTrans\ (NTDom\ f)) \approx\!>\!\cdot\ f$
⟨*proof*⟩

lemma *IdNatTransCompDefCod*: $NatTrans\ f \Longrightarrow f \approx\!>\!\cdot\ (IdNatTrans\ (NTCod\ f))$
⟨*proof*⟩

lemma *NatTransCompDefCod*:
assumes *NatTrans* η **and** $f\ maps_{NTCatDom\ \eta}\ X\ to\ Y$
shows $(\eta\ \$\$\ X) \approx\!>_{NTCatCod\ \eta}\ (NTCod\ \eta\ \#\#\ f)$
⟨*proof*⟩

lemma *NatTransCompDefDom*:
assumes *NatTrans* η **and** $f\ maps_{NTCatDom\ \eta}\ X\ to\ Y$
shows $(NTDom\ \eta\ \#\#\ f) \approx\!>_{NTCatCod\ \eta}\ (\eta\ \$\$\ Y)$
⟨*proof*⟩

lemma *NatTransCompCompDef*:
assumes $\eta 1 \approx\!>\!\cdot\ \eta 2$ **and** $X \in obj_{NTCatDom\ \eta 1}$

shows $(\eta 1\ \$\$\ X) \approx>_{NTCatCod\ \eta 1} (\eta 2\ \$\$\ X)$
⟨*proof*⟩

lemma *NatTransCompNatTrans'*:
assumes $\eta 1 \approx>\cdot \eta 2$
shows $NatTransP\ (\eta 1\ \cdot 1\ \eta 2)$
⟨*proof*⟩

lemma *NatTransCompNatTrans*: $\eta 1 \approx>\cdot \eta 2 \Longrightarrow NatTrans\ (\eta 1 \cdot \eta 2)$
⟨*proof*⟩

definition
$CatExp' :: ('o1,'m1,'a)\ Category\text{-}scheme \Rightarrow ('o2,'m2,'b)\ Category\text{-}scheme \Rightarrow$
$(('o1, 'o2, 'm1, 'm2, 'a, 'b)\ Functor,$
$('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans)\ Category$ **where**
$CatExp'\ A\ B \equiv (\!|$
$Category.Obj = \{F\ .\ Ftor\ F : A \longrightarrow B\}$,
$Category.Mor = \{\eta\ .\ NatTrans\ \eta \wedge NTCatDom\ \eta = A \wedge NTCatCod\ \eta = B\}$
,
$Category.Dom = NTDom$,
$Category.Cod = NTCod$,
$Category.Id = IdNatTrans$,
$Category.Comp = \lambda f\ g.\ (f \cdot g)$
$|\!)$

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

lemma *IdNatTransMapL*:
assumes *NT*: *NatTrans f*
shows $IdNatTrans\ (NTDom\ f) \cdot f = f$
⟨*proof*⟩

lemma *IdNatTransMapR*:
assumes *NT*: *NatTrans f*
shows $f \cdot IdNatTrans\ (NTCod\ f) = f$
⟨*proof*⟩

lemma *NatTransCompDefined*:
assumes $f \approx>\cdot g$ **and** $g \approx>\cdot h$
shows $(f \cdot g) \approx>\cdot h$ **and** $f \approx>\cdot (g \cdot h)$
⟨*proof*⟩

lemma *NatTransCompAssoc*:
assumes $f \approx>\cdot g$ **and** $g \approx>\cdot h$
shows $(f \cdot g) \cdot h = f \cdot (g \cdot h)$
⟨*proof*⟩

lemma *CatExpCatAx*:
assumes *Category A* **and** *Category B*

shows *Category-axioms* (*CatExp′ A B*)
⟨*proof*⟩

lemma *CatExpCat*: ⟦*Category A* ; *Category B*⟧ ⟹ *Category* (*CatExp A B*)
⟨*proof*⟩

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

lemma *CatExpDom*: $f \in Mor\ (CatExp\ A\ B) \Longrightarrow dom_{CatExp\ A\ B}\ f = NTDom\ f$
⟨*proof*⟩

lemma *CatExpCod*: $f \in Mor\ (CatExp\ A\ B) \Longrightarrow cod_{CatExp\ A\ B}\ f = NTCod\ f$
⟨*proof*⟩

lemma *CatExpId*: $X \in Obj\ (CatExp\ A\ B) \Longrightarrow Id\ (CatExp\ A\ B)\ X = IdNatTrans\ X$
⟨*proof*⟩

lemma *CatExpNatTransCompDef*: **assumes** $f \approx>_{CatExp\ A\ B}\ g$ **shows** $f \approx>\cdot\ g$
⟨*proof*⟩

lemma *CatExpDist*:
assumes $X \in Obj\ A$ **and** $f \approx>_{CatExp\ A\ B}\ g$
shows $(f\ ;;_{CatExp\ A\ B}\ g)\ \$\$\ X = (f\ \$\$\ X)\ ;;_B\ (g\ \$\$\ X)$
⟨*proof*⟩

lemma *CatExpMorNT*: $f \in Mor\ (CatExp\ A\ B) \Longrightarrow NatTrans\ f$
⟨*proof*⟩

end

6 The Category of Sets

theory *SetCat*
imports *Functors Universe*
begin

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

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

definition
ZFfunDom :: *ZF* ⇒ *ZF* (|*dom*|- [*72*] *72*) **where**

$ZFfunDom\ f \equiv Fst\ (Fst\ f)$

definition
$ZFfunCod :: ZF \Rightarrow ZF\ (|cod|\text{-}\ [72]\ 72)$ **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 $isZFfunE[elim]$: $[\![isZFfun\ f\ ;$
$[\![isOpair\ f\ ;\ isOpair\ (Fst\ f)\ ;\ isFun\ (Snd\ f)\ ;$
$((Snd\ f)\ |\subseteq|\ (Domain\ (Snd\ f))\ |\times|\ (Range\ (Snd\ f)))\ ;$
$(Domain\ (Snd\ f) = |dom|\ f) \wedge (Range\ (Snd\ f)\ |\subseteq|\ |cod|\ f)]\!] \Longrightarrow R]\!] \Longrightarrow R$
$\langle proof \rangle$

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

definition $SET \equiv MakeCat\ SET'$

lemma $ZFfunDom$: $|dom|\ (ZFfun\ A\ B\ f) = A$
$\langle proof \rangle$

lemma $ZFfunCod$: $|cod|\ (ZFfun\ A\ B\ f) = B$
$\langle proof \rangle$

lemma $SETfun$:
assumes $\forall\ x\ .\ x\ |\in|\ A \longrightarrow (f\ x)\ |\in|\ B$
shows $isZFfun\ (ZFfun\ A\ B\ f)$

$\langle proof \rangle$

lemma *ZFCartProd*:
assumes $x \mid\in\mid A \mid\times\mid B$
shows $Fst\ x \mid\in\mid A \wedge Snd\ x \mid\in\mid B \wedge isOpair\ x$
$\langle proof \rangle$

lemma *ZFfunDomainOpair*:
assumes $isFun\ f$
and $x \mid\in\mid Domain\ f$
shows $Opair\ x\ (app\ f\ x) \mid\in\mid f$
$\langle proof \rangle$

lemma *ZFFunToLambda*:
assumes *1*: $isFun\ f$
and *2*: $f \mid\subseteq\mid (Domain\ f) \mid\times\mid (Range\ f)$
shows $f = Lambda\ (Domain\ f)\ (\lambda x.\ app\ f\ x)$
$\langle proof \rangle$

lemma *ZFfunApp*:
assumes $x \mid\in\mid A$
shows $(ZFfun\ A\ B\ f) \mid@\mid x = f\ x$
$\langle proof \rangle$

lemma *ZFfun*:
assumes $isZFfun\ f$
shows $f = ZFfun\ (\ \mid dom\mid f)\ (\ \mid cod\mid f)\ (\lambda x.\ f \mid@\mid x)$
$\langle proof \rangle$

lemma *ZFfun-ext*:
assumes $\forall\ x\ .\ x \mid\in\mid A \longrightarrow f\ x = g\ x$
shows $(ZFfun\ A\ B\ f) = (ZFfun\ A\ B\ g)$
$\langle proof \rangle$

lemma *ZFfunExt*:
assumes $\mid dom\mid\ f = \mid dom\mid\ g$ **and** $\mid cod\mid\ f = \mid cod\mid\ g$ **and** *funf*: $isZFfun\ f$ **and** *fung*: $isZFfun\ g$
and $\bigwedge\ x\ .\ x \mid\in\mid (\ \mid dom\mid\ f) \Longrightarrow f \mid@\mid x = g \mid@\mid x$
shows $f = g$
$\langle proof \rangle$

lemma *ZFfunDomAppCod*:
assumes $isZFfun\ f$
and $x \mid\in\mid \mid dom\mid f$
shows $f \mid@\mid x \mid\in\mid \mid cod\mid f$
$\langle proof \rangle$

lemma *ZFfunComp*:
assumes $\forall\ x\ .\ x \mid\in\mid A \longrightarrow f\ x \mid\in\mid B$

shows $(ZFfun\ A\ B\ f)\ |o|\ (ZFfun\ B\ C\ g) = ZFfun\ A\ C\ (g\ o\ f)$
$\langle proof \rangle$

lemma *ZFfunCompApp*:
assumes $a{:}isZFfun\ f$ **and** $b{:}isZFfun\ g$ **and** $c{:}|dom|g = |cod|f$
shows $f\ |o|\ g = ZFfun\ (\ |dom|\ f)\ (\ |cod|\ g)\ (\lambda\ x\ .\ g\ |@|\ (f\ |@|\ x))$
$\langle proof \rangle$

lemma *ZFfunCompAppZFfun*:
assumes $isZFfun\ f$ **and** $isZFfun\ g$ **and** $|dom|g = |cod|f$
shows $isZFfun\ (f\ |o|\ g)$
$\langle proof \rangle$

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 $f\ |o|\ g\ |o|\ h = f\ |o|\ (g\ |o|\ h)$
$\langle proof \rangle$

lemma *ZFfunCompAppDomCod*:
assumes $isZFfun\ f$ **and** $isZFfun\ g$ **and** $|dom|g = |cod|f$
shows $|dom|\ (f\ |o|\ g) = |dom|\ f \wedge |cod|\ (f\ |o|\ g) = |cod|\ g$
$\langle proof \rangle$

lemma *ZFfunIdLeft*:
assumes $a{:}\ isZFfun\ f$ **shows** $(ZFfun\ (\ |dom|f)\ (\ |dom|f)\ (\lambda x.\ x))\ |o|\ f = f$
$\langle proof \rangle$

lemma *ZFfunIdRight*:
assumes $a{:}\ isZFfun\ f$ **shows** $f\ |o|\ (ZFfun\ (\ |cod|f)\ (\ |cod|f)\ (\lambda x.\ x)) = f$
$\langle proof \rangle$

lemma *SETCategory*: $Category(SET)$
$\langle proof \rangle$

lemma *SETobj*: $X \in Obj\ (SET)$
$\langle proof \rangle$

lemma *SETcod*: $isZFfun\ (ZFfun\ A\ B\ f) \Longrightarrow cod_{SET}\ ZFfun\ A\ B\ f = B$
$\langle proof \rangle$

lemma *SETmor*: $(isZFfun\ f) = (f \in mor_{SET})$
$\langle proof \rangle$

lemma *SETdom*: $isZFfun\ (ZFfun\ A\ B\ f) \Longrightarrow dom_{SET}\ ZFfun\ A\ B\ f = A$
$\langle proof \rangle$

lemma *SETId*: **assumes** $x\ |\in|\ X$ **shows** $(Id\ SET\ X)\ |@|\ x = x$
$\langle proof \rangle$

lemma *SETCompE*[*elim*]: $[\![f \approx>_{SET} g ; [\![isZFfun\ f ; isZFfun\ g ; |cod|\ f = |dom|\ g]\!] \Longrightarrow R]\!] \Longrightarrow R$
⟨*proof*⟩

lemma *SETmapsTo*: $f\ maps_{SET}\ X\ to\ Y \Longrightarrow isZFfun\ f \wedge |dom|\ f = X \wedge |cod|\ f = Y$
⟨*proof*⟩

lemma *SETComp*: **assumes** $f \approx>_{SET} g$ **shows** $f\ ;;_{SET}\ g = f\ |o|\ g$
⟨*proof*⟩

lemma *SETCompAt*:
assumes $f \approx>_{SET} g$ **and** $x\ |\in|\ \ |dom|\ f$ **shows** $(f\ ;;_{SET}\ g)\ |@|\ x = g\ |@|\ (f\ |@|\ x)$
⟨*proof*⟩

lemma *SETZFfun*:
assumes $f\ maps_{SET}\ X\ to\ Y$ **shows** $f = ZFfun\ X\ Y\ (\lambda x\ .\ f\ |@|\ x)$
⟨*proof*⟩

lemma *SETfunDomAppCod*:
assumes $f\ maps_{SET}\ X\ to\ Y$ **and** $x\ |\in|\ X$
shows $f\ |@|\ x\ |\in|\ Y$
⟨*proof*⟩

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

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

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

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

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

lemma [*elim*]: $[\![LSCategory\ C ;$
$[\![Category\ C ; [\![x \in mor_C ; y \in mor_C ; m2z_C\ x = m2z_C\ y]\!] \Longrightarrow x = y;$

$[\![X \in obj_C ; Y \in obj_C]\!] \Longrightarrow HOMCollection\ C\ X\ Y \in range\ explode]\!] \Longrightarrow R]\!]$
$\Longrightarrow R$
$\langle proof \rangle$

definition
$HomFtorMap$ :: ('o,'m,'a) $LSCategory$-$scheme$ ⇒ 'o ⇒ 'm ⇒ ZF (Hom_1[-,-] [65,65] 65) **where**
$HomFtorMap\ C\ X\ g \equiv ZFfun\ (Hom_C\ X\ (dom_C\ g))\ (Hom_C\ X\ (cod_C\ g))\ (\lambda\ f\ .\ m2z_C\ ((z2m_C\ f)\ ;;_C\ g))$

definition
$HomFtor'$ :: ('o,'m,'a) $LSCategory$-$scheme$ ⇒ 'o ⇒
('o,ZF,'m,ZF,(|mor2ZF :: 'm ⇒ ZF, ... :: 'a|),unit) $Functor$ ($HomP_1$[-,-] [65] 65) **where**
$HomFtor'$ C X ≡ (|
$CatDom = C$,
$CatCod = SET$,
$MapM\ = \lambda\ g\ .\ Hom_C[X,g]$
|)

definition $HomFtor$ (Hom_1[-,-] [65] 65) **where** $HomFtor\ C\ X \equiv MakeFtor\ (HomFtor'\ C\ X)$

lemma [$simp$]: $LSCategory\ C \Longrightarrow Category\ C$
$\langle proof \rangle$

lemma (**in** $LSCategory$) $m2zz2m$:
assumes f $maps$ X to Y **shows** ($m2z$ f) |∈| (Hom X Y)
$\langle proof \rangle$

lemma (**in** $LSCategory$) $m2zz2mInv$:
assumes $f \in mor$
shows $z2m\ (m2z\ f) = f$
$\langle proof \rangle$

lemma (**in** $LSCategory$) $z2mm2z$:
assumes $X \in obj$ **and** $Y \in obj$ **and** f |∈| (Hom X Y)
shows $z2m$ f $maps$ X to Y ∧ $m2z\ (z2m\ f) = f$
$\langle proof \rangle$

lemma $HomFtorMapLemma1$:
assumes a: $LSCategory\ C$ **and** b: $X \in obj_C$ **and** c: $f \in mor_C$ **and** d: x |∈| $(Hom_C\ X\ (dom_C\ f))$
shows $(m2z_C\ ((z2m_C\ x)\ ;;_C\ f))$ |∈| $(Hom_C\ X\ (cod_C\ f))$
$\langle proof \rangle$

lemma $HomFtorInMor'$:
assumes $LSCategory\ C$ **and** $X \in obj_C$ **and** $f \in mor_C$
shows $Hom_C[X,f] \in mor_{SET'}$

$\langle proof \rangle$

lemma *HomFtorMor'*:
 assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \in mor_C$
 shows $Hom_C[X,f]$ $maps_{SET'}$ $Hom_C\ X\ (dom_C\ f)$ *to* $Hom_C\ X\ (cod_C\ f)$
$\langle proof \rangle$

lemma *HomFtorMapsTo*:
 $[\![LSCategory\ C\ ;\ X \in obj_C\ ;\ f \in mor_C\ ]\!] \Longrightarrow Hom_C[X,f]\ maps_{SET}\ Hom_C\ X$ $(dom_C\ f)$ *to* $Hom_C\ X\ (cod_C\ f)$
$\langle proof \rangle$

lemma *HomFtorMor*:
 assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \in mor_C$
 shows $Hom_C[X,f] \in Mor\ SET$ **and** $dom_{SET}\ (Hom_C[X,f]) = Hom_C\ X\ (dom_C$ $f)$
 and $cod_{SET}\ (Hom_C[X,f]) = Hom_C\ X\ (cod_C\ f)$
$\langle proof \rangle$

lemma *HomFtorCompDef'*:
 assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \approx>_C g$
 shows $(Hom_C[X,f]) \approx>_{SET'} (Hom_C[X,g])$
$\langle proof \rangle$

lemma *HomFtorDist'*:
 assumes *a*: *LSCategory C* **and** *b*: $X \in obj_C$ **and** *c*: $f \approx>_C g$
 shows $(Hom_C[X,f])\ ;;_{SET'}\ (Hom_C[X,g]) = Hom_C[X,f\ ;;_C\ g]$
$\langle proof \rangle$

lemma *HomFtorDist*:
 assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \approx>_C g$
 shows $(Hom_C[X,f])\ ;;_{SET}\ (Hom_C[X,g]) = Hom_C[X,f\ ;;_C\ g]$
$\langle proof \rangle$

lemma *HomFtorId'*:
 assumes *a*: *LSCategory C* **and** *b*: $X \in obj_C$ **and** *c*: $Y \in obj_C$
 shows $Hom_C[X,id_C\ Y] = id_{SET'}\ (Hom_C\ X\ Y)$
$\langle proof \rangle$

lemma *HomFtorId*:
 assumes *LSCategory C* **and** $X \in obj_C$ **and** $Y \in obj_C$
 shows $Hom_C[X,id_C\ Y] = id_{SET}\ (Hom_C\ X\ Y)$
$\langle proof \rangle$

lemma *HomFtorObj'*:
 assumes *a*: *LSCategory C*
 and *b*: *PreFunctor* $(HomP_C[X,\text{-}])$ **and** *c*: $X \in obj_C$ **and** *d*: $Y \in obj_C$
 shows $(HomP_C[X,\text{-}])$ @@ $Y = Hom_C\ X\ Y$
$\langle proof \rangle$

lemma *HomFtorFtor'*:
assumes a: *LSCategory C*
and b: $X \in obj_C$
shows *FunctorM* $(HomP_C[X,\text{-}])$
⟨*proof*⟩

lemma *HomFtorFtor*:
assumes a: *LSCategory C*
and b: $X \in obj_C$
shows *Functor* $(Hom_C[X,\text{-}])$
⟨*proof*⟩

lemma *HomFtorObj*:
assumes *LSCategory C*
and $X \in obj_C$ **and** $Y \in obj_C$
shows $(Hom_C[X,\text{-}])$ @@ $Y = Hom_C\ X\ Y$
⟨*proof*⟩

definition
HomFtorMapContra :: $('o,'m,'a)$ *LSCategory-scheme* $\Rightarrow 'm \Rightarrow 'o \Rightarrow ZF$ $(HomC_1[\text{-},\text{-}]$ $[65,65]$ $65)$ **where**
HomFtorMapContra $C\ g\ X \equiv ZFfun\ (Hom_C\ (cod_C\ g)\ X)\ (Hom_C\ (dom_C\ g)\ X)$ $(\lambda\ f\ .\ m2z_C\ (g\ ;;_C\ (z2m_C\ f)))$

definition
HomFtorContra' :: $('o,'m,'a)$ *LSCategory-scheme* $\Rightarrow 'o \Rightarrow$
$('o,ZF,'m,ZF,(\!|mor2ZF :: 'm \Rightarrow ZF, \ldots :: 'a|\!),unit)$ *Functor* $(HomP_1[\text{-},\text{-}]$ $[65]$ $65)$ **where**
HomFtorContra' $C\ X \equiv (\!|$
$CatDom = (Op\ C)$,
$CatCod = SET$,
$MapM\ = \lambda\ g\ .\ HomC_C[g,X]$
$|\!)$

definition *HomFtorContra* $(Hom_1[\text{-},\text{-}]$ $[65]$ $65)$ **where** *HomFtorContra* $C\ X \equiv$ $MakeFtor(HomFtorContra'\ C\ X)$

lemma *HomContraAt*: x |∈| $(Hom_C\ (cod_C\ f)\ X) \Longrightarrow (HomC_C[f,X])$ |@| $x =$ $m2z_C\ (f\ ;;_C\ (z2m_C\ x))$
⟨*proof*⟩

lemma *mor2ZF-Op*: $mor2ZF\ (Op\ C) = mor2ZF\ C$
⟨*proof*⟩

lemma *mor-Op*: $mor_{Op\ C} = mor_C$ ⟨*proof*⟩
lemma *obj-Op*: $obj_{Op\ C} = obj_C$ ⟨*proof*⟩

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

⟨*proof*⟩

lemma *mapsTo-Op*: $f\ maps_{Op\ C}\ Y\ to\ X = f\ maps_{C}\ X\ to\ Y$
⟨*proof*⟩

lemma *HOMCollection-Op*: $HOMCollection\ (Op\ C)\ X\ Y = HOMCollection\ C\ Y\ X$
⟨*proof*⟩

lemma *Hom-Op*: $Hom_{Op\ C}\ X\ Y = Hom_{C}\ Y\ X$
⟨*proof*⟩

lemma *HomFtorContra'*: $HomP_{C}[\text{-},X] = HomP_{Op\ C}[X,\text{-}]$
⟨*proof*⟩

lemma *HomFtorContra*: $Hom_{C}[\text{-},X] = Hom_{Op\ C}[X,\text{-}]$
⟨*proof*⟩

lemma *HomFtorContraDom*: $CatDom\ (Hom_{C}[\text{-},X]) = Op\ C$
⟨*proof*⟩

lemma *HomFtorContraCod*: $CatCod\ (Hom_{C}[\text{-},X]) = SET$
⟨*proof*⟩

lemma *LSCategory-Op*: **assumes** *LSCategory C* **shows** *LSCategory* (*Op C*)
⟨*proof*⟩

lemma *HomFtorContraFtor*:
  assumes *LSCategory C*
  and $X \in obj_{C}$
  shows $Ftor\ (Hom_{C}[\text{-},X]) : (Op\ C) \longrightarrow SET$
⟨*proof*⟩

lemma *HomFtorOpObj*:
  assumes *LSCategory C*
  and $X \in obj_{C}$ **and** $Y \in obj_{C}$
  shows $(Hom_{C}[\text{-},X])\ @@\ Y = Hom_{C}\ Y\ X$
⟨*proof*⟩

lemma *HomCHomOp*: $HomC_{C}[g,X] = Hom_{Op\ C}[X,g]$
⟨*proof*⟩

lemma *HomFtorContraMapsTo*:
  assumes *LSCategory C* **and** $X \in obj_{C}$ **and** $f \in mor_{C}$
  shows $HomC_{C}[f,X]\ maps_{SET}\ Hom_{C}\ (cod_{C}\ f)\ X\ \ to\ Hom_{C}\ (dom_{C}\ f)\ X$
⟨*proof*⟩

lemma *HomFtorContraMor*:

assumes *LSCategory C* **and** $X \in obj_C$ **and** $f \in mor_C$
shows $HomC_C[f,X] \in Mor\ SET$ **and** $dom_{SET}\ (HomC_C[f,X]) = Hom_C\ (cod_C\ f)\ X$
and $cod_{SET}\ (HomC_C[f,X]) = Hom_C\ (dom_C\ f)\ X$
⟨*proof*⟩

lemma *HomContraMor*:
assumes *LSCategory C* **and** $f \in Mor\ C$
shows $(Hom_C[\text{-},X])\ \#\#\ f = HomC_C[f,X]$
⟨*proof*⟩

lemma *HomCHom*:
assumes *LSCategory C* **and** $f \in Mor\ C$ **and** $g \in Mor\ C$
shows $(HomC_C[g,dom_C\ f])\ ;;_{SET}\ (Hom_C[dom_C\ g,f]) = (Hom_C[cod_C\ g,f])\ ;;_{SET}\ (HomC_C[g,cod_C\ f])$
⟨*proof*⟩

end

7 Yoneda

theory *Yoneda*
imports *NatTrans SetCat*
begin

definition $YFtorNT'\ C\ f \equiv (\!|NTDom = Hom_C[\text{-},dom_C\ f]\ ,\ NTCod = Hom_C[\text{-},cod_C\ f]\ ,$
$NatTransMap = \lambda\ B\ .\ Hom_C[B,f]|\!)$

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

lemmas $YFtorNT\text{-}defs = YFtorNT'\text{-}def\ YFtorNT\text{-}def\ MakeNT\text{-}def$

lemma *YFtorNTCatDom*: $NTCatDom\ (YFtorNT\ C\ f) = Op\ C$
⟨*proof*⟩

lemma *YFtorNTCatCod*: $NTCatCod\ (YFtorNT\ C\ f) = SET$
⟨*proof*⟩

lemma *YFtorNTApp1*: **assumes** $X \in Obj\ (NTCatDom\ (YFtorNT\ C\ f))$ **shows** $(YFtorNT\ C\ f)\ \$\$\ X = Hom_C[X,f]$
⟨*proof*⟩

definition
$YFtor'\ C \equiv (\!|$

$CatDom = C$,
$CatCod = CatExp\ (Op\ C)\ SET$,
$MapM = \lambda\ f\ .\ YFtorNT\ C\ f$
⦈

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

lemmas $YFtor\text{-}defs = YFtor'\text{-}def\ YFtor\text{-}def\ MakeFtor\text{-}def$

lemma $YFtorNTNatTrans'$:
assumes $LSCategory\ C$ **and** $f \in Mor\ C$
shows $NatTransP\ (YFtorNT'\ C\ f)$
$\langle proof \rangle$

lemma $YFtorNTNatTrans$:
assumes $LSCategory\ C$ **and** $f \in Mor\ C$
shows $NatTrans\ (YFtorNT\ C\ f)$
$\langle proof \rangle$

lemma $YFtorNTMor$:
assumes $LSCategory\ C$ **and** $f \in Mor\ C$
shows $YFtorNT\ C\ f \in Mor\ (CatExp\ (Op\ C)\ SET)$
$\langle proof \rangle$

lemma $YFtorNtMapsTo$:
assumes $LSCategory\ C$ **and** $f \in Mor\ C$
shows $YFtorNT\ C\ f\ maps_{CatExp\ (Op\ C)\ SET}\ (Hom_C[\text{-},dom_C\ f])\ to\ (Hom_C[\text{-},cod_C\ f])$
$\langle proof \rangle$

lemma $YFtorNTCompDef$:
assumes $LSCategory\ C$ **and** $f \approx>_C g$
shows $YFtorNT\ C\ f \approx>_{CatExp\ (Op\ C)\ SET}\ YFtorNT\ C\ g$
$\langle proof \rangle$

lemma $PreSheafCat$: $LSCategory\ C \Longrightarrow Category\ (CatExp\ (Op\ C)\ SET)$
$\langle proof \rangle$

lemma $YFtor'Obj1$:
assumes $X \in Obj\ (CatDom\ (YFtor'\ C))$ **and** $LSCategory\ C$
shows $(YFtor'\ C)\ \#\#\ (Id\ (CatDom\ (YFtor'\ C))\ X) = Id\ (CatCod\ (YFtor'\ C))\ (Hom_C\ [\text{-},X])$
$\langle proof \rangle$

lemma $YFtorPreFtor$:
assumes $LSCategory\ C$
shows $PreFunctor\ (YFtor'\ C)$
$\langle proof \rangle$

lemma *YFtor′Obj*:
  assumes $X \in Obj\ (CatDom\ (YFtor'\ C))$
  and $LSCategory\ C$
  shows $(YFtor'\ C)\ @@\ X = Hom_C\ [\text{-},X]$
⟨*proof*⟩

lemma *YFtorFtor′*:
  assumes $LSCategory\ C$
  shows $FunctorM\ (YFtor'\ C)$
⟨*proof*⟩

lemma *YFtorFtor*: **assumes** $LSCategory\ C$ **shows** $Ftor\ (YFtor\ C) : C \longrightarrow (CatExp\ (Op\ C)\ SET)$
⟨*proof*⟩

lemma *YFtorObj*:
  assumes $LSCategory\ C$ **and** $X \in Obj\ C$
  shows $(YFtor\ C)\ @@\ X = Hom_C\ [\text{-},X]$
⟨*proof*⟩

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*⟩

lemma *YFtorMor*: $[\![LSCategory\ C\ ;\ f \in Mor\ C]\!] \Longrightarrow (YFtor\ C)\ \#\#\ f = YFtorNT\ C\ f$
⟨*proof*⟩

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

lemma *YMapInvApp*:
  assumes $X \in Obj\ C$ **and** $B \in Obj\ C$ **and** $LSCategory\ C$
  shows $(YMapInv\ C\ X\ F\ x)\ \$\$\ B = ZFfun\ (Hom_C\ B\ X)\ (F\ @@\ B)\ (\lambda\ f\ .\ (F\ \#\#\ (z2m_C\ f))\ |@|\ x)$
⟨*proof*⟩

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 $(YMap\ C\ X\ \eta)$ |∈| $(F\ @@\ X)$
⟨*proof*⟩

lemma *YMapInvNatTransP*:
assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** *xobj*: $X \in Obj\ C$ **and** *xinF*: x |∈| $(F\ @@\ X)$
shows *NatTransP* $(YMapInv'\ C\ X\ F\ x)$
⟨*proof*⟩

lemma *YMapInvNatTrans*:
assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$ **and** x |∈| $(F\ @@\ X)$
shows *NatTrans* $(YMapInv\ C\ X\ F\ x)$
⟨*proof*⟩

lemma *YMapInvImage*:
assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$
and x |∈| $(F\ @@\ X)$
shows $NT\ (YMapInv\ C\ X\ F\ x) : (YFtor\ C\ @@\ X) \Longrightarrow F$
⟨*proof*⟩

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

lemma *YMap2*:
assumes *LSCategory C* **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$
and x |∈| $(F\ @@\ X)$
shows $YMap\ C\ X\ (YMapInv\ C\ X\ F\ x) = x$
⟨*proof*⟩

lemma *YFtorNT-YMapInv*:
assumes *LSCategory C* **and** $f\ maps_C\ X\ to\ Y$
shows $YFtorNT\ C\ f = YMapInv\ C\ X\ (Hom_C[\text{-},Y])\ (m2z_C\ f)$
⟨*proof*⟩

lemma *YMapYoneda*:
assumes *LSCategory C* **and** $f\ maps_C\ X\ to\ Y$
shows $YFtor\ C\ \#\#\ f = YMapInv\ C\ X\ (YFtor\ C\ @@\ Y)\ (m2z_C\ f)$
⟨*proof*⟩

lemma *YonedaFull*:
assumes *LSCategory C* **and** $X \in Obj\ C$ **and** $Y \in Obj\ C$
and $NT\ \eta : (YFtor\ C\ @@\ X) \Longrightarrow (YFtor\ C\ @@\ Y)$
shows $YFtor\ C\ \#\#\ (z2m_C\ (YMap\ C\ X\ \eta)) = \eta$
and $z2m_C\ (YMap\ C\ X\ \eta)\ maps_C\ X\ to\ Y$

$\langle proof \rangle$

lemma *YonedaFaithful*:
assumes *LSCategory C* **and** f $maps_C$ X *to* Y **and** g $maps_C$ X *to* Y
and *YFtor C* ## f = *YFtor C* ## g
shows $f = g$
$\langle proof \rangle$

lemma *YonedaEmbedding*:
assumes *LSCategory C* **and** $A \in Obj\ C$ **and** $B \in Obj\ C$ **and** (*YFtor C*) @@ A = (*YFtor C*) @@ B
shows $A = B$
$\langle proof \rangle$

end

References

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