Category Theory to Yoneda’s Lemma

Greg O’Keefe

June 11, 2019

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
  1.1 Definitions ........................................... 2
  1.2 Lemmas ............................................. 2

2 Set is a Category
  2.1 Definitions ........................................... 4
  2.2 Simple Rules and Lemmas ............................. 4
  2.3 Set is a Category .................................... 6

3 Functors
  3.1 Definitions ........................................... 10
  3.2 Simple Lemmas ....................................... 11
  3.3 Identity Functor .................................... 12

4 HomFunctors .......................... 14

5 Natural Transformations .......................... 18

6 Yonedas Lemma
  6.1 The Sandwich Natural Transformation ............... 19
  6.2 Sandwich Components are Bijective ................ 23
1 Categories

theory Cat
imports HOL-Library.FuncSet
begin

1.1 Definitions

record ('o, 'a) category =
  ob :: 'o set (Ob 70)
  ar :: 'a set (Ar 70)
  dom :: 'a ⇒ 'o (Dom - [81] 70)
  cod :: 'a ⇒ 'o (Cod - [81] 70)
  id :: 'o ⇒ 'a (Id - [81] 80)
  comp :: 'a ⇒ 'a ⇒ 'a (infixl · 60)

definition hom :: [('o, 'a, 'm) category-scheme, 'o, 'o] ⇒ 'a set
  (Hom - - [81,81] 80) 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]:
  A B C. (comp CC) :: (Hom B C) ⇒ (Hom A B) ⇒ (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 \( \text{Dom} \, f = A \) using assms by (simp add: hom-def)
  show \( \text{Cod} \, f = B \) using assms by (simp add: hom-def)
qed

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

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

lemma (in category) compI [intro]:
  assumes \( f : f \in \text{Ar} \) and \( g : g \in \text{Ar} \) and \( \text{Cod} \, f = \text{Dom} \, g \)
  shows \( g \cdot f \in \text{Ar} \) and \( \text{Dom} \, (g \cdot f) = \text{Dom} \, f \)
  and \( \text{Cod} \, (g \cdot f) = \text{Cod} