

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	8
3	Monadic Equational Theory	11
4	Functor	19
5	Natural Transformation	25
6	The Category of Sets	31
7	Yoneda	40

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* : $f \in mor \implies dom\ f \in obj$
and *Ccod* : $f \in mor \implies cod\ f \in obj$
and *Cidm* [*dest*]: $X \in obj \implies (id\ X)$ *maps X to X*
and *Cidl* : $f \in mor \implies id\ (dom\ f) ;; f = f$
and *Cidr* : $f \in mor \implies f ;; id\ (cod\ f) = f$
and *Cassoc* : $\llbracket f \approx_{>}\ g ; g \approx_{>}\ h \rrbracket \implies (f ;; g) ;; h = f ;; (g ;; h)$
and *Ccompt* : $\llbracket f\ maps\ X\ to\ Y ; g\ maps\ Y\ to\ Z \rrbracket \implies (f ;; g)\ maps\ X\ to\ Z$

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*: $f\ maps_C\ X\ to\ Y \implies f\ maps_{MakeCat\ C}\ X\ to\ Y$
 $\langle proof \rangle$

lemma *MakeCatComp*: $f \approx_{>C} g \implies f ;;_{MakeCat\ C} g = f ;;_C g$
 $\langle proof \rangle$

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

<proof>

lemma *MakeCatObj*: $obj_{MakeCat\ C} = obj_C$
<proof>

lemma *MakeCatMor*: $mor_{MakeCat\ C} = mor_C$
<proof>

lemma *MakeCatDom*: $f \in mor_C \implies dom_C f = dom_{MakeCat\ C} f$
<proof>

lemma *MakeCatCod*: $f \in mor_C \implies cod_C f = cod_{MakeCat\ C} f$
<proof>

lemma *MakeCatCompDef*: $f \approx_{> MakeCat\ C} g = f \approx_{> C} g$
<proof>

lemma *MakeCatComp2*: $f \approx_{> MakeCat\ C} g \implies f ;;_{MakeCat\ C} g = f ;;_C g$
<proof>

lemma *ExtCategoryMakeCat*: *ExtCategory* (*MakeCat C*)
<proof>

lemma *MakeCat*: *Category-axioms C* \implies *Category* (*MakeCat C*)
<proof>

lemma *MapsToE[elim]*: $\llbracket f \text{ maps}_C X \text{ to } Y ; \llbracket f \in mor_C ; dom_C f = X ; cod_C f = Y \rrbracket \implies R \rrbracket \implies R$
<proof>

lemma *MapsToI[intro]*: $\llbracket f \in mor_C ; dom_C f = X ; cod_C f = Y \rrbracket \implies f \text{ maps}_C X \text{ to } Y$
<proof>

lemma *CompDefinedE[elim]*: $\llbracket f \approx_{> C} g ; \llbracket f \in mor_C ; g \in mor_C ; cod_C f = dom_C g \rrbracket \implies R \rrbracket \implies R$
<proof>

lemma *CompDefinedI[intro]*: $\llbracket f \in mor_C ; g \in mor_C ; cod_C f = dom_C g \rrbracket \implies f \approx_{> C} g$
<proof>

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

lemma *MapsToCompDef*:

assumes $f \text{ maps}_C X \text{ to } Y$ **and** $g \text{ maps}_C Y \text{ to } Z$
shows $f \approx_C g$
 <proof>

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>

lemma (in Category) MapsToObj:
assumes $f \text{ maps } X \text{ to } Y$
shows $X \in \text{obj}$ **and** $Y \in \text{obj}$
 <proof>

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>

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

lemma (in Category) CatIdInMor: $X \in \text{obj} \implies \text{id } X \in \text{mor}$
 <proof>

lemma (in Category) MapsToId: **assumes $X \in \text{obj}$ **shows** $\text{id } X \approx \text{id } X$**
 <proof>

lemmas (in Category) $\text{Simps} = \text{Cdom } \text{Ccod } \text{Cidm } \text{Cidl } \text{Cidr } \text{MapsToCompI } \text{IdInj}$
 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>

lemma (in Category) CatIdDomCod:
assumes $X \in \text{obj}$
shows $\text{dom } (\text{id } X) = X$ **and** $\text{cod } (\text{id } X) = X$
 <proof>

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

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

definition
inverse-rel :: $(\iota, 'm, 'a)$ *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 :: $(\iota, 'm, 'a)$ *Category-scheme* $\Rightarrow 'm \Rightarrow bool$ (*ciso*₁ - [70]) **where**
isomorphism $C\ f \equiv \exists g . inverse-rel\ C\ f\ g$

lemma (in *Category*) *Inverse-relI*: $\llbracket f \approx_{> g ; f ; ; g = id (dom f) ; g ; ; f = id (cod f) \rrbracket \Longrightarrow (cinv\ f\ g)$
 $\langle proof \rangle$

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

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 :: $(\iota, 'm, 'a)$ *Category-scheme* $\Rightarrow 'm \Rightarrow 'm$ (*Cinv*₁ - [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 $\text{cinv } f \text{ (Cinv } f)$

<proof>

lemma (in *Category*) *InvInv*:

assumes *ciso f*

shows $\text{ciso (Cinv } f) \text{ and (Cinv (Cinv } f)) = f$

<proof>

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

<proof>

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

<proof>

lemma (in *Category*) *InvDomCod*:

assumes *ciso f*

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

<proof>

lemma (in *Category*) *IsoCompDef*:

assumes *1: f ≈ > g and 2: ciso f and 3: ciso g*

shows $(\text{Cinv } g) \approx > (\text{Cinv } f)$

<proof>

lemma (in *Category*) *IsoCompose*:

assumes *1: f ≈ > g and 2: ciso f and 3: ciso g*

shows $\text{ciso (} f ;; g) \text{ and Cinv (} f ;; g) = (\text{Cinv } g) ;; (\text{Cinv } f)$

<proof>

definition *ObjIso* $C \ A \ B \equiv \exists k . (k \text{ maps}_C \ A \text{ to } B) \wedge \text{ciso}_C \ k$

definition

UnitCategory :: $(\text{unit}, \text{unit})$ *Category* **where**

UnitCategory = *MakeCat* {

Obj = $\{()\}$,

$$\begin{aligned}
Mor &= \{()\}, \\
Dom &= (\lambda f.()), \\
Cod &= (\lambda f.()), \\
Id &= (\lambda f.()), \\
Comp &= (\lambda f g. ())
\end{aligned}$$

)

lemma [simp]: *Category*(*UnitCategory*)
⟨proof⟩

definition

OppositeCategory :: ('o,'m,'a) *Category-scheme* \Rightarrow ('o,'m,'a) *Category-scheme*
(*Op* - [65] 65) **where**

$$\begin{aligned}
\textit{OppositeCategory } C &\equiv () \\
Obj &= Obj C, \\
Mor &= Mor C, \\
Dom &= Cod C, \\
Cod &= Dom C, \\
Id &= Id C, \\
Comp &= (\lambda f g. g ;;_C f), \\
\dots &= \textit{Category.more } C
\end{aligned}$$

)

lemma *OpCatOpCat*: *Op* (*Op* *C*) = *C*
⟨proof⟩

lemma *OpCatCatAx*: *Category-axioms* *C* \Longrightarrow *Category-axioms* (*Op* *C*)
⟨proof⟩

lemma *OpCatCatExt*: *ExtCategory* *C* \Longrightarrow *ExtCategory* (*Op* *C*)
⟨proof⟩

lemma *OpCatCat*: *Category* *C* \Longrightarrow *Category* (*Op* *C*)
⟨proof⟩

lemma *MapsToOp*: *f* *maps*_{*C*} *X* to *Y* \Longrightarrow *f* *maps*_{*Op C*} *Y* to *X*
⟨proof⟩

lemma *MapsToOpOp*: *f* *maps*_{*Op C*} *X* to *Y* \Longrightarrow *f* *maps*_{*C*} *Y* to *X*
⟨proof⟩

lemma *CompDefOp*: *f* $\approx_{>_C}$ *g* \Longrightarrow *g* $\approx_{>_{Op C}}$ *f*
⟨proof⟩

end

2 Universe

```
theory Universe
imports HOL-ZF.MainZF
begin
```

```
locale Universe =
```

```
  fixes U :: ZF (structure)
  assumes Uempty : Elem Empty U
  and Usubset : Elem u U  $\implies$  subset u U
  and Usingle : Elem u U  $\implies$  Elem (Singleton u) U
  and Upow : Elem u U  $\implies$  Elem (Power u) U
  and Uim :  $\llbracket$ Elem I U ; Elem u (Fun I U)  $\rrbracket \implies$  Elem (Sum (Range u)) U
  and Unat : Elem Nat U
```

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

```
lemma RangeRepl: Range (Lambda A f) = Repl A f
   $\langle$ proof $\rangle$ 
```

```
lemma (in Universe) Utrans:  $\llbracket$ Elem a U ; Elem b a $\rrbracket \implies \text{Elem } b \ U$ 
   $\langle$ proof $\rangle$ 
```

```
lemma ReplId: Repl A id = A
   $\langle$ proof $\rangle$ 
```

```
lemma (in Universe) UniverseSum : Elem u U  $\implies$  Elem (Sum u) U
   $\langle$ proof $\rangle$ 
```

```
lemma (in Universe) UniverseUnion:
  assumes Elem u U and Elem v U
  shows Elem (union u v) U
   $\langle$ proof $\rangle$ 
```

```
lemma UPairSingleton: Upair u v = union (Singleton u) (Singleton v)
   $\langle$ proof $\rangle$ 
```

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

```
lemma (in Universe) UniversePair:  $\llbracket$ Elem u U ; Elem v U $\rrbracket \implies \text{Elem } (\text{Opair } u \ v) \ U$ 
   $\langle$ proof $\rangle$ 
```

```
lemma (in Universe)  $\llbracket$ Elem u U ; Elem v U $\rrbracket \implies \text{Elem } (\text{Sum } (\text{Repl } u \ (\%x \ .$ 
```


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] \implies Elem u U*
<proof>

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*
<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 \implies subset (Sep A p) B*
<proof>

lemma *ProductSubset: subset (Product u) (Power (CartProd u (Sum u)))*
<proof>

lemma (in *Universe*) *UniverseProduct: Elem u U \implies Elem (Product u) U*
<proof>

lemma *ZFImageRangeExplode: x \in range explode \implies f ' x \in range explode*
<proof>

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

lemma *subsetFn: [Y \neq {} ; Y \subseteq X] \implies (subsetFn X Y) ' X = Y*
<proof>

lemma *ZFSubsetRangeExplode*: $\llbracket X \in \text{range explode} ; Y \subseteq X \rrbracket \implies Y \in \text{range explode}$
 <proof>

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>

lemma *ZFUnionNatInRangeExplode*: $(\bigwedge (n :: \text{nat}) . f n \in \text{range explode}) \implies (\bigcup n . f n) \in \text{range explode}$
 <proof>

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

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

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

definition *ZFTriple* :: $[ZF, ZF, ZF] \Rightarrow ZF$ where

$ZFTriple a b c = \text{Opair} (\text{Opair} a b) c$

definition *ZFTFst* = $\text{Fst} \circ \text{Fst}$

definition *ZFTSnd* = $\text{Snd} \circ \text{Fst}$

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*: $ZFTriple a b c = ZFTriple a' b' c' \implies (a = a' \wedge b = b' \wedge c = c')$
 <proof>

lemma *ZFSucZero*: $\text{Nat2nat} (\text{SucNat Empty}) = 1$
 <proof>

lemma *ZFZero*: $\text{Nat2nat Empty} = 0$
 <proof>

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

<proof>

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*: $f \in \text{Fn} \implies \text{sDom } f \in \text{Ty}$
 and *Codt*: $f \in \text{Fn} \implies \text{sCod } f \in \text{Ty}$

definition *funsignature-abbrev* ($- \in \text{Sig} - : - \rightarrow -$) **where**
 $f \in \text{Sig } S : A \rightarrow B \equiv f \in (\text{BaseFunctions } S) \wedge A \in (\text{BaseTypes } S) \wedge B \in$
 $(\text{BaseTypes } S) \wedge$
 $(\text{SigDom } S f) = A \wedge (\text{SigCod } S f) = B \wedge \text{Signature } S$

lemma *funsignature-abbrevE[elim]*:
 $\llbracket f \in \text{Sig } S : A \rightarrow B ; \llbracket f \in (\text{BaseFunctions } S) ; A \in (\text{BaseTypes } S) ; B \in (\text{BaseTypes } S) ;$
 $(\text{SigDom } S f) = A ; (\text{SigCod } S f) = B ; \text{Signature } S \rrbracket \implies R \rrbracket$
 $\implies R$
 <proof>

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

inductive

WellDefined :: (*t,f*) *Signature* \Rightarrow (*t,f*) *Language* \Rightarrow *bool* (*Sig* \triangleright $-$) **where**
 WellDefinedTy: $A \in \text{BaseTypes } S \implies \text{Sig } S \triangleright \vdash A \text{ Type}$
 | *WellDefinedVar*: $\text{Sig } S \triangleright \vdash A \text{ Type} \implies \text{Sig } S \triangleright (Vx : A \vdash Vx : A)$
 | *WellDefinedFn*: $\llbracket \text{Sig } S \triangleright (Vx : A \vdash e : B) ; f \in \text{Sig } S : B \rightarrow C \rrbracket \implies \text{Sig } S \triangleright$
 $(Vx : A \vdash (f E@ e) : C)$
 | *WellDefinedEq*: $\llbracket \text{Sig } S \triangleright (Vx : A \vdash e1 : B) ; \text{Sig } S \triangleright (Vx : A \vdash e2 : B) \rrbracket \implies$
 $\text{Sig } S \triangleright (Vx : A \vdash e1 \equiv e2 : B)$

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

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

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

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* : $A \in BaseTypes\ iS \implies Ty[A] \in Obj\ iC$
and *If* : $(f \in Sig\ iS : A \rightarrow B) \implies Fn[f]\ maps_{iC}\ Ty[A]\ to\ Ty[B]$

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

lemmas *Interp.intros* [intro]
inductive-cases *InterpTyE* [elim!]: $L[\vdash A\ Type]_I \rightarrow i$
inductive-cases *InterpVarE* [elim!]: $L[Vx : A \vdash Vx : A]_I \rightarrow i$
inductive-cases *InterpFnE* [elim!]: $L[Vx : A \vdash (f\ E@ e) : C]_I \rightarrow i$

inductive-cases *InterpEqE* [elim!]: $L[Vx : A \vdash e1 \equiv e2 : B]_I \rightarrow i$

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

$\llbracket L[Vx : A \vdash e1 : B] \rightarrow (IMor\ g) ; L[Vx : A \vdash e2 : B] \rightarrow (IMor\ g) \rrbracket \Longrightarrow L[Vx : A \vdash e1 \equiv e2 : B] \rightarrow (IBool\ True)$
 ⟨proof⟩

lemma (in *Interpretation*) *InterpExprWellDefined*:

$L[Vx : A \vdash e : B] \rightarrow i \Longrightarrow Sig\ iS \triangleright Vx : A \vdash e : B$
 ⟨proof⟩

lemma (in *Interpretation*) *WellDefined*: $L[\varphi] \rightarrow i \Longrightarrow Sig\ iS \triangleright \varphi$

⟨proof⟩

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

⟨proof⟩

lemma (in *Interpretation*) *FunctionalExpr*:

$\bigwedge i\ j\ A\ B . \llbracket L[Vx : A \vdash e : B] \rightarrow i ; L[Vx : A \vdash e : B] \rightarrow j \rrbracket \Longrightarrow i = j$
 ⟨proof⟩

lemma (in *Interpretation*) *Functional*: $\llbracket L[\varphi] \rightarrow i1 ; L[\varphi] \rightarrow i2 \rrbracket \Longrightarrow i1 = i2$

⟨proof⟩

lemma (in *Interpretation*) *MorphismsPreserved*:

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

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

⟨proof⟩

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

⟨proof⟩

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⟩

record (*t*,*f*) *Axioms* =

aAxioms :: (*t*,*f*) *Language set*
aSignature :: (*t*,*f*) *Signature (aS1)*

locale *Axioms* =

fixes *Ax* :: (*t*,*f*) *Axioms (structure)*
assumes *AxT*: $(aAxioms\ Ax) \subseteq \{(Vx : A \vdash e1 \equiv e2 : B) \mid A\ B\ e1\ e2 . Sig\ iS \triangleright Vx : A \vdash e1 \equiv e2 : B\}$

$(aSignature\ Ax) \triangleright (Vx : A \vdash e1 \equiv e2 : B)$
assumes $AxSig: Signature\ (aSignature\ Ax)$

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

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 $SubstWellDefined: \bigwedge C . \llbracket Sig\ S \triangleright (Vx : A \vdash e : B); Sig\ S \triangleright (Vx : B \vdash d : C) \rrbracket$
 $\implies Sig\ S \triangleright (Vx : A \vdash (sub\ e\ in\ d) : C)$
 $\langle proof \rangle$

inductive-set (in Axioms) Theory where

$Ax: A \in (aAxioms\ Ax) \implies A \in Theory$
| $Reft: Sig\ (aSignature\ Ax) \triangleright (Vx : A \vdash e : B) \implies (Vx : A \vdash e \equiv e : B) \in Theory$
| $Symm: (Vx : A \vdash e1 \equiv e2 : B) \in Theory \implies (Vx : A \vdash e2 \equiv e1 : B) \in Theory$
| $Trans: \llbracket (Vx : A \vdash e1 \equiv e2 : B) \in Theory ; (Vx : A \vdash e2 \equiv e3 : B) \in Theory \rrbracket$
 \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: $\varphi \in Theory \implies Sig\ aS \triangleright \varphi$
 $\langle proof \rangle$

lemma (in Axioms) Subst':

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

locale $Model = Interpretation\ I + Axioms\ Ax$

for $I :: ('t,'f,'o,'m)\ Interpretation\ (\mathbf{structure})$

and $Ax :: ('t,'f)\ Axioms +$

assumes $AxSound: \varphi \in (aAxioms\ Ax) \implies L\llbracket \varphi \rrbracket \rightarrow (IBool\ True)$

and $Seq[simp]: (aSignature\ Ax) = iS$

lemma (in Interpretation) Equiv:

assumes $L\llbracket Vx : A \vdash e1 \equiv e2 : B \rrbracket \rightarrow (IBool\ True)$

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

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

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) \in range explode

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

primrec *Expr2ZF* :: (*ZF, ZF*) *Expression \Rightarrow ZF where*
Expr2ZFX: Expr2ZF Vx = ZFTriple (nat2Nat 0) (nat2Nat 0) Empty
 $| Expr2ZFXe: Expr2ZF (f\ E@ e) = ZFTriple (SucNat (ZFTFst (Expr2ZF e)))$
 $(nat2Nat\ 1)$
 $(Opair\ f\ (Expr2ZF\ e))$

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*
 $\langle proof \rangle$

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

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

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

$\langle proof \rangle$

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

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

lemma *Expr2ZFinj*: $inj Expr2ZF$
 $\langle proof \rangle$

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**
 $ZF2m T \equiv inv-into \{TermEquivCl' T A e B \mid A e B . True\} m2ZF$

lemma *TDomain*: $TDomain (TermEquivCl' T A e B) = A$ $\langle proof \rangle$

lemma *TCodomain*: $TCodomain (TermEquivCl' T A e B) = B$ $\langle proof \rangle$

primrec *WellFormedToSet* :: $(ZF, ZF) Signature \Rightarrow nat \Rightarrow (ZF, ZF) Expression set$ **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*: $ZFAxioms T \implies (Expr2ZF ' (WellFormedToSet aS_T n)) \in range explode$
 $\langle proof \rangle$

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

lemma *TermSetInSet*: $ZFAxioms T \implies Expr2ZF ' (TermEquivClGen T A e B) \in range explode$
 $\langle proof \rangle$

lemma *m2ZFinj-on*: $ZFAxioms T \implies inj-on m2ZF \{TermEquivCl' T A e B \mid A e B . True\}$
 $\langle proof \rangle$

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

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 $CLDomain$: $ZFAxioms\ T \implies CLDomain\ T\ [A, e, B]_T = A$ $\langle proof \rangle$

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

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 $CanonicalCat'\ T \equiv ()$

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

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

lemma $CanonicalCat'MapsTo$:

assumes $f\ maps_{CanonicalCat'\ T}\ X\ to\ Y$ **and** $zx : ZFAxioms\ T$
shows $\exists\ ef.\ f = [X, ef, Y]_T \wedge Sig\ (aSignature\ T) \triangleright (Vx : X \vdash ef : Y)$
 $\langle proof \rangle$

lemma *CanonicalCatCat'*: $ZFAxioms\ T \implies Category\text{-}axioms\ (CanonicalCat'\ T)$
 ⟨proof⟩

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

definition *CanonicalInterpretation where*

CanonicalInterpretation $T \equiv \langle$
 ISignature = *aSignature* T ,
 ICategory = *CanonicalCat* T ,
 ITypes = $\lambda\ A.\ A$,
 IFunctions = $\lambda\ f.\ [SigDom\ (aSignature\ T)\ f,\ f\ E@ \ \forall x,\ SigCod\ (aSignature\ T)$
 $f]_T$
 \rangle

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

lemma *CIObj*: $Obj\ (CanonicalCat\ T) = BaseType\ (aSignature\ T)$
 ⟨proof⟩

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

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

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

lemma *CIId*: $\llbracket A \in BaseType\ (aSignature\ T) \rrbracket \implies Id\ (CanonicalCat\ T)\ A = [A, \forall x, A]_T$
 ⟨proof⟩

lemma *CIComp*:

assumes $ZFAxioms\ T$ **and** $Sig\ (aSignature\ T) \triangleright (\forall x : A \vdash e : B)$ **and** $Sig\ (aSignature\ T) \triangleright (\forall x : B \vdash d : C)$

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

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

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

lemma *CIInterpretation*: $ZFAxioms\ T \implies Interpretation\ (CI\ T)$
 ⟨proof⟩

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⟩

lemma *CIModel*: $ZFAxioms\ T \implies Model\ (CI\ T)\ T$
 ⟨proof⟩

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

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⟩

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*: $f \approx >_{CatDom\ F} g \implies F\ \#\# (f ::_{CatDom\ F} g) = (F\ \#\#$

$f) \text{ ;; } \text{CatCod } F (F \text{ ## } g)$
and $\text{FunctorId}: X \in \text{obj}_{\text{CatDom } F} \implies \exists Y \in \text{obj}_{\text{CatCod } F} \cdot F \text{ ##}$
 $(\text{id}_{\text{CatDom } F} X) = \text{id}_{\text{CatCod } F} Y$
and $\text{CatDom}[\text{simp}]: \text{Category}(\text{CatDom } F)$
and $\text{CatCod}[\text{simp}]: \text{Category}(\text{CatCod } F)$

locale $\text{FunctorM} = \text{PreFunctor} +$
assumes $\text{FunctorCompM}: f \text{ maps } \text{CatDom } F \text{ } X \text{ to } Y \implies (F \text{ ## } f) \text{ maps } \text{CatCod } F$
 $(F \text{ @@ } X) \text{ to } (F \text{ @@ } Y)$

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

locale $\text{Functor} = \text{FunctorM} + \text{FunctorExt}$

definition

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

lemma $\text{PreFunctorFunctor}[\text{simp}]: \text{Functor } F \implies \text{PreFunctor } F$
 $\langle \text{proof} \rangle$

lemmas $\text{functor-simps} = \text{PreFunctor.FunctorComp } \text{PreFunctor.FunctorId}$

definition

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

lemma $\text{functor-abbrevE}[\text{elim}]: \llbracket \text{Ftor } F : A \longrightarrow B ; \llbracket (\text{Functor } F) ; (\text{CatDom } F =$
 $A) ; (\text{CatCod } F = B) \rrbracket \implies R \rrbracket \implies R$
 $\langle \text{proof} \rangle$

definition

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

lemma $\text{functor-comp-def}[\text{elim}]: \llbracket F \approx > ; ; G ; \llbracket \text{Functor } F ; \text{Functor } G ; \text{CatDom}$
 $G = \text{CatCod } F \rrbracket \implies R \rrbracket \implies R$
 $\langle \text{proof} \rangle$

lemma **(in** Functor) FunctorMapsTo :
assumes $f \in \text{mor}_{\text{CatDom } F}$

shows $F \#\# f \text{ maps } \text{CatCod } F (F \text{ @@ } (\text{dom } \text{CatDom } F f)) \text{ to } (F \text{ @@ } (\text{cod } \text{CatDom } F f))$
 ⟨proof⟩

lemma (in *Functor*) *FunctorCodDom*:

assumes $f \in \text{mor } \text{CatDom } F$
shows $\text{dom } \text{CatCod } F (F \#\# f) = F \text{ @@ } (\text{dom } \text{CatDom } F f)$ **and** $\text{cod } \text{CatCod } F (F \#\# f) = F \text{ @@ } (\text{cod } \text{CatDom } F f)$
 ⟨proof⟩

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

lemma (in *Functor*) *FunctorCompDef*:

assumes $f \approx\!>_{\text{CatDom } F} g$ **shows** $(F \#\# f) \approx\!>_{\text{CatCod } F} (F \#\# g)$
 ⟨proof⟩

lemma *FunctorComp*: $\llbracket \text{Ftor } F : A \longrightarrow B ; f \approx\!>_A g \rrbracket \implies F \#\# (f \text{ ;;}_A g) = (F \#\# f) \text{ ;;}_B (F \#\# g)$
 ⟨proof⟩

lemma *FunctorCompDef*: $\llbracket \text{Ftor } F : A \longrightarrow B ; f \approx\!>_A g \rrbracket \implies (F \#\# f) \approx\!>_B (F \#\# g)$
 ⟨proof⟩

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⟩

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⟩

lemma *FunctorId*:

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

lemma (in *Functor*) *DomFunctor*: $f \in \text{mor } \text{CatDom } F \implies \text{dom } \text{CatCod } F (F \#\# f) = F \text{ @@ } (\text{dom } \text{CatDom } F f)$
 ⟨proof⟩

lemma (in *Functor*) *CodFunctor*: $f \in \text{mor } \text{CatDom } F \implies \text{cod } \text{CatCod } F (F \#\# f) = F \text{ @@ } (\text{cod } \text{CatDom } F f)$
 ⟨proof⟩

lemma (in *Functor*) *FunctorId3Dom*:

assumes $f \in \text{mor}_{\text{CatDom } F}$
shows $F \#\# (\text{id}_{\text{CatDom } F} (\text{dom}_{\text{CatDom } F} f)) = \text{id}_{\text{CatCod } F} (\text{dom}_{\text{CatCod } F} (F \#\# f))$
 $\langle \text{proof} \rangle$

lemma (in *Functor*) *FunctorId3Cod*:

assumes $f \in \text{mor}_{\text{CatDom } F}$
shows $F \#\# (\text{id}_{\text{CatDom } F} (\text{cod}_{\text{CatDom } F} f)) = \text{id}_{\text{CatCod } F} (\text{cod}_{\text{CatCod } F} (F \#\# f))$
 $\langle \text{proof} \rangle$

lemma (in *PreFunctor*) *FmToFo*: $\llbracket X \in \text{obj}_{\text{CatDom } F} ; Y \in \text{obj}_{\text{CatCod } F} ; F \#\# (\text{id}_{\text{CatDom } F} X) = \text{id}_{\text{CatCod } F} Y \rrbracket \implies F \text{@@} X = Y$
 $\langle \text{proof} \rangle$

lemma *MakeFtorPreFtor*:

assumes *PreFunctor* F **shows** *PreFunctor* (*MakeFtor* F)
 $\langle \text{proof} \rangle$

lemma *MakeFtorMor*: $f \in \text{mor}_{\text{CatDom } F} \implies \text{MakeFtor } F \#\# f = F \#\# f$
 $\langle \text{proof} \rangle$

lemma *MakeFtorObj*:

assumes *PreFunctor* F **and** $X \in \text{obj}_{\text{CatDom } F}$
shows $\text{MakeFtor } F \text{@@} X = F \text{@@} X$
 $\langle \text{proof} \rangle$

lemma *MakeFtor*: **assumes** *FunctorM* F **shows** *Functor* (*MakeFtor* F)
 $\langle \text{proof} \rangle$

definition

IdentityFunctor' :: $(\text{'o}, \text{'m}, \text{'a})$ *Category-scheme* \Rightarrow $(\text{'o}, \text{'o}, \text{'m}, \text{'m}, \text{'a}, \text{'a})$ *Functor* (*FId'* - [70]) **where**
IdentityFunctor' $C \equiv (\llbracket \text{CatDom} = C , \text{CatCod} = C , \text{MapM} = (\lambda f . f) \rrbracket)$

definition

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

lemma *IdFtor'PreFunctor*: *Category* $C \implies \text{PreFunctor}$ (*FId'* C)
 $\langle \text{proof} \rangle$

lemma *IdFtor'Obj*:

assumes *Category* C **and** $X \in \text{obj}_{\text{CatDom}} (\text{FId}' C)$
shows $(\text{FId}' C) \text{@@} X = X$
 $\langle \text{proof} \rangle$

lemma *IdFtor'FtorM*:

assumes *Category C* **shows** *FunctorM (FId' C)*

<proof>

lemma *IdFtorFtor*: *Category C* \implies *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 \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* \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 \langle

CatDom = *CatDom F* ,

CatCod = *CatCod G* ,

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

\rangle

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 $G \#\# (F \#\# (f ::_{CatDom\ F}\ g)) = (G \#\# (F \#\# f)) ::_{CatCod\ G}\ (G \#\# (F \#\# g))$
 $\langle proof \rangle$

lemma *FtorCompId*:

assumes $a: X \in (Obj\ (CatDom\ F))$
and $F \approx > ::;$ G
shows $G \#\# (F \#\# (id_{CatDom\ F}\ X)) = id_{CatCod\ G}\ (G \@\@ (F \@\@ X)) \wedge G \@\@ (F \@\@ X) \in (Obj\ (CatCod\ G))$
 $\langle proof \rangle$

lemma *FtorCompIdDef*:

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

lemma *FunctorCompMapsTo*:

assumes $f \in mor_{CatDom}\ (F ::;$ $G)$ **and** $F \approx > ::;$ G
shows $(G \#\# (F \#\# f))\ maps_{CatCod\ G}\ (G \@\@ (F \@\@ (dom_{CatDom}\ F\ f)))\ to\ (G \@\@ (F \@\@ (cod_{CatDom}\ F\ f)))$
 $\langle proof \rangle$

lemma *FunctorCompMapsTo2*:

assumes $f \in mor_{CatDom}\ (F ::;$ $G)$
and $F \approx > ::;$ G
and $PreFunctor\ (F ::;$ $G)$
shows $((F ::;$ $G) \#\# f)\ maps_{CatCod}\ (F ::;$ $G)\ ((F ::;$ $G) \@\@ (dom_{CatDom}\ (F ::;$ $G)\ f))\ to\ ((F ::;$ $G) \@\@ (cod_{CatDom}\ (F ::;$ $G)\ f))$
 $\langle proof \rangle$

lemma *FunctorCompMapsTo3*:

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

lemma *FtorCompPreFtor*:

assumes $F \approx > ::;$ G

shows $PreFunctor (F ;; G)$
 $\langle proof \rangle$

lemma $FtorCompM$:
assumes $F \approx > ; ; G$
shows $FunctorM (F ;; G)$
 $\langle proof \rangle$

lemma $FtorComp$:
assumes $F \approx > ; ; G$
shows $Functor (F ;; G)$
 $\langle proof \rangle$

lemma (in $Functor$) $FunctorPreservesIso$:
assumes $ciso\ CatDom\ F\ k$
shows $ciso\ CatCod\ F\ (F\ \#\#\ k)$
 $\langle proof \rangle$

declare $PreFunctor.CatDom[simp]\ PreFunctor.CatCod [simp]$

lemma $FunctorMFunctor[simp]$: $Functor\ F \implies FunctorM\ F$
 $\langle proof \rangle$

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

end

5 Natural Transformation

theory $NatTrans$
imports $Functors$
begin

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

abbreviation

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

definition $NTCatDom \eta \equiv CatDom (NTDom \eta)$

definition $NTCatCod \eta \equiv CatCod (NTCod \eta)$

locale $NatTransExt =$

fixes $\eta :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans$ (**structure**)

assumes $NTExt : NatTransMap \eta \in extensional (Obj (NTCatDom \eta))$

locale $NatTransP =$

fixes $\eta :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) NatTrans$ (**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 $NatTransMapsTo : X \in objNTCatDom \eta \implies$

$(\eta \$\$ X) mapsNTCatCod \eta ((NTDom \eta) @@ X) to ((NTCod$

$\eta) @@ X)$

and $NatTrans : f mapsNTCatDom \eta X to Y \implies$

$((NTDom \eta) \#\# f) ;;NTCatCod \eta (\eta \$\$ Y) = (\eta \$\$ X) ;;NTCatCod \eta$

$((NTCod \eta) \#\# f)$

locale $NatTrans = NatTransP + NatTransExt$

lemma $[simp] : NatTrans \eta \implies NatTransP \eta$

$\langle proof \rangle$

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

$nt-abbrev (NT - : - \implies - [81])$ **where**

$NT f : F \implies G \equiv (NatTrans f) \wedge (NTDom f = F) \wedge (NTCod f = G)$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma $MakeNT-dom : NTCatDom f = NTCatDom (MakeNT f)$

<proof>

lemma *MakeNT-cod*: $NTCatCod\ f = NTCatCod\ (MakeNT\ f)$
<proof>

lemma *MakeNTApp*: $X \in Obj\ (NTCatDom\ (MakeNT\ f)) \implies f\ \$\$ X = (MakeNT\ f)\ \$\$ X$
<proof>

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>

definition

$NTCompDefined :: ('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans$
 $\implies ('o1, 'o2, 'm1, 'm2, 'a, 'b)\ NatTrans \implies bool\ (\mathbf{infixl}\ \approx>\cdot\ 65)$

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]*: $\llbracket \eta1 \approx>\cdot\ \eta2 ; \llbracket NatTrans\ \eta1 ; NatTrans\ \eta2 ; NT-CatDom\ \eta2 = NTCatDom\ \eta1 ;$
 $NTCatCod\ \eta2 = NTCatCod\ \eta1 ; NTCod\ \eta1 = NTDom\ \eta2 \rrbracket$
 $\implies R \rrbracket \implies R$
<proof>

lemma *NTCompDefinedI*: $\llbracket NatTrans\ \eta1 ; NatTrans\ \eta2 ; NTCatDom\ \eta2 = NT-CatDom\ \eta1 ;$
 $NTCatCod\ \eta2 = NTCatCod\ \eta1 ; NTCod\ \eta1 = NTDom\ \eta2 \rrbracket$
 $\implies \eta1 \approx>\cdot\ \eta2$
<proof>

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>

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>

lemma *NatTransExt*:

assumes $\text{NatTrans } \eta1$ **and** $\text{NatTrans } \eta2$ **and** $\text{NTDom } \eta1 = \text{NTDom } \eta2$ **and**
 $\text{NTCod } \eta1 = \text{NTCod } \eta2$
and $\bigwedge X . X \in \text{Obj } (\text{NTCatDom } \eta1) \implies \eta1 \ \$\$ X = \eta2 \ \$\$ X$
shows $\eta1 = \eta2$
 $\langle \text{proof} \rangle$

definition

$\text{IdNatTrans}' :: ('o1, 'o2, 'm1, 'm2, 'a1, 'a2) \text{ Functor} \Rightarrow ('o1, 'o2, 'm1, 'm2, 'a1, 'a2) \text{ NatTrans}$ **where**
 $\text{IdNatTrans}' F \equiv \langle$
 $\text{NTDom} = F$,
 $\text{NTCod} = F$,
 $\text{NatTransMap} = \lambda X . \text{id}_{\text{CatCod } F} (F \ @\@ X)$
 \rangle

definition $\text{IdNatTrans } F \equiv \text{MakeNT}(\text{IdNatTrans}' F)$

lemma $\text{IdNatTrans-map}: X \in \text{obj}_{\text{CatDom } F} \implies (\text{IdNatTrans } F) \ \$\$ X = \text{id}_{\text{CatCod } F} (F \ @\@ X)$
 $\langle \text{proof} \rangle$

lemmas $\text{IdNatTrans-defs} = \text{IdNatTrans-def } \text{IdNatTrans}'\text{-def } \text{MakeNT-def } \text{IdNatTrans-map } \text{NTCatCod-def } \text{NTCatDom-def}$

lemma $\text{IdNatTransNatTrans}'$: $\text{Functor } F \implies \text{NatTransP}(\text{IdNatTrans}' F)$
 $\langle \text{proof} \rangle$

lemma $\text{IdNatTransNatTrans}$: $\text{Functor } F \implies \text{NatTrans } (\text{IdNatTrans } F)$
 $\langle \text{proof} \rangle$

definition

$\text{NatTransComp}' :: ('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{ NatTrans} \Rightarrow$
 $('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{ NatTrans} \Rightarrow$
 $('o1, 'o2, 'm1, 'm2, 'a, 'b) \text{ NatTrans}$ (**infixl** $\cdot 1$ 75) **where**
 $\text{NatTransComp}' \eta1 \eta2 = \langle$
 $\text{NTDom} = \text{NTDom } \eta1$,
 $\text{NTCod} = \text{NTCod } \eta2$,
 $\text{NatTransMap} = \lambda X . (\eta1 \ \$\$ X) \ ;\ ; \ \text{NTCatCod } \eta1 \ (\eta2 \ \$\$ X)$
 \rangle

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

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

lemma $\text{NatTransComp-Comp2}$: $\llbracket x \in \text{Obj } (\text{NTCatDom } f) ; f \approx \triangleright \cdot g \rrbracket \implies (f \cdot g) \ \$\$ x = (f \ \$\$ x) \ ;\ ; \ \text{NTCatCod } f \ (g \ \$\$ x)$

<proof>

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

NatTransComp-Comp1 NTCatCod-def NTCatDom-def

lemma *[simp]: $\eta 1 \approx > \cdot \eta 2 \implies \text{NatTrans } \eta 1$* *<proof>*

lemma *[simp]: $\eta 1 \approx > \cdot \eta 2 \implies \text{NatTrans } \eta 2$* *<proof>*

lemma *NTCatDom: $\eta 1 \approx > \cdot \eta 2 \implies \text{NTCatDom } \eta 1 = \text{NTCatDom } \eta 2$* *<proof>*

lemma *NTCatCod: $\eta 1 \approx > \cdot \eta 2 \implies \text{NTCatCod } \eta 1 = \text{NTCatCod } \eta 2$* *<proof>*

lemma *[simp]: $\eta 1 \approx > \cdot \eta 2 \implies \text{NTCatDom } (\eta 1 \cdot 1 \eta 2) = \text{NTCatDom } \eta 1$* *<proof>*

lemma *[simp]: $\eta 1 \approx > \cdot \eta 2 \implies \text{NTCatCod } (\eta 1 \cdot 1 \eta 2) = \text{NTCatCod } \eta 1$* *<proof>*

lemma *[simp]: $\eta 1 \approx > \cdot \eta 2 \implies \text{NTCatDom } (\eta 1 \cdot \eta 2) = \text{NTCatDom } \eta 1$* *<proof>*

lemma *[simp]: $\eta 1 \approx > \cdot \eta 2 \implies \text{NTCatCod } (\eta 1 \cdot \eta 2) = \text{NTCatCod } \eta 1$* *<proof>*

lemma *[simp]: $\text{NatTrans } \eta \implies \text{Category}(\text{NTCatDom } \eta)$* *<proof>*

lemma *[simp]: $\text{NatTrans } \eta \implies \text{Category}(\text{NTCatCod } \eta)$* *<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 $\implies (\text{IdNatTrans } (\text{NTDom } f)) \approx > \cdot f$*

<proof>

lemma *IdNatTransCompDefCod: NatTrans f $\implies f \approx > \cdot (\text{IdNatTrans } (\text{NTCod } f))$*

<proof>

lemma *NatTransCompDefCod:*

assumes *NatTrans η and f maps_{NTCatDom η} X to Y*

shows *$(\eta \ \$\$ X) \approx >_{\text{NTCatCod } \eta} (\text{NTCod } \eta \ \#\# f)$*

<proof>

lemma *NatTransCompDefDom:*

assumes *NatTrans η and f maps_{NTCatDom η} X to Y*

shows *$(\text{NTDom } \eta \ \#\# f) \approx >_{\text{NTCatCod } \eta} (\eta \ \$\$ Y)$*

<proof>

lemma *NatTransCompCompDef:*

assumes *$\eta 1 \approx > \cdot \eta 2$ and $X \in \text{obj}_{\text{NTCatDom } \eta 1}$*

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

<proof>

lemma *NatTransCompNatTrans':*

assumes *$\eta 1 \approx > \cdot \eta 2$*

shows *$\text{NatTransP } (\eta 1 \cdot 1 \eta 2)$*

<proof>

lemma *NatTransCompNatTrans*: $\eta1 \approx\>\cdot \eta2 \implies \text{NatTrans } (\eta1 \cdot \eta2)$

<proof>

definition

CatExp' :: ('o1,'m1,'a) *Category-scheme* \implies ('o2,'m2,'b) *Category-scheme* \implies
 (('o1, 'o2, 'm1, 'm2, 'a, 'b) *Functor*,
 ('o1, 'o2, 'm1, 'm2, 'a, 'b) *NatTrans*) *Category* **where**

CatExp' A B \equiv \langle

Category.Obj = $\{F \cdot \text{Ftor } F : A \longrightarrow B\}$,

Category.Mor = $\{\eta \cdot \text{NatTrans } \eta \wedge \text{NTCatDom } \eta = A \wedge \text{NTCatCod } \eta = B\}$

,

Category.Dom = *NTDom* ,

Category.Cod = *NTCod* ,

Category.Id = *IdNatTrans* ,

Category.Comp = $\lambda f g. (f \cdot g)$

\rangle

definition *CatExp A B* $\equiv \text{MakeCat}(\text{CatExp}' A B)$

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*: $\llbracket \text{Category } A ; \text{Category } B \rrbracket \implies \text{Category } (\text{CatExp } A B)$

<proof>

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

lemma *CatExpDom*: $f \in \text{Mor } (\text{CatExp } A \ B) \implies \text{dom}_{\text{CatExp } A \ B} f = \text{NTDom } f$
 ⟨proof⟩

lemma *CatExpCod*: $f \in \text{Mor } (\text{CatExp } A \ B) \implies \text{cod}_{\text{CatExp } A \ B} f = \text{NTCod } f$
 ⟨proof⟩

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

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

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⟩

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

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**
 $ZFfun \ d \ r \ f \equiv \text{Opair } (\text{Opair } \ d \ r) (\text{Lambda } \ d \ f)$

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

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

definition
 $ZFfunApp :: ZF \Rightarrow ZF \Rightarrow ZF \ (|\text{@}| \ 73)$ **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 $isZFfunE[elim]$: $\llbracket isZFfun\ f ;$

$\llbracket isOpair\ f ; isOpair\ (Fst\ f) ; isFun\ (Snd\ f) ;$
 $((Snd\ f)\ |\subseteq| (Domain\ (Snd\ f))\ |\times| (Range\ (Snd\ f))) ;$
 $(Domain\ (Snd\ f) = |dom| f) \wedge (Range\ (Snd\ f)\ |\subseteq| |cod| f) \rrbracket \Longrightarrow R \rrbracket \Longrightarrow R$
 $\langle proof \rangle$

definition

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

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 $x\ |\in| A\ |\times| B$
shows $Fst\ x\ |\in| A \wedge Snd\ x\ |\in| B \wedge isOpair\ x$
 $\langle proof \rangle$

lemma $ZFfunDomainOpair$:

assumes $isFun\ f$
and $x \in |Domain\ f|$
shows $Opair\ x\ (app\ f\ x) \in |f|$
 $\langle proof \rangle$

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

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

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

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

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 $f = g$
 $\langle proof \rangle$

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

lemma $ZFfunComp$:
assumes $\forall x . x \in |A| \longrightarrow f\ x \in |B|$
shows $(ZFfun\ A\ B\ f) \ @\ (ZFfun\ B\ C\ g) = ZFfun\ A\ C\ (g\ o\ f)$
 $\langle proof \rangle$

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

lemma *ZFfunCompAppZFfun*:

assumes *isZFfun f* **and** *isZFfun g* **and** $|dom|g = |cod|f$

shows $isZFfun (f \circ g)$

<proof>

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 $f \circ g \circ h = f \circ (g \circ h)$

<proof>

lemma *ZFfunCompAppDomCod*:

assumes *isZFfun f* **and** *isZFfun g* **and** $|dom|g = |cod|f$

shows $|dom|(f \circ g) = |dom|f \wedge |cod|(f \circ g) = |cod|g$

<proof>

lemma *ZFfunIdLeft*:

assumes *a: isZFfun f* **shows** $(ZFfun (|dom|f) (|dom|f) (\lambda x. x)) \circ f = f$

<proof>

lemma *ZFfunIdRight*:

assumes *a: isZFfun f* **shows** $f \circ (ZFfun (|cod|f) (|cod|f) (\lambda x. x)) = f$

<proof>

lemma *SETCategory*: *Category(SET)*

<proof>

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

<proof>

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

<proof>

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

<proof>

lemma *SETdom*: $isZFfun (ZFfun A B f) \implies dom_{SET} ZFfun A B f = A$

<proof>

lemma *SETId*: **assumes** $x \in X$ **shows** $(Id_{SET} X) \circ x = x$

<proof>

lemma *SETCompE[elim]*: $\llbracket f \approx_{SET} g ; \llbracket isZFfun f ; isZFfun g ; |cod|f = |dom|g \rrbracket \implies R \rrbracket \implies R$

<proof>

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

<proof>

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

lemma *SETCompAt*:

assumes $f \approx_{>SET} g$ **and** $x \mid \in \mid \mid dom \mid f$ **shows** $(f \;;_{SET} g) \mid @ \mid x = g \mid @ \mid (f \mid @ \mid x)$
 ⟨*proof*⟩

lemma *SETZFfun*:

assumes $f \text{ maps}_{SET} X \text{ to } Y$ **shows** $f = ZFfun X Y (\lambda x . f \mid @ \mid x)$
 ⟨*proof*⟩

lemma *SETfunDomAppCod*:

assumes $f \text{ maps}_{SET} X \text{ to } Y$ **and** $x \mid \in \mid X$
shows $f \mid @ \mid x \mid \in \mid Y$
 ⟨*proof*⟩

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

definition

$ZF2mor (z2m1- [70] 70)$ **where**
 $ZF2mor C f \equiv THE m . m \in mor_C \wedge m2z_C m = f$

definition

$HOMCollection C X Y \equiv \{m2z_C f \mid f . f \text{ maps}_C X \text{ to } Y\}$

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 $HOMSetIsSet: \llbracket X \in obj ; Y \in obj \rrbracket \Longrightarrow HOMCollection C X Y \in range \text{explode}$
and $m2zExt: mor2ZF C \in extensional (Mor C)$

lemma *[elim]*: $\llbracket LSCategory C ;$

$\llbracket Category C ; \llbracket x \in mor_C ; y \in mor_C ; m2z_C x = m2z_C y \rrbracket \Longrightarrow x = y;$
 $\llbracket X \in obj_C ; Y \in obj_C \rrbracket \Longrightarrow HOMCollection C X Y \in range \text{explode} \rrbracket \Longrightarrow R \rrbracket \Longrightarrow R$
 ⟨*proof*⟩

definition

$HomFtorMap :: (\prime o, \prime m, \prime a)$ *LSCategory-scheme* $\Rightarrow \prime o \Rightarrow \prime m \Rightarrow ZF (Hom1[-, -] [65, 65] 65)$ **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

$HomFtor' :: ('o, 'm, 'a) LSCategory-scheme \Rightarrow 'o \Rightarrow$
 $('o, ZF, 'm, ZF, (\text{mor}2ZF :: 'm \Rightarrow ZF, \dots :: 'a), \text{unit}) \text{ Functor } (HomP1[-, -] [65]$
 65) **where**
 $HomFtor' C X \equiv ()$
 $CatDom = C,$
 $CatCod = SET,$
 $MapM = \lambda g . Hom_C[X, g]$
 \rangle

definition $HomFtor (Hom[-, -] [65] 65)$ **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** $(m2z f) \mid \in \mid (Hom X Y)$
 $\langle proof \rangle$

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 $f \mid \in \mid (Hom X Y)$
shows $z2m f$ maps X to $Y \wedge m2z (z2m f) = f$
 $\langle proof \rangle$

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

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 $HomFtorMapsTo$:

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

lemma *HomFtorMor*:

assumes *LSCategory* C **and** $X \in \text{obj}_C$ **and** $f \in \text{mor}_C$
shows $\text{Hom}_C[X,f] \in \text{Mor SET}$ **and** $\text{dom}_{\text{SET}}(\text{Hom}_C[X,f]) = \text{Hom}_C X (\text{dom}_C f)$
and $\text{cod}_{\text{SET}}(\text{Hom}_C[X,f]) = \text{Hom}_C X (\text{cod}_C f)$
 $\langle \text{proof} \rangle$

lemma *HomFtorCompDef'*:

assumes *LSCategory* C **and** $X \in \text{obj}_C$ **and** $f \approx \triangleright_C g$
shows $(\text{Hom}_C[X,f]) \approx \triangleright_{\text{SET}'} (\text{Hom}_C[X,g])$
 $\langle \text{proof} \rangle$

lemma *HomFtorDist'*:

assumes $a: \text{LSCategory } C$ **and** $b: X \in \text{obj}_C$ **and** $c: f \approx \triangleright_C g$
shows $(\text{Hom}_C[X,f]) \text{ ;;}_{\text{SET}'} (\text{Hom}_C[X,g]) = \text{Hom}_C[X,f \text{ ;;}_C g]$
 $\langle \text{proof} \rangle$

lemma *HomFtorDist*:

assumes *LSCategory* C **and** $X \in \text{obj}_C$ **and** $f \approx \triangleright_C g$
shows $(\text{Hom}_C[X,f]) \text{ ;;}_{\text{SET}} (\text{Hom}_C[X,g]) = \text{Hom}_C[X,f \text{ ;;}_C g]$
 $\langle \text{proof} \rangle$

lemma *HomFtorId'*:

assumes $a: \text{LSCategory } C$ **and** $b: X \in \text{obj}_C$ **and** $c: Y \in \text{obj}_C$
shows $\text{Hom}_C[X, \text{id}_C Y] = \text{id}_{\text{SET}'}(\text{Hom}_C X Y)$
 $\langle \text{proof} \rangle$

lemma *HomFtorId*:

assumes *LSCategory* C **and** $X \in \text{obj}_C$ **and** $Y \in \text{obj}_C$
shows $\text{Hom}_C[X, \text{id}_C Y] = \text{id}_{\text{SET}}(\text{Hom}_C X Y)$
 $\langle \text{proof} \rangle$

lemma *HomFtorObj'*:

assumes $a: \text{LSCategory } C$
and $b: \text{PreFunctor } (\text{HomP}_C[X, -])$ **and** $c: X \in \text{obj}_C$ **and** $d: Y \in \text{obj}_C$
shows $(\text{HomP}_C[X, -]) \text{ @@ } Y = \text{Hom}_C X Y$
 $\langle \text{proof} \rangle$

lemma *HomFtorFtor'*:

assumes $a: \text{LSCategory } C$
and $b: X \in \text{obj}_C$
shows $\text{FunctorM } (\text{HomP}_C[X, -])$
 $\langle \text{proof} \rangle$

lemma *HomFtorFtor*:

assumes $a: LSCategory\ C$
and $b: X \in obj_C$
shows $Functor\ (Hom_C[X,-])$
 $\langle proof \rangle$

lemma $HomFtorObj$:
assumes $LSCategory\ C$
and $X \in obj_C$ **and** $Y \in obj_C$
shows $(Hom_C[X,-])\ @\@ Y = Hom_C\ X\ Y$
 $\langle proof \rangle$

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

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

lemma $HomContraAt: x \in |Hom_C\ (cod_C\ f)\ X| \implies (Hom_C[f,X])\ |@|\ x =$
 $m2z_C\ (f\ ;;_C\ (z2m_C\ x))$
 $\langle proof \rangle$

lemma $mor2ZF-Op: mor2ZF\ (Op\ C) = mor2ZF\ C$
 $\langle proof \rangle$

lemma $mor-Op: mor_{Op\ C} = mor_C\ \langle proof \rangle$

lemma $obj-Op: obj_{Op\ C} = obj_C\ \langle proof \rangle$

lemma $ZF2mor-Op: ZF2mor\ (Op\ C)\ f = ZF2mor\ C\ f$
 $\langle proof \rangle$

lemma $mapsTo-Op: f\ maps_{Op\ C}\ Y\ to\ X = f\ maps_C\ X\ to\ Y$
 $\langle proof \rangle$

lemma $HOMCollection-Op: HOMCollection\ (Op\ C)\ X\ Y = HOMCollection\ C\ Y$
 X
 $\langle proof \rangle$

lemma *Hom-Op*: $\text{Hom}_{\text{Op } C} X Y = \text{Hom}_C Y X$

<proof>

lemma *HomFtorContra'*: $\text{Hom}_C[-, X] = \text{Hom}_{\text{Op } C}[X, -]$

<proof>

lemma *HomFtorContra*: $\text{Hom}_C[-, X] = \text{Hom}_{\text{Op } C}[X, -]$

<proof>

lemma *HomFtorContraDom*: $\text{CatDom} (\text{Hom}_C[-, X]) = \text{Op } C$

<proof>

lemma *HomFtorContraCod*: $\text{CatCod} (\text{Hom}_C[-, X]) = \text{SET}$

<proof>

lemma *LSCategory-Op*: **assumes** *LSCategory* C **shows** *LSCategory* $(\text{Op } C)$

<proof>

lemma *HomFtorContraFtor*:

assumes *LSCategory* C

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

shows $\text{Ftor} (\text{Hom}_C[-, X]) : (\text{Op } C) \longrightarrow \text{SET}$

<proof>

lemma *HomFtorOpObj*:

assumes *LSCategory* C

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

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

<proof>

lemma *HomCHomOp*: $\text{Hom}_C C[g, X] = \text{Hom}_{\text{Op } C}[X, g]$

<proof>

lemma *HomFtorContraMapsTo*:

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

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

<proof>

lemma *HomFtorContraMor*:

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

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

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

<proof>

lemma *HomContraMor*:

assumes *LSCategory* C **and** $f \in \text{Mor } C$

shows $(Hom_C[-, X]) \# \# f = Hom_C C[f, X]$
 $\langle proof \rangle$

lemma *HomCHom*:

assumes *LSCategory* C **and** $f \in Mor\ C$ **and** $g \in Mor\ C$
shows $(Hom_C C[g, dom_C f]) \# \# SET (Hom_C C[dom_C g, f]) = (Hom_C C[cod_C g, f]) \# \# SET$
 $(Hom_C C[g, cod_C f])$
 $\langle proof \rangle$

end

7 Yoneda

theory *Yoneda*

imports *NatTrans SetCat*

begin

definition $YFtorNT' C f \equiv (\lambda B. (NTDom = Hom_C C[-, dom_C f], NTCod = Hom_C C[-, cod_C f]))$,

$$NatTransMap = \lambda B. Hom_C C[B, f])$$

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

lemmas $YFtorNT-defs = YFtorNT'-def\ YFtorNT-def\ MakeNT-def$

lemma $YFtorNTCatDom: NTCatDom (YFtorNT C f) = Op\ C$
 $\langle proof \rangle$

lemma $YFtorNTCatCod: NTCatCod (YFtorNT C f) = SET$
 $\langle proof \rangle$

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

definition

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

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

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

lemma *YFtorNTNatTrans'*:

assumes *LSCategory C* **and** $f \in \text{Mor } C$

shows *NatTransP (YFtorNT' C f)*

<proof>

lemma *YFtorNTNatTrans*:

assumes *LSCategory C* **and** $f \in \text{Mor } C$

shows *NatTrans (YFtorNT C f)*

<proof>

lemma *YFtorNTMor*:

assumes *LSCategory C* **and** $f \in \text{Mor } C$

shows *YFtorNT C f \in Mor (CatExp (Op C) SET)*

<proof>

lemma *YFtorNtMapsTo*:

assumes *LSCategory C* **and** $f \in \text{Mor } C$

shows *YFtorNT C f maps_{CatExp (Op C) SET} (Hom_C[-, dom_C f]) to (Hom_C[-, cod_C f])*

<proof>

lemma *YFtorNTCompDef*:

assumes *LSCategory C* **and** $f \approx_{>C} g$

shows *YFtorNT C f \approx_{>CatExp (Op C) SET} YFtorNT C g*

<proof>

lemma *PreSheafCat*: *LSCategory C \implies Category (CatExp (Op C) SET)*

<proof>

lemma *YFtor'Obj1*:

assumes $X \in \text{Obj} (\text{CatDom} (\text{YFtor}' C))$ **and** *LSCategory C*

shows $(\text{YFtor}' C) \#\# (\text{Id} (\text{CatDom} (\text{YFtor}' C)) X) = \text{Id} (\text{CatCod} (\text{YFtor}' C)) (\text{Hom}_C [-, X])$

<proof>

lemma *YFtorPreFtor*:

assumes *LSCategory C*

shows *PreFunctor (YFtor' C)*

<proof>

lemma *YFtor'Obj*:

assumes $X \in \text{Obj} (\text{CatDom} (\text{YFtor}' C))$

and *LSCategory C*

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

<proof>

lemma *YFtorFtor'*:

assumes *LSCategory C*

shows *FunctorM (YFtor' C)*

$\langle proof \rangle$

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

lemma *YFtorObj*:
assumes *LSCategory C* **and** $X \in \text{Obj } C$
shows $(YFtor C) @@ X = \text{Hom}_C [-, X]$
 $\langle proof \rangle$

lemma *YFtorObj2*:
assumes *LSCategory C* **and** $X \in \text{Obj } C$ **and** $Y \in \text{Obj } C$
shows $((YFtor C) @@ Y) @@ X = \text{Hom}_C X Y$
 $\langle proof \rangle$

lemma *YFtorMor*: $\llbracket LSCategory C ; f \in \text{Mor } C \rrbracket \implies (YFtor C) \#\# f = YFtorNT C f$
 $\langle proof \rangle$

definition *YMap C X η* $\equiv (\eta \ \$\$ X) |@| (m2z_C (id_C X))$

definition *YMapInv' C X F x* $\equiv \langle$
 $\text{NTDom} = ((YFtor C) @@ X),$
 $\text{NTCod} = F,$
 $\text{NatTransMap} = \lambda B . ZFfun (\text{Hom}_C B X) (F @@ B) (\lambda f . (F \#\# (z2m_C$
 $f)) |@| x)$
 \rangle

definition *YMapInv C X F x* $\equiv \text{MakeNT} (YMapInv' C X F x)$

lemma *YMapInvApp*:
assumes $X \in \text{Obj } C$ **and** $B \in \text{Obj } C$ **and** *LSCategory C*
shows $(YMapInv C X F x) \ \$\$ B = ZFfun (\text{Hom}_C B X) (F @@ B) (\lambda f . (F \#\# (z2m_C f)) |@| x)$
 $\langle proof \rangle$

lemma *YMapImage*:
assumes *LSCategory C* **and** *Ftor F : (Op C) \longrightarrow SET* **and** $X \in \text{Obj } C$
and *NT $\eta : (YFtor C) @@ X \implies F$*
shows $(YMap C X \eta) | \in | (F @@ X)$
 $\langle proof \rangle$

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)*
 $\langle proof \rangle$

lemma *YMapInvNatTrans*:

assumes *LSCategory* C **and** $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$ **and** x
 $|\in| (F\ @\@ X)$
shows $NatTrans\ (YMapInv\ C\ X\ F\ x)$
<proof>

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>

lemma *YMap1*:

assumes *LSCat*: *LSCategory* C **and** $Ftor$: $Ftor\ F : (Op\ C) \longrightarrow SET$ **and** $X \in Obj\ C$
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 $x |\in| (F\ @\@ X)$
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 $YFtorNT\ C\ f = YMapInv\ C\ X\ (Hom_C[-, Y])\ (m2z_C\ f)$
<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 $z2m_C\ (YMap\ C\ X\ \eta)\ maps_C\ X\ to\ Y$
<proof>

lemma *YonedaFaithful*:

assumes *LSCategory* C **and** $f\ maps_C\ X\ to\ Y$ **and** $g\ maps_C\ X\ to\ Y$
and $YFtor\ C\ \#\#\ f = YFtor\ C\ \#\#\ g$
shows $f = g$
<proof>

lemma *YonedaEmbedding*:

```
  assumes LSCategory C and A ∈ Obj C and B ∈ Obj C and (YFtor C) @@ A  
= (YFtor C) @@ B  
  shows A = B  
  <proof>  
end
```

References

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