

Category Theory to Yoneda's Lemma

Greg O'Keefe

December 12, 2009

This development proves Yoneda's lemma and aims to be readable by humans. It only defines what is needed for the lemma: categories, functors and natural transformations. Limits, adjunctions and other important concepts are not included.

There is no explanation or discussion in this document. See [O'K04] for this and a survey of category theory formalisations.

Contents

1	Categories	2
1.1	Definitions	2
1.2	Lemmas	2
2	Set is a Category	4
2.1	Definitions	4
2.2	Simple Rules and Lemmas	4
2.3	Set is a Category	6
3	Functors	10
3.1	Definitions	10
3.2	Simple Lemmas	11
3.3	Identity Functor	12
4	HomFunctors	14
5	Natural Transformations	18
6	Yoneda's Lemma	20
6.1	The Sandwich Natural Transformation	20
6.2	Sandwich Components are Bijective	23

1 Categories

```
theory Cat
imports FuncSet
begin
```

1.1 Definitions

```
record ('o, 'a) category =
  ob :: 'o set (Ob1 70)
  ar :: 'a set (Ar1 70)
  dom :: 'a ⇒ 'o (Dom1 - [81] 70)
  cod :: 'a ⇒ 'o (Cod1 - [81] 70)
  id :: 'o ⇒ 'a (Id1 - [81] 80)
  comp :: 'a ⇒ 'a ⇒ 'a (infixl · 60)
```

definition

```
hom :: [('o, 'a, 'm) category-scheme, 'o, 'o] ⇒ 'a set (Hom1 - -) where
hom CC A B = { f. f ∈ ar CC & dom CC f = A & cod CC f = B }
```

```
locale category =
```

```
  fixes CC (structure)
```

```
  assumes dom-object [intro]:
```

```
    f ∈ Ar ⇒ Dom f ∈ Ob
```

```
  and cod-object [intro]:
```

```
    f ∈ Ar ⇒ Cod f ∈ Ob
```

```
  and id-left [simp]:
```

```
    f ∈ Ar ⇒ Id (Cod f) · f = f
```

```
  and id-right [simp]:
```

```
    f ∈ Ar ⇒ f · Id (Dom f) = f
```

```
  and id-hom [intro]:
```

```
    A ∈ Ob ⇒ Id A ∈ Hom A A
```

```
  and comp-types [intro]:
```

```
     $\bigwedge A B C. (comp\ CC) : (Hom\ B\ C) \rightarrow (Hom\ A\ B) \rightarrow (Hom\ A\ C)$ 
```

```
  and comp-associative [simp]:
```

```
    f ∈ Ar ⇒ g ∈ Ar ⇒ h ∈ Ar
```

```
    ⇒ Cod h = Dom g ⇒ Cod g = Dom f
```

```
    ⇒ f · (g · h) = (f · g) · h
```

1.2 Lemmas

```
lemma (in category) homI:
```

```
  assumes f ∈ Ar and Dom f = A and Cod f = B
```

```
  shows f ∈ Hom A B
```

```
  using assms by (auto simp add: hom-def)
```

```
lemma (in category) homE:
```

```
  assumes A ∈ Ob and B ∈ Ob and f ∈ Hom A B
```

```
  shows Dom f = A and Cod f = B
```

```
proof -
```

show $Dom\ f = A$ **using** *assms* **by** (*simp add: hom-def*)
show $Cod\ f = B$ **using** *assms* **by** (*simp add: hom-def*)
qed

lemma (*in category*) *id-arrow* [*intro*]:
assumes $A \in Ob$
shows $Id\ A \in Ar$
proof –
from $\langle A \in Ob \rangle$ **have** $Id\ A \in Hom\ A\ A$ **by** (*rule id-hom*)
thus $Id\ A \in Ar$ **by** (*simp add: hom-def*)
qed

lemma (*in category*) *id-dom-cod*:
assumes $A \in Ob$
shows $Dom\ (Id\ A) = A$ **and** $Cod\ (Id\ A) = A$
proof –
from $\langle A \in Ob \rangle$ **have** $1: Id\ A \in Hom\ A\ A$..
then show $Dom\ (Id\ A) = A$ **and** $Cod\ (Id\ A) = A$
by (*simp-all add: hom-def*)
qed

lemma (*in category*) *compI* [*intro*]:
assumes $f: f \in Ar$ **and** $g: g \in Ar$ **and** $Cod\ f = Dom\ g$
shows $g \cdot f \in Ar$
and $Dom\ (g \cdot f) = Dom\ f$
and $Cod\ (g \cdot f) = Cod\ g$
proof –
have $f \in Hom\ (Dom\ f)\ (Cod\ f)$ **using** f **by** (*simp add: hom-def*)
with $\langle Cod\ f = Dom\ g \rangle$ **have** $f\text{-homset}: f \in Hom\ (Dom\ f)\ (Dom\ g)$ **by** *simp*
have $g\text{-homset}: g \in Hom\ (Dom\ g)\ (Cod\ g)$ **using** g **by** (*simp add: hom-def*)
have $(op\ \cdot) : Hom\ (Dom\ g)\ (Cod\ g) \rightarrow Hom\ (Dom\ f)\ (Dom\ g) \rightarrow Hom\ (Dom\ f)\ (Cod\ g)$..
from *this* **and** $g\text{-homset}$
have $(op\ \cdot)\ g \in Hom\ (Dom\ f)\ (Dom\ g) \rightarrow Hom\ (Dom\ f)\ (Cod\ g)$
by (*rule funcset-mem*)
from *this* **and** $f\text{-homset}$
have $gf\text{-homset}: g \cdot f \in Hom\ (Dom\ f)\ (Cod\ g)$
by (*rule funcset-mem*)
thus $g \cdot f \in Ar$
by (*simp add: hom-def*)
from $gf\text{-homset}$ **show** $Dom\ (g \cdot f) = Dom\ f$ **and** $Cod\ (g \cdot f) = Cod\ g$
by (*simp-all add: hom-def*)
qed

end

2 Set is a Category

```
theory SetCat
imports Cat
begin
```

2.1 Definitions

```
record 'c set-arrow =
  set-dom :: 'c set
  set-func :: 'c  $\Rightarrow$  'c
  set-cod :: 'c set
```

definition

```
set-arrow :: ['c set, 'c set-arrow]  $\Rightarrow$  bool where
set-arrow U f  $\longleftrightarrow$  set-dom f  $\subseteq$  U & set-cod f  $\subseteq$  U
  & (set-func f): (set-dom f)  $\rightarrow$  (set-cod f)
  & set-func f  $\in$  extensional (set-dom f)
```

definition

```
set-id :: ['c set, 'c set]  $\Rightarrow$  'c set-arrow where
set-id U = ( $\lambda s \in Pow\ U. (set-dom=s, set-func=\lambda x \in s. x, set-cod=s)$ )
```

definition

```
set-comp :: ['c set-arrow, 'c set-arrow]  $\Rightarrow$  'c set-arrow (infix  $\odot$  70) where
set-comp g f =
  (
    set-dom = set-dom f,
    set-func = compose (set-dom f) (set-func g) (set-func f),
    set-cod = set-cod g
  )
```

definition

```
set-cat :: 'c set  $\Rightarrow$  ('c set, 'c set-arrow) category where
set-cat U =
  (
    ob = Pow U,
    ar = {f. set-arrow U f},
    dom = set-dom,
    cod = set-cod,
    id = set-id U,
    comp = set-comp
  )
```

2.2 Simple Rules and Lemmas

```
lemma set-objectI [intro]: A  $\subseteq$  U  $\Longrightarrow$  A  $\in$  ob (set-cat U)
by (simp add: set-cat-def)
```

```
lemma set-objectE [intro]: A  $\in$  ob (set-cat U)  $\Longrightarrow$  A  $\subseteq$  U
```

by (*simp add: set-cat-def*)

lemma *set-homI* [*intro*]:

assumes $A \subseteq U$

and $B \subseteq U$

and $f : A \rightarrow B$

and $f \in \text{extensional } A$

shows $(\setminus \text{set-dom}=A, \text{set-func}=f, \text{set-cod}=B) \in \text{hom } (\text{set-cat } U) A B$

using *assms* by (*simp add: set-cat-def hom-def set-arrow-def*)

lemma *set-dom* [*simp*]: $\text{dom } (\text{set-cat } U) f = \text{set-dom } f$

by (*simp add: set-cat-def*)

lemma *set-cod* [*simp*]: $\text{cod } (\text{set-cat } U) f = \text{set-cod } f$

by (*simp add: set-cat-def*)

lemma *set-id* [*simp*]: $\text{id } (\text{set-cat } U) A = \text{set-id } U A$

by (*simp add: set-cat-def*)

lemma *set-comp* [*simp*]: $\text{comp } (\text{set-cat } U) g f = g \odot f$

by (*simp add: set-cat-def*)

lemma *set-dom-cod-object-subset* [*intro*]:

assumes $f : f \in \text{ar } (\text{set-cat } U)$

shows $\text{dom } (\text{set-cat } U) f \in \text{ob } (\text{set-cat } U)$

and $\text{cod } (\text{set-cat } U) f \in \text{ob } (\text{set-cat } U)$

and $\text{set-cod } f \subseteq U$

and $\text{set-dom } f \subseteq U$

proof–

note [*simp*] = *set-cat-def set-arrow-def*

have $\text{dom } (\text{set-cat } U) f = \text{set-dom } f$ **using** f **by** *simp*

also show $\dots \subseteq U$ **using** f **by** *simp*

finally show $\text{dom } (\text{set-cat } U) f \in \text{ob } (\text{set-cat } U)$..

have $\text{cod } (\text{set-cat } U) f = \text{set-cod } f$ **using** f **by** *simp*

also show $\dots \subseteq U$ **using** f **by** *simp*

finally show $\text{cod } (\text{set-cat } U) f \in \text{ob } (\text{set-cat } U)$..

qed

In this context, $f \in \text{hom } A B$ is quite a strong claim.

lemma *set-homE* [*intro*]:

assumes $f : f \in \text{hom } (\text{set-cat } U) A B$

shows $A \subseteq U$

and $B \subseteq U$

and $\text{set-dom } f = A$

and $\text{set-func } f : A \rightarrow B$

and $\text{set-cod } f = B$

proof–

have $1 : f \in \text{ar } (\text{set-cat } U)$

```

    using f by (simp add: hom-def set-cat-def)
  show 2: set-dom f = A
    using f by (simp add: set-cat-def hom-def set-arrow-def)
  from 1 have set-dom f  $\subseteq$  U ..
  thus A  $\subseteq$  U by (simp add: 2)
  show 3: set-cod f = B
    using f by (simp add: set-cat-def hom-def set-arrow-def)
  from 1 have set-cod f  $\subseteq$  U ..
  thus B  $\subseteq$  U by (simp add: 3)
  have set-func f  $\in$  (set-dom f)  $\rightarrow$  (set-cod f)
    using f by (auto simp add: set-cat-def hom-def set-arrow-def)
  thus set-func f  $\in$  A  $\rightarrow$  B
    by (simp add: 2 3)
qed

```

2.3 Set is a Category

lemma *set-id-left*:

```

  assumes f: f  $\in$  ar (set-cat U)
  shows set-id U (set-cod f)  $\odot$  f = f
proof -
  from  $\langle f \in \text{ar } (\text{set-cat } U) \rangle$  have set-cod f  $\subseteq$  U ..
  hence 1: set-id U (set-cod f)  $\odot$  f =
    (
      set-dom=set-dom f,
      set-func=compose (set-dom f) ( $\lambda x \in \text{set-cod f. } x$ ) (set-func f),
      set-cod=set-cod f
    )
    using f by (simp add: set-comp-def set-id-def)
  have 2: compose (set-dom f) ( $\lambda x \in \text{set-cod f. } x$ ) (set-func f) = set-func f
  proof (rule extensionalityI)
    show compose (set-dom f) ( $\lambda x \in \text{set-cod f. } x$ ) (set-func f)  $\in$  extensional (set-dom
  f)
      by (rule compose-extensional)
    show set-func f  $\in$  extensional (set-dom f)
      using f by (simp add: set-cat-def set-arrow-def)
    fix x
    assume x-in-dom: x  $\in$  set-dom f
    have f-into-cod: set-func f : (set-dom f)  $\rightarrow$  (set-cod f)
      using f by (simp add: set-cat-def set-arrow-def)
    from f-into-cod and x-in-dom
    have f-x-in-cod: set-func f x  $\in$  set-cod f
      by (rule funcset-mem)
    show compose (set-dom f) ( $\lambda x \in \text{set-cod f. } x$ ) (set-func f) x = set-func f x
      by (simp add: x-in-dom f-x-in-cod compose-def)
  qed
  from 1 have set-id U (set-cod f)  $\odot$  f =
    (
      set-dom=set-dom f,

```

$set\text{-}func = set\text{-}func\ f$,
 $set\text{-}cod = set\text{-}cod\ f$
 \rangle
by (*simp only: 2*)
also have $\dots = f$
by *simp*
finally show *?thesis* .
qed

lemma *set-id-right*:

assumes $f: f \in ar\ (set\text{-}cat\ U)$
shows $f \odot (set\text{-}id\ U\ (set\text{-}dom\ f)) = f$

proof –

from $\langle f \in ar\ (set\text{-}cat\ U) \rangle$ **have** $set\text{-}dom\ f \subseteq U$..

hence $1: f \odot (set\text{-}id\ U\ (set\text{-}dom\ f)) =$

\langle
 $set\text{-}dom = set\text{-}dom\ f$,
 $set\text{-}func = compose\ (set\text{-}dom\ f)\ (set\text{-}func\ f)\ (\lambda x \in set\text{-}dom\ f. x)$,
 $set\text{-}cod = set\text{-}cod\ f$
 \rangle

using f **by** (*simp add: set-comp-def set-id-def*)

have $2: compose\ (set\text{-}dom\ f)\ (set\text{-}func\ f)\ (\lambda x \in set\text{-}dom\ f. x) = set\text{-}func\ f$

proof (*rule extensionalityI*)

show $compose\ (set\text{-}dom\ f)\ (set\text{-}func\ f)\ (\lambda x \in set\text{-}dom\ f. x) \in extensional\ (set\text{-}dom\ f)$

by (*rule compose-extensional*)

show $set\text{-}func\ f \in extensional\ (set\text{-}dom\ f)$

using f **by** (*simp add: set-cat-def set-arrow-def*)

fix x

assume $x\text{-in-dom}: x \in set\text{-}dom\ f$

thus $compose\ (set\text{-}dom\ f)\ (set\text{-}func\ f)\ (\lambda x \in set\text{-}dom\ f. x)\ x = set\text{-}func\ f\ x$

by (*simp add: compose-def*)

qed

from 1 **have** $f \odot (set\text{-}id\ U\ (set\text{-}dom\ f)) =$

\langle
 $set\text{-}dom = set\text{-}dom\ f$,
 $set\text{-}func = set\text{-}func\ f$,
 $set\text{-}cod = set\text{-}cod\ f$
 \rangle

by (*simp only: 2*)

also have $\dots = f$

by *simp*

finally show *?thesis* .

qed

lemma *set-id-hom*:

assumes $A \in ob\ (set\text{-}cat\ U)$

shows $id\ (set\text{-}cat\ U)\ A \in hom\ (set\text{-}cat\ U)\ A\ A$

proof –

from $\langle A \in \text{ob}(\text{set-cat } U) \rangle$ **have** $1: A \subseteq U$..
hence $\text{id}(\text{set-cat } U) A = (\setminus \text{set-dom} = A, \text{set-func} = \lambda x \in A. x, \text{set-cod} = A)$
by (*simp add: set-cat-def set-id-def*)
also have $\dots \in \text{hom}(\text{set-cat } U) A A$
proof (*rule set-homI*)
show $(\lambda x \in A. x) \in A \rightarrow A$
by (*rule funcsetI, auto*)
show $(\lambda x \in A. x) \in \text{extensional } A$
by (*rule restrict-extensional*)
qed (*rule 1, rule 1*)
finally show *?thesis* .
qed

lemma *set-comp-types*:

$\text{comp}(\text{set-cat } U) \in \text{hom}(\text{set-cat } U) B C \rightarrow \text{hom}(\text{set-cat } U) A B \rightarrow \text{hom}(\text{set-cat } U) A C$

proof (*rule funcsetI*)

fix g

assume $g\text{-}BC: g \in \text{hom}(\text{set-cat } U) B C$

hence $\text{comp-cod}: \text{set-cod } g = C$..

show $\text{comp}(\text{set-cat } U) g \in \text{hom}(\text{set-cat } U) A B \rightarrow \text{hom}(\text{set-cat } U) A C$

proof (*rule funcsetI*)

fix f

assume $f\text{-}AB: f \in \text{hom}(\text{set-cat } U) A B$

hence $\text{comp-dom}: \text{set-dom } f = A$..

show $\text{comp}(\text{set-cat } U) g f \in \text{hom}(\text{set-cat } U) A C$

proof–

have $\text{comp}(\text{set-cat } U) g f =$

(

$\text{set-dom} = A,$

$\text{set-func} = \text{compose}(\text{set-dom } f)(\text{set-func } g)(\text{set-func } f),$

$\text{set-cod} = C$

)

by (*simp add: set-cat-def set-comp-def comp-cod comp-dom*)

also have $\dots \in \text{hom}(\text{set-cat } U) A C$

proof (*rule set-homI*)

from $f\text{-}AB$ **show** $A \subseteq U$..

from $g\text{-}BC$ **show** $C \subseteq U$..

from $f\text{-}AB$ **have** $fs\text{-}f: \text{set-func } f: A \rightarrow B$..

from $g\text{-}BC$ **have** $fs\text{-}g: \text{set-func } g: B \rightarrow C$..

from $fs\text{-}g$ **and** $fs\text{-}f$

show $\text{compose}(\text{set-dom } f)(\text{set-func } g)(\text{set-func } f) : A \rightarrow C$

by (*simp only: comp-dom*) (*rule funcset-compose*)

show $\text{compose}(\text{set-dom } f)(\text{set-func } g)(\text{set-func } f) \in \text{extensional } A$

by (*simp only: comp-dom*) (*rule compose-extensional*)

qed

finally show *?thesis* .

qed

qed
qed

We reason explicitly about the function component of the composite arrow, leaving the rest to the simplifier.

lemma *set-comp-associative*:

fixes *f* **and** *g* **and** *h*

assumes *f*: $f \in ar (set-cat U)$

and *g*: $g \in ar (set-cat U)$

and *h*: $h \in ar (set-cat U)$

and *hg*: $cod (set-cat U) h = dom (set-cat U) g$

and *gf*: $cod (set-cat U) g = dom (set-cat U) f$

shows $comp (set-cat U) f (comp (set-cat U) g h) =$

$comp (set-cat U) (comp (set-cat U) f g) h$

proof (*simp add: set-cat-def set-comp-def*)

show $compose (set-dom h) (set-func f) (compose (set-dom h) (set-func g) (set-func h)) =$

$compose (set-dom h) (compose (set-dom g) (set-func f) (set-func g)) (set-func h)$

proof (*rule compose-assoc*)

have *1*: $set-cod h = set-dom g$ **using** *hg* **by** *simp*

have *2*: $set-cod g = set-dom f$ **using** *gf* **by** *simp*

show $set-func h \in set-dom h \rightarrow set-dom g$

using *h* **by** (*simp add: set-cat-def set-arrow-def 1*)

show $set-func g \in set-dom g \rightarrow set-dom f$

using *g* **by** (*simp add: set-cat-def set-arrow-def 2*)

show $set-func f \in set-dom f \rightarrow set-cod f$

using *f* **by** (*simp add: set-cat-def set-arrow-def*)

qed

qed

theorem *set-cat-cat*: *category (set-cat U)*

proof (*rule category.intro*)

fix *f*

assume *f*: $f \in ar (set-cat U)$

show $dom (set-cat U) f \in ob (set-cat U)$ **using** *f* **..**

show $cod (set-cat U) f \in ob (set-cat U)$ **using** *f* **..**

show $comp (set-cat U) (id (set-cat U) (cod (set-cat U) f)) f = f$

using *f* **by** (*simp add: set-id-left*)

show $comp (set-cat U) f (id (set-cat U) (dom (set-cat U) f)) = f$

using *f* **by** (*simp add: set-id-right*)

next

fix *A*

assume *A* $\in ob (set-cat U)$

then show $id (set-cat U) A \in hom (set-cat U) A A$

by (*rule set-id-hom*)

next

fix *A* **and** *B* **and** *C*

```

show  $\text{comp } (\text{set-cat } U) \in \text{hom } (\text{set-cat } U) B C \rightarrow \text{hom } (\text{set-cat } U) A B \rightarrow \text{hom } (\text{set-cat } U) A C$ 
  by (rule set-comp-types)
next
  fix  $f$  and  $g$  and  $h$ 
  assume  $f \in \text{ar } (\text{set-cat } U)$ 
    and  $g \in \text{ar } (\text{set-cat } U)$ 
    and  $h \in \text{ar } (\text{set-cat } U)$ 
    and  $\text{cod } (\text{set-cat } U) h = \text{dom } (\text{set-cat } U) g$ 
    and  $\text{cod } (\text{set-cat } U) g = \text{dom } (\text{set-cat } U) f$ 
  then show  $\text{comp } (\text{set-cat } U) f (\text{comp } (\text{set-cat } U) g h) = \text{comp } (\text{set-cat } U) (\text{comp } (\text{set-cat } U) f g) h$ 
    by (rule set-comp-associative)
qed

end

```

3 Functors

```

theory Functors
imports Cat
begin

```

3.1 Definitions

```

record  $(\text{'o1}, \text{'a1}, \text{'o2}, \text{'a2})$  functor =
   $om :: \text{'o1} \Rightarrow \text{'o2}$ 
   $am :: \text{'a1} \Rightarrow \text{'a2}$ 

```

abbreviation

```

 $om\text{-syn } (- \circ [81])$  where
 $F_\circ \equiv om F$ 

```

abbreviation

```

 $am\text{-syn } (- \text{a} [81])$  where
 $F_{\text{a}} \equiv am F$ 

```

```

locale two-cats =  $AA$ : category  $AA$  +  $BB$ : category  $BB$ 
  for  $AA$  (structure) and  $BB$  (structure) +
  assumes  $AA = (AA :: (\text{'o1}, \text{'a1}, \text{'m1}) \text{category-scheme})$ 
  assumes  $BB = (BB :: (\text{'o2}, \text{'a2}, \text{'m2}) \text{category-scheme})$ 
  fixes  $\text{preserves-dom} :: (\text{'o1}, \text{'a1}, \text{'o2}, \text{'a2}) \text{functor} \Rightarrow \text{bool}$ 
  and  $\text{preserves-cod} :: (\text{'o1}, \text{'a1}, \text{'o2}, \text{'a2}) \text{functor} \Rightarrow \text{bool}$ 
  and  $\text{preserves-id} :: (\text{'o1}, \text{'a1}, \text{'o2}, \text{'a2}) \text{functor} \Rightarrow \text{bool}$ 
  and  $\text{preserves-comp} :: (\text{'o1}, \text{'a1}, \text{'o2}, \text{'a2}) \text{functor} \Rightarrow \text{bool}$ 
  defines  $\text{preserves-dom } G \equiv$ 
 $\forall f \in Ar_1. G_\circ (\text{Dom}_1 f) = \text{Dom}_2 (G_{\text{a}} f)$ 
  and  $\text{preserves-cod } G \equiv$ 

```

$\forall f \in Ar_1. G_o (Cod_1 f) = Cod_2 (G_a f)$
and *preserves-id* $G \equiv$
 $\forall A \in Ob_1. G_a (Id_1 A) = Id_2 (G_o A)$
and *preserves-comp* $G \equiv$
 $\forall f \in Ar_1. \forall g \in Ar_1. Cod_1 f = Dom_1 g \longrightarrow G_a (g \cdot_1 f) = (G_a g) \cdot_2 (G_a f)$

locale *functor* = *two-cats* +

fixes F (**structure**)

assumes *F-preserves-arrows*: $F_a : Ar_1 \rightarrow Ar_2$

and *F-preserves-objects*: $F_o : Ob_1 \rightarrow Ob_2$

and *F-preserves-dom*: *preserves-dom* F

and *F-preserves-cod*: *preserves-cod* F

and *F-preserves-id*: *preserves-id* F

and *F-preserves-comp*: *preserves-comp* F

notes *F-axioms* = *F-preserves-arrows* *F-preserves-objects* *F-preserves-dom*

F-preserves-cod *F-preserves-id* *F-preserves-comp*

notes *func-pred-defs* = *preserves-dom-def* *preserves-cod-def* *preserves-id-def* *preserves-comp-def*

This gives us nicer notation for asserting that things are functors.

abbreviation

Functor (*Functor* - : - \longrightarrow - [81]) **where**

Functor $F : AA \longrightarrow BB \equiv$ *functor* AA BB F

3.2 Simple Lemmas

For example:

lemma (**in** *functor*) *Functor* $F : AA \longrightarrow BB$..

lemma *functors-preserve-arrows* [*intro*]:

assumes *Functor* $F : AA \longrightarrow BB$

and $f \in ar$ AA

shows $F_a f \in ar$ BB

proof –

from \langle *Functor* $F : AA \longrightarrow BB$ \rangle

have $F_a : ar$ $AA \rightarrow ar$ BB

by (*simp add: functor-def functor-axioms-def*)

from *this* **and** \langle $f \in ar$ AA \rangle

show *?thesis* **by** (*rule funcset-mem*)

qed

lemma (**in** *functor*) *functors-preserve-homsets*:

assumes $1: A \in Ob_1$

and $2: B \in Ob_1$

and $3: f \in Hom_1 A B$

shows $F_a f \in Hom_2 (F_o A) (F_o B)$

proof –

from 3

have $4: f \in Ar$
by (*simp add: hom-def*)
with *F-preserves-arrows*
have $5: F_a f \in Ar_2$
by (*rule funcset-mem*)
from 4 **and** *F-preserves-dom*
have $Dom_2 (F_a f) = F_o (Dom_1 f)$
by (*simp add: preserves-dom-def*)
also from 3 **have** $\dots = F_o A$
by (*simp add: hom-def*)
finally have $6: Dom_2 (F_a f) = F_o A .$
from 4 **and** *F-preserves-cod*
have $Cod_2 (F_a f) = F_o (Cod_1 f)$
by (*simp add: preserves-cod-def*)
also from 3 **have** $\dots = F_o B$
by (*simp add: hom-def*)
finally have $7: Cod_2 (F_a f) = F_o B .$
from 5 **and** 6 **and** 7
show *?thesis*
by (*simp add: hom-def*)
qed

lemma *functors-preserve-objects* [*intro*]:
assumes *Functor F : AA \longrightarrow BB*
and $A \in ob\ AA$
shows $F_o A \in ob\ BB$
proof –
from $\langle Functor\ F : AA \longrightarrow BB \rangle$
have $F_o : ob\ AA \rightarrow ob\ BB$
by (*simp add: functor-def functor-axioms-def*)
from *this* **and** $\langle A \in ob\ AA \rangle$
show *?thesis* **by** (*rule funcset-mem*)
qed

3.3 Identity Functor

definition

id-func :: $('o, 'a, 'm)$ *category-scheme* \Rightarrow $('o, 'a, 'o, 'a)$ *functor* **where**
id-func $CC = (\!| om = (\lambda A \in ob\ CC. A), am = (\lambda f \in ar\ CC. f) \!|)$

locale *one-cat* = *two-cats* +
assumes *endo*: $BB = AA$

lemma (**in** *one-cat*) *id-func-preserves-arrows*:
shows $(id-func\ AA)_a : Ar \rightarrow Ar$
by (*unfold id-func-def, rule funcsetI, simp*)

lemma (in *one-cat*) *id-func-preserves-objects*:
 shows $(id\text{-func } AA)_o : Ob \rightarrow Ob$
 by (unfold *id-func-def*, rule *funcsetI*, *simp*)

lemma (in *one-cat*) *id-func-preserves-dom*:
 shows *preserves-dom* (*id-func AA*)
 unfolding *preserves-dom-def* *endo*
proof
 fix *f*
 assume $f : f \in Ar$
 hence *lhs*: $(id\text{-func } AA)_o (Dom f) = Dom f$
 by (*simp add: id-func-def*) *auto*
 have $(id\text{-func } AA)_a f = f$
 using *f* by (*simp add: id-func-def*)
 hence *rhs*: $Dom (id\text{-func } AA)_a f = Dom f$
 by *simp*
 from *lhs* and *rhs* show $(id\text{-func } AA)_o (Dom f) = Dom (id\text{-func } AA)_a f$
 by *simp*
qed

lemma (in *one-cat*) *id-func-preserves-cod*:
preserves-cod (*id-func AA*)
 apply (unfold *preserves-cod-def*, *simp only: endo*)
proof
 fix *f*
 assume $f : f \in Ar$
 hence *lhs*: $(id\text{-func } AA)_o (Cod f) = Cod f$
 by (*simp add: id-func-def*) *auto*
 have $(id\text{-func } AA)_a f = f$
 using *f* by (*simp add: id-func-def*)
 hence *rhs*: $Cod (id\text{-func } AA)_a f = Cod f$
 by *simp*
 from *lhs* and *rhs* show $(id\text{-func } AA)_o (Cod f) = Cod (id\text{-func } AA)_a f$
 by *simp*
qed

lemma (in *one-cat*) *id-func-preserves-id*:
preserves-id (*id-func AA*)
 unfolding *preserves-id-def* *endo*
proof
 fix *A*
 assume $A : A \in Ob$
 hence *lhs*: $(id\text{-func } AA)_a (Id A) = Id A$
 by (*simp add: id-func-def*) *auto*
 have $(id\text{-func } AA)_o A = A$
 using *A* by (*simp add: id-func-def*)
 hence *rhs*: $Id ((id\text{-func } AA)_o A) = Id A$

```

    by simp
  from lhs and rhs show (id-func AA)a (Id A) = Id ((id-func AA)o A)
    by simp
qed

```

```

lemma (in one-cat) id-func-preserves-comp:
  preserves-comp (id-func AA)
unfolding preserves-comp-def endo
proof (intro ballI impI)
  fix f and g
  assume f: f ∈ Ar and g: g ∈ Ar and Cod f = Dom g
  then have g · f ∈ Ar ..
  hence lhs: (id-func AA)a (g · f) = g · f
    by (simp add: id-func-def)
  have id-f: (id-func AA)a f = f
    using f by (simp add: id-func-def)
  have id-g: (id-func AA)a g = g
    using g by (simp add: id-func-def)
  hence rhs: (id-func AA)a g · (id-func AA)a f = g · f
    by (simp add: id-f id-g)
  from lhs and rhs
  show (id-func AA)a (g · f) = (id-func AA)a g · (id-func AA)a f
    by simp
qed

```

```

theorem (in one-cat) id-func-functor:
  Functor (id-func AA) : AA → AA
proof -
  from id-func-preserves-arrows
  and id-func-preserves-objects
  and id-func-preserves-dom
  and id-func-preserves-cod
  and id-func-preserves-id
  and id-func-preserves-comp
  show ?thesis
    by unfold-locales (simp-all add: endo preserves-dom-def
      preserves-cod-def preserves-id-def preserves-comp-def)
qed

```

end

4 HomFunctors

```

theory HomFunctors
imports SetCat Functors
begin

```

```

locale into-set = two-cats +
  assumes AA = (AA::('o,'a,'m)category-scheme)
  fixes U and Set
  defines U ≡ (UNIV::'a set)
  defines Set ≡ set-cat U
  assumes BB-Set: BB = Set
  fixes homf (Hom'(-,'-'))
  defines homf A ≡ (
    om = (λB∈Ob. Hom A B),
    am = (λf∈Ar. (λset-dom=Hom A (Dom f),set-func=(λg∈Hom A (Dom f). f ·
g),set-cod=Hom A (Cod f)))
  )

```

lemma (in into-set) homf-preserves-arrows:

```

  Hom(A,-)a : Ar → ar Set
proof (rule funcsetI)
  fix f
  assume f: f ∈ Ar
  thus Hom(A,-)a f ∈ ar Set
proof (simp add: homf-def Set-def set-cat-def set-arrow-def U-def)
  have 1: (op ·) : Hom (Dom f) (Cod f) → Hom A (Dom f) → Hom A (Cod f)
  ..
  have 2: f ∈ Hom (Dom f) (Cod f) using f by (simp add: hom-def)
  from 1 and 2 have 3: (op ·) f : Hom A (Dom f) → Hom A (Cod f)
  by (rule funcset-mem)
  show (λg∈Hom A (Dom f). f · g) : Hom A (Dom f) → Hom A (Cod f)
proof (rule funcsetI)
  fix g'
  assume g' ∈ Hom A (Dom f)
  from 3 and this show (λg∈Hom A (Dom f). f · g) g' ∈ Hom A (Cod f)
  by simp (rule funcset-mem)
  qed
qed
qed

```

lemma (in into-set) homf-preserves-objects:

```

  Hom(A,-)o : Ob → ob Set
proof (rule funcsetI)
  fix B
  assume B: B ∈ Ob
  have Hom(A,-)o B = Hom A B
  using B by (simp add: homf-def)
  moreover have ... ∈ ob Set
  by (simp add: U-def Set-def set-cat-def)
  ultimately show Hom(A,-)o B ∈ ob Set by simp
qed

```

lemma (in *into-set*) *homf-preserves-dom*:
 assumes $f: f \in Ar$
 shows $Hom(A,-)_o (Dom f) = dom Set (Hom(A,-)_a f)$
proof –
 have $Dom f \in Ob$ using f ..
 hence 1: $Hom(A,-)_o (Dom f) = Hom A (Dom f)$
 using f by (simp add: homf-def)
 have 2: $dom Set (Hom(A,-)_a f) = Hom A (Dom f)$
 using f by (simp add: Set-def homf-def)
 from 1 and 2 show ?thesis by simp
qed

lemma (in *into-set*) *homf-preserves-cod*:
 assumes $f: f \in Ar$
 shows $Hom(A,-)_o (Cod f) = cod Set (Hom(A,-)_a f)$
proof –
 have $Cod f \in Ob$ using f ..
 hence 1: $Hom(A,-)_o (Cod f) = Hom A (Cod f)$
 using f by (simp add: homf-def)
 have 2: $cod Set (Hom(A,-)_a f) = Hom A (Cod f)$
 using f by (simp add: Set-def homf-def)
 from 1 and 2 show ?thesis by simp
qed

lemma (in *into-set*) *homf-preserves-id*:
 assumes $B: B \in Ob$
 shows $Hom(A,-)_a (Id B) = id Set (Hom(A,-)_o B)$
proof –
 have 1: $Id B \in Ar$ using B ..
 have 2: $Dom (Id B) = B$
 using B by (rule AA.id-dom-cod)
 have 3: $Cod (Id B) = B$
 using B by (rule AA.id-dom-cod)
 have 4: $(\lambda g \in Hom A B. (Id B) \cdot g) = (\lambda g \in Hom A B. g)$
 by (rule ext) (auto simp add: hom-def)
 have $Hom(A,-)_a (Id B) = \{\}$
 $set-dom = Hom A B,$
 $set-func = (\lambda g \in Hom A B. g),$
 $set-cod = Hom A B\}$
 by (simp add: homf-def 1 2 3 4)
 also have $\dots = id Set (Hom(A,-)_o B)$
 using B by (simp add: Set-def U-def set-cat-def set-id-def homf-def)
 finally show ?thesis .
qed

lemma (in *into-set*) *homf-preserves-comp*:

assumes $f: f \in Ar$
and $g: g \in Ar$
and $fg: Cod f = Dom g$
shows $Hom(A, -)_a (g \cdot f) = (Hom(A, -)_a g) \odot (Hom(A, -)_a f)$
proof –
have 1: $g \cdot f \in Ar$ **using** *assms ..*
have 2: $Dom (g \cdot f) = Dom f$ **using** *f g fg ..*
have 3: $Cod (g \cdot f) = Cod g$ **using** *f g fg ..*
have *lhs: Hom(A, -)_a (g \cdot f) =* ($\{$
set-dom=Hom A (Dom f),
set-func=($\lambda h \in Hom A (Dom f). (g \cdot f) \cdot h$),
set-cod=Hom A (Cod g) $\}$)
by (*simp add: homf-def 1 2 3*)
have 4: $set-dom ((Hom(A, -)_a g) \odot (Hom(A, -)_a f)) = Hom A (Dom f)$
using *f by (simp add: set-comp-def homf-def)*
have 5: $set-cod ((Hom(A, -)_a g) \odot (Hom(A, -)_a f)) = Hom A (Cod g)$
using *g by (simp add: set-comp-def homf-def)*
have *set-func ((Hom(A, -)_a g) \odot (Hom(A, -)_a f))*
 $= compose (Hom A (Dom f)) (\lambda y \in Hom A (Dom g). g \cdot y) (\lambda x \in Hom A (Dom$
 $f). f \cdot x)$
using *f g by (simp add: set-comp-def homf-def)*
also have $\dots = (\lambda h \in Hom A (Dom f). (g \cdot f) \cdot h)$
proof (
rule extensionalityI,
rule compose-extensional,
rule restrict-extensional,
simp)
fix *h*
assume 10: $h \in Hom A (Dom f)$
hence 11: $f \cdot h \in Hom A (Dom g)$
proof –
from 10 **have** $h \in Ar$ **by** (*simp add: hom-def*)
have 100: $(op \cdot) : Hom (Dom f) (Dom g) \rightarrow Hom A (Dom f) \rightarrow Hom A$
 $(Dom g)$
by (*rule AA.comp-types*)
have $f \in Hom (Dom f) (Cod f)$ **using** *f by (simp add: hom-def)*
hence 101: $f \in Hom (Dom f) (Dom g)$ **using** *fg by simp*
from 100 **and** 101
have $(op \cdot) f : Hom A (Dom f) \rightarrow Hom A (Dom g)$
by (*rule funcset-mem*)
from *this* **and** 10
show $f \cdot h \in Hom A (Dom g)$
by (*rule funcset-mem*)
qed
hence $Cod (f \cdot h) = Dom g$
and $Dom (f \cdot h) = A$
and $f \cdot h \in Ar$
by (*simp-all add: hom-def*)
thus $compose (Hom A (Dom f)) (\lambda y \in Hom A (Dom g). g \cdot y) (\lambda x \in Hom A$

```

(Dom f). f · x) h =
  (g · f) · h
  using f g fg 10 by (simp add: compose-def 10 11 hom-def)
qed
finally have 6: set-func ((Hom(A,-)a g) ∘ (Hom(A,-)a f))
  = (λh∈Hom A (Dom f). (g · f) · h) .
from 4 and 5 and 6
have rhs: (Hom(A,-)a g) ∘ (Hom(A,-)a f) = (|
  set-dom=Hom A (Dom f),
  set-func=(λh∈Hom A (Dom f). (g · f) · h),
  set-cod=Hom A (Cod g)|)
by simp
show ?thesis
by (simp add: lhs rhs)
qed

```

theorem (in into-set) homf-into-set:

Functor Hom(A,-) : AA → Set

proof (intro functor.intro functor-axioms.intro)

show Hom(A,-)_a : Ar → ar Set

by (rule homf-preserves-arrows)

show Hom(A,-)_o : Ob → ob Set

by (rule homf-preserves-objects)

show ∀f∈Ar. Hom(A,-)_o (Dom f) = dom Set (Hom(A,-)_a f)

by (intro ballI) (rule homf-preserves-dom)

show ∀f∈Ar. Hom(A,-)_o (Cod f) = cod Set (Hom(A,-)_a f)

by (intro ballI) (rule homf-preserves-cod)

show ∀B∈Ob. Hom(A,-)_a (Id B) = id Set (Hom(A,-)_o B)

by (intro ballI) (rule homf-preserves-id)

show ∀f∈Ar. ∀g∈Ar.

Cod f = Dom g →

Hom(A,-)_a (g · f) = comp Set (Hom(A,-)_a g) (Hom(A,-)_a f)

by (intro ballI impI, simp add: Set-def set-cat-def) (rule homf-preserves-comp)

show two-cats AA Set

proof intro-locales

show category Set

by (unfold Set-def, rule set-cat-cat)

show two-cats-axioms AA Set

proof qed rule+

qed

qed

end

5 Natural Transformations

theory NatTrans

imports *Functors*
begin

locale *natural-transformation* = *two-cats* +
fixes *F* **and** *G* **and** *u*
assumes *Functor* *F* : *AA* \longrightarrow *BB*
and *Functor* *G* : *AA* \longrightarrow *BB*
and *u* : *ob* *AA* \rightarrow *ar* *BB*
and *u* \in *extensional* (*ob* *AA*)
and $\forall A \in \text{Ob. } u\ A \in \text{Hom}_2\ (F_o\ A)\ (G_o\ A)$
and $\forall A \in \text{Ob. } \forall B \in \text{Ob. } \forall f \in \text{Hom}\ A\ B. (G_a\ f) \cdot_2\ (u\ A) = (u\ B) \cdot_2\ (F_a\ f)$

abbreviation

nt-syn ($- : - \Rightarrow -$ in *Func* '($- , -$) [*81*]) **where**
 $u : F \Rightarrow G$ in *Func*(*AA*, *BB*) \equiv *natural-transformation* *AA* *BB* *F* *G* *u*

locale *endoNT* = *natural-transformation* + *one-cat*

theorem (**in** *endoNT*) *id-restrict-natural*:

$(\lambda A \in \text{Ob. } \text{Id}\ A) : (\text{id-func}\ AA) \Rightarrow (\text{id-func}\ AA)$ in *Func*(*AA*, *AA*)

proof (*intro* *natural-transformation.intro* *natural-transformation-axioms.intro*

two-cats.intro *two-cats-axioms.intro* *ballI*)

show $(\lambda A \in \text{Ob. } \text{Id}\ A) : \text{Ob} \rightarrow \text{Ar}$

by (*rule* *funcsetI*) *auto*

show $(\lambda A \in \text{Ob. } \text{Id}\ A) \in \text{extensional}\ (\text{Ob})$

by (*rule* *restrict-extensional*)

fix *A*

assume *A*: $A \in \text{Ob}$

hence $\text{Id}\ A \in \text{Hom}\ A\ A$..

thus $(\lambda X \in \text{Ob. } \text{Id}\ X) A \in \text{Hom}\ ((\text{id-func}\ AA)_o\ A)\ ((\text{id-func}\ AA)_o\ A)$

using *A* **by** (*simp* *add*: *id-func-def*)

fix *B* **and** *f*

assume *B*: $B \in \text{Ob}$

and $f \in \text{Hom}\ A\ B$

hence $f \in \text{Ar}$ **and** $A = \text{Dom}\ f$ **and** $B = \text{Cod}\ f$ **and** $\text{Dom}\ f \in \text{Ob}$ **and** $\text{Cod}\ f \in \text{Ob}$

using *A* **by** (*simp*-*all* *add*: *hom-def*)

thus $(\text{id-func}\ AA)_a\ f \cdot (\lambda A \in \text{Ob. } \text{Id}\ A) A$

$= (\lambda A \in \text{Ob. } \text{Id}\ A) B \cdot (\text{id-func}\ AA)_a\ f$

by (*simp* *add*: *id-func-def*)

qed (*auto* *intro*: *id-func-functor*, *unfold-locales*, *unfold-locales*)

end

6 Yoneda Lemma

```
theory Yoneda
imports HomFunctors NatTrans
begin
```

6.1 The Sandwich Natural Transformation

```
locale Yoneda = functor + into-set +
  assumes AA = (AA::('o,'a,'m)category-scheme)
  fixes sandwich :: ['o,'a,'o] ⇒ 'a set-arrow (σ'(-,-))
  defines sandwich A a ≡ (λB∈Ob. (
    set-dom=Hom A B,
    set-func=(λf∈Hom A B. set-func (Fa f) a),
    set-cod=Fo B
  ))
  fixes unsandwich :: ['o,'o ⇒ 'a set-arrow] ⇒ 'a (σ-'(-,-))
  defines unsandwich A u ≡ set-func (u A) (Id A)
```

lemma (in Yoneda) *F-into-set*:

Functor F : AA → Set

proof –

from *F-axioms* **have** *Functor F : AA → BB* **by** *intro-locales*

thus *?thesis*

by (*simp only: BB-Set*)

qed

lemma (in Yoneda) *F-comp-func*:

assumes *1: A ∈ Ob and 2: B ∈ Ob and 3: C ∈ Ob*

and *4: g ∈ Hom A B and 5: f ∈ Hom B C*

shows *set-func (F_a (f · g)) = compose (F_o A) (set-func (F_a f)) (set-func (F_a g))*

proof –

from *4 and 5*

have *7: Cod g = Dom f*

and *8: g ∈ Ar*

and *9: f ∈ Ar*

and *10: Dom g = A*

by (*simp-all add: hom-def*)

from *F-preserves-dom and 8 and 10*

have *11: set-dom (F_a g) = F_o A*

by (*simp add: preserves-dom-def BB-Set Set-def*) *auto*

from *F-preserves-comp and 7 and 8 and 9*

have *F_a (f · g) = (F_a f) ·₂ (F_a g)*

by (*simp add: preserves-comp-def*)

hence *set-func (F_a (f · g)) = set-func ((F_a f) ◦ (F_a g))*

by (*simp add: BB-Set Set-def*)

also have *... = compose (F_o A) (set-func (F_a f)) (set-func (F_a g))*

by (*simp add: set-comp-def 11*)

finally show *?thesis* .
 qed

lemma (in *Yoneda*) *sandwich-funcset*:

assumes $A: A \in Ob$

and $a \in F_o A$

shows $\sigma(A,a) : Ob \rightarrow ar Set$

proof (rule *funcsetI*)

fix B

assume $B: B \in Ob$

thus $\sigma(A,a) B \in ar Set$

proof (simp add: *Set-def sandwich-def set-cat-def*)

show *set-arrow* $U \{$

set-dom = $Hom A B$,

set-func = $\lambda f \in Hom A B. set-func (F_a f) a$,

set-cod = $F_o B$)

proof (simp add: *set-arrow-def, intro conjI*)

show $Hom A B \subseteq U$ and $F_o B \subseteq U$

by (simp-all add: *U-def*)

show $(\lambda f \in Hom A B. set-func (F_a f) a) \in Hom A B \rightarrow F_o B$

proof (rule *funcsetI, simp*)

fix f

assume $f: f \in Hom A B$

with $A B$ have $F_a f \in Hom_2 (F_o A) (F_o B)$

by (rule *functors-preserve-homsets*)

hence $F_a f \in ar Set$

and *set-dom* $(F_a f) = (F_o A)$

and *set-cod* $(F_a f) = (F_o B)$

by (simp-all add: *hom-def BB-Set Set-def*)

hence *set-func* $(F_a f) : (F_o A) \rightarrow (F_o B)$

by (simp add: *Set-def set-cat-def set-arrow-def*)

thus *set-func* $(F_a f) a \in F_o B$

using $\langle a \in F_o A \rangle$

by (rule *funcset-mem*)

qed

qed

qed

qed

lemma (in *Yoneda*) *sandwich-type*:

assumes $A: A \in Ob$ and $B: B \in Ob$

and $a \in F_o A$

shows $\sigma(A,a) B \in hom Set (Hom A B) (F_o B)$

proof –

have $\sigma(A,a) \in Ob \rightarrow Ar Set$

using A and $\langle a \in F_o A \rangle$ by (rule *sandwich-funcset*)

hence $\sigma(A,a) B \in ar Set$

using B by (rule *funcset-mem*)

thus *?thesis*
 using B by (*simp add: sandwich-def hom-def Set-def*)
 qed

lemma (in *Yoneda*) *sandwich-commutes*:

assumes $AOb: A \in Ob$ and $BOb: B \in Ob$ and $COB: C \in Ob$
 and $aFa: a \in F_o A$
 and $fBC: f \in Hom B C$
 shows $(F_a f) \odot (\sigma(A,a) B) = (\sigma(A,a) C) \odot (Hom(A,-)_a f)$

proof –

from fBC have 1: $f \in Ar$ and 2: $Dom f = B$ and 3: $Cod f = C$
 by (*simp-all add: hom-def*)

from BOb have *set-dom* $((F_a f) \odot (\sigma(A,a) B)) = Hom A B$
 by (*simp add: set-comp-def sandwich-def*)

also have $\dots = set-dom ((\sigma(A,a) C) \odot (Hom(A,-)_a f))$
 by (*simp add: set-comp-def homf-def 1 2*)

finally have *set-dom-eq*:

$set-dom ((F_a f) \odot (\sigma(A,a) B))$
 $= set-dom ((\sigma(A,a) C) \odot (Hom(A,-)_a f)) .$

from $BOb COB fBC$ have $(F_a f) \in Hom_2 (F_o B) (F_o C)$
 by (*rule functors-preserve-homsets*)

hence *set-cod* $((F_a f) \odot (\sigma(A,a) B)) = F_o C$

by (*simp add: set-comp-def BB-Set Set-def set-cat-def hom-def*)

also from COB

have $\dots = set-cod ((\sigma(A,a) C) \odot (Hom(A,-)_a f))$

by (*simp add: set-comp-def sandwich-def*)

finally have *set-cod-eq*:

$set-cod ((F_a f) \odot (\sigma(A,a) B))$
 $= set-cod ((\sigma(A,a) C) \odot (Hom(A,-)_a f)) .$

from AOb and BOb and COB and fBC and aFa

have *set-func-lhs*:

$set-func ((F_a f) \odot (\sigma(A,a) B)) =$
 $(\lambda g \in Hom A B. set-func (F_a (f \cdot g)) a)$

apply (*simp add: set-comp-def sandwich-def compose-def*)

apply (*rule extensionalityI, rule restrict-extensional, rule restrict-extensional*)

by (*simp add: F-comp-func compose-def*)

have $(op \cdot) : Hom B C \rightarrow Hom A B \rightarrow Hom A C ..$

from *this* and fBC

have *opfType*: $(op \cdot) f : Hom A B \rightarrow Hom A C$

by (*rule funcset-mem*)

from 1 and 2

have *set-func* $((\sigma(A,a) C) \odot (Hom(A,-)_a f)) =$

$(\lambda g \in Hom A B. set-func (\sigma(A,a) C) (f \cdot g))$

apply (*simp add: set-comp-def homf-def*)

apply (*simp add: compose-def*)

apply (*rule extensionalityI, rule restrict-extensional, rule restrict-extensional*)

by *auto*

also from COB and *opfType*

```

have ... = ( $\lambda g \in \text{Hom } A \ B. \text{ set-func } (F \text{ }_a (f \cdot g)) \ a$ )
  apply (simp add: sandwich-def)
  apply (rule extensionalityI, rule restrict-extensional, rule restrict-extensional)
  by (simp add:Pi-def)
finally have set-func-rhs:
  set-func (( $\sigma(A,a) \ C$ )  $\odot$  (Hom( $A,-$ )  $_a f$ )) =
  ( $\lambda g \in \text{Hom } A \ B. \text{ set-func } (F \text{ }_a (f \cdot g)) \ a$ ) .
from set-func-lhs and set-func-rhs have
  set-func (( $F \text{ }_a f$ )  $\odot$  ( $\sigma(A,a) \ B$ ))
  = set-func (( $\sigma(A,a) \ C$ )  $\odot$  (Hom( $A,-$ )  $_a f$ ))
  by simp
with set-dom-eq and set-cod-eq show ?thesis
  by simp
qed

```

```

lemma (in Yoneda) sandwich-natural:
  assumes  $A \in \text{Ob}$ 
  and  $a \in F \circ A$ 
  shows  $\sigma(A,a) : \text{Hom}(A,-) \Rightarrow F \text{ in } \text{Func}(AA, \text{Set})$ 
proof (intro natural-transformation.intro natural-transformation-axioms.intro two-cats.intro)
  show category AA ..
  show category Set
  by (simp only: Set-def)(rule set-cat-cat)
  show Functor Hom(A,-) : AA  $\longrightarrow$  Set
  by (rule homf-into-set)
  show Functor F : AA  $\longrightarrow$  Set
  by (rule F-into-set)
  show  $\forall B \in \text{Ob}. \sigma(A,a) \ B \in \text{hom } \text{Set} \ (\text{Hom}(A,-) \circ B) \ (F \circ B)$ 
  using assms by (auto simp add: homf-def intro: sandwich-type)
  show  $\sigma(A,a) : \text{Ob} \rightarrow \text{ar } \text{Set}$ 
  using assms by (rule sandwich-funcset)
  show  $\sigma(A,a) \in \text{extensional } (\text{Ob})$ 
  unfolding sandwich-def by (rule restrict-extensional)
  show  $\forall B \in \text{Ob}. \forall C \in \text{Ob}. \forall f \in \text{Hom } B \ C.$ 
  comp Set ( $F \text{ }_a f$ ) ( $\sigma(A,a) \ B$ ) = comp Set ( $\sigma(A,a) \ C$ ) (Hom( $A,-$ )  $_a f$ )
  using assms by (auto simp add: Set-def intro: sandwich-commutes)
qed (intro two-cats-axioms.intro, simp-all)

```

6.2 Sandwich Components are Bijective

```

lemma (in Yoneda) unsandwich-left-inverse:
  assumes  $1: A \in \text{Ob}$ 
  and  $2: a \in F \circ A$ 
  shows  $\sigma^{\leftarrow}(A, \sigma(A,a)) = a$ 
proof -
  from  $1$  have  $\text{Id } A \in \text{Hom } A \ A \ ..$ 
  with  $1$ 
  have  $3: \sigma^{\leftarrow}(A, \sigma(A,a)) = \text{set-func } (F \text{ }_a (\text{Id } A)) \ a$ 

```

by (*simp add: sandwich-def homf-def unsandwich-def*)
from *F-preserves-id and 1*
have $4: F_a (Id A) = id Set (F_o A)$
 by (*simp add: preserves-id-def BB-Set*)
from *F-preserves-objects and 1*
have $F_o A \in Ob_2$
 by (*rule funcset-mem*)
hence $F_o A \subseteq U$
 by (*simp add: BB-Set Set-def set-cat-def*)
with 2
have $5: set-func (id Set (F_o A)) a = a$
 by (*simp add: Set-def set-id-def*)
show *?thesis*
 by (*simp add: 3 4 5*)
qed

lemma (*in Yoneda*) *unsandwich-right-inverse:*

assumes $1: A \in Ob$
and $2: u : Hom(A, -) \Rightarrow F$ *in* *Func(AA, Set)*
shows $\sigma(A, \sigma^{\leftarrow}(A, u)) = u$
proof (*rule extensionalityI*)
show $\sigma(A, \sigma^{\leftarrow}(A, u)) \in extensional (Ob)$
 by (*unfold sandwich-def, rule restrict-extensional*)
from 2 **show** $u \in extensional (Ob)$
 by (*simp add: natural-transformation-def natural-transformation-axioms-def*)
fix B
assume $3: B \in Ob$
with 1
have *one:* $\sigma(A, \sigma^{\leftarrow}(A, u)) B = ()$
 $set-dom = Hom A B,$
 $set-func = (\lambda f \in Hom A B. (set-func (F_a f)) (set-func (u A) (Id A))),$
 $set-cod = F_o B ()$
 by (*simp add: sandwich-def unsandwich-def*)
from 1 **have** $Hom(A, -)_o A = Hom A A$
 by (*simp add: homf-def*)
with 1 **and** 2 **have** $(u A) \in hom Set (Hom A A) (F_o A)$
 by (*simp add: natural-transformation-def natural-transformation-axioms-def, auto*)
hence $set-dom (u A) = Hom A A$
 by (*simp add: hom-def Set-def*)
with 1 **have** *applicable:* $Id A \in set-dom (u A)$
 by (*simp*)(*rule*)
have *two:* $(\lambda f \in Hom A B. (set-func (F_a f)) (set-func (u A) (Id A)))$
 $= (\lambda f \in Hom A B. (set-func ((F_a f) \odot (u A)) (Id A)))$
 by (*rule extensionalityI,*
rule restrict-extensional, rule restrict-extensional,
simp add: set-comp-def compose-def applicable)
from 2

have $(\forall X \in Ob. \forall Y \in Ob. \forall f \in Hom\ X\ Y. (F_a\ f) \cdot_2 (u\ X) = (u\ Y) \cdot_2 (Hom(A, -)_a\ f))$
by (*simp add: natural-transformation-def natural-transformation-axioms-def BB-Set*)
with 1 and 3
have *three*: $(\lambda f \in Hom\ A\ B. (set-func\ ((F_a\ f) \odot (u\ A)) (Id\ A)))$
 $= (\lambda f \in Hom\ A\ B. (set-func\ ((u\ B) \odot (Hom(A, -)_a\ f)) (Id\ A)))$
apply (*simp add: BB-Set Set-def*)
apply (*rule extensionalityI*)
apply (*rule restrict-extensional, rule restrict-extensional*)
by *simp*
have $\forall f \in Hom\ A\ B. set-dom\ (Hom(A, -)_a\ f) = Hom\ A\ A$
by (*intro ballI, simp add: homf-def hom-def*)
have *roolz*: $\bigwedge f. f \in Hom\ A\ B \implies set-dom\ (Hom(A, -)_a\ f) = Hom\ A\ A$
by (*simp add: homf-def hom-def*)
from 1 **have** *rooly*: $Id\ A \in Hom\ A\ A$..
have *roolx*: $\bigwedge f. f \in Hom\ A\ B \implies f \in Ar$
by (*simp add: hom-def*)
have *roolw*: $\bigwedge f. f \in Hom\ A\ B \implies Id\ A \in Hom\ A\ (Dom\ f)$
proof–
fix *f*
assume $f \in Hom\ A\ B$
hence $Dom\ f = A$ **by** (*simp add: hom-def*)
thus $Id\ A \in Hom\ A\ (Dom\ f)$
by (*simp add: rooly*)
qed
have *annoying*: $\bigwedge f. f \in Hom\ A\ B \implies Id\ A = Id\ (Dom\ f)$
by (*simp add: hom-def*)
have $(\lambda f \in Hom\ A\ B. (set-func\ ((u\ B) \odot (Hom(A, -)_a\ f)) (Id\ A)))$
 $= (\lambda f \in Hom\ A\ B. (compose\ (Hom\ A\ A)\ (set-func\ (u\ B))\ (set-func\ (Hom(A, -)_a\ f)))) (Id\ A))$
apply (*rule extensionalityI*)
apply (*rule restrict-extensional, rule restrict-extensional*)
by (*simp add: compose-def set-comp-def roolz rooly*)
also have $\dots = (\lambda f \in Hom\ A\ B. (set-func\ (u\ B)\ f))$
apply (*rule extensionalityI*)
apply (*rule restrict-extensional, rule restrict-extensional*)
apply (*simp add: compose-def homf-def rooly roolx roolw*)
apply (*simp only: annoying*)
apply (*simp add: roolx id-right*)
done
finally have *four*:
 $(\lambda f \in Hom\ A\ B. (set-func\ ((u\ B) \odot (Hom(A, -)_a\ f)) (Id\ A)))$
 $= (\lambda f \in Hom\ A\ B. (set-func\ (u\ B)\ f)) .$
from 2 and 3
have *uBhom*: $u\ B \in hom\ Set\ (Hom(A, -)_o\ B)\ (F_o\ B)$
by (*simp add: natural-transformation-def natural-transformation-axioms-def*)
with 3
have *five*: $set-dom\ (u\ B) = Hom\ A\ B$

```

  by (simp add: hom-def homf-def Set-def set-cat-def)
from uBhom
have six: set-cod (u B) = Fo B
  by (simp add: hom-def homf-def Set-def set-cat-def)
have seven: restrict (set-func (u B)) (Hom A B) = set-func (u B)
  apply (rule extensionalityI)
  apply (rule restrict-extensional)
proof-
  from uBhom have u B ∈ ar Set
    by (simp add: hom-def)
  hence almost: set-func (u B) ∈ extensional (set-dom (u B))
    by (simp add: Set-def set-cat-def set-arrow-def)
  from almost and five
  show set-func (u B) ∈ extensional (Hom A B)
    by simp
  fix f
  assume f ∈ Hom A B
  thus restrict (set-func (u B)) (Hom A B) f = set-func (u B) f
    by simp
qed
from one and two and three and four and five and six and seven
show σ(A,σ←(A,u)) B = u B
  by simp
qed

```

In order to state the lemma, we must rectify a curious omission from the Isabelle/HOL library. They define the idea of injectivity on a given set, but surjectivity is only defined relative to the entire universe of the target type.

definition

```

surj-on :: ['a ⇒ 'b, 'a set, 'b set] ⇒ bool where
surj-on f A B ⟷ (∀ y∈B. ∃ x∈A. f(x)=y)

```

definition

```

bij-on :: ['a ⇒ 'b, 'a set, 'b set] ⇒ bool where
bij-on f A B ⟷ inj-on f A & surj-on f A B

```

definition

```

equinumerous :: ['a set, 'b set] ⇒ bool (infix ≅ 40) where
equinumerous A B ⟷ (∃ f. bij-on f A B)

```

theorem (in Yoneda) Yoneda:

```

  assumes 1: A ∈ Ob
  shows Fo A ≅ {u. u : Hom(A,-) ⇒ F in Func(AA,Set)}
  apply (unfold equinumerous-def bij-on-def surj-on-def inj-on-def)
  apply (intro exI conjI bexI ballI impI)
proof-
  — Sandwich is injective
  fix x and y
  assume 2: x ∈ Fo A and 3: y ∈ Fo A

```

```

and 4:  $\sigma(A,x) = \sigma(A,y)$ 
hence  $\sigma^{\leftarrow}(A,\sigma(A,x)) = \sigma^{\leftarrow}(A,\sigma(A,y))$ 
  by simp
with unsandwich-left-inverse
show  $x = y$ 
  by (simp add: 1 2 3)
next
  — Sandwich covers F A
  fix  $u$ 
  assume  $u \in \{y. y : \text{Hom}(A,-) \Rightarrow F \text{ in } \text{Func}(AA,\text{Set})\}$ 
  hence 2:  $u : \text{Hom}(A,-) \Rightarrow F \text{ in } \text{Func}(AA,\text{Set})$ 
    by simp
  with 1 show  $\sigma(A,\sigma^{\leftarrow}(A,u)) = u$ 
    by (rule unsandwich-right-inverse)
  — Sandwich is into F A
  from 1 and 2
  have  $u A \in \text{hom Set}(\text{Hom } A A)(F_{\circ} A)$ 
    by (simp add: natural-transformation-def natural-transformation-axioms-def
homf-def)
  hence  $u A \in \text{ar Set}$  and  $\text{dom Set}(u A) = \text{Hom } A A$  and  $\text{cod Set}(u A) = F_{\circ} A$ 
    by (simp-all add: hom-def)
  hence  $uA\text{funcset}: \text{set-func}(u A) : (\text{Hom } A A) \rightarrow (F_{\circ} A)$ 
    by (simp add: Set-def set-cat-def set-arrow-def)
  from 1 have  $\text{Id } A \in \text{Hom } A A$  ..
  with  $uA\text{funcset}$ 
  show  $\sigma^{\leftarrow}(A,u) \in F_{\circ} A$ 
    by (simp add: unsandwich-def, rule funcset-mem)
qed

end

```

References

- [O’K04] Greg O’Keefe. Towards a readable formalisation of category theory. In Mike Atkinson, editor, *Computing: The Australasian Theory Symposium*, volume 91 of *Electronic Notes in Theoretical Computer Science*, pages 212–228. Elsevier, 2004.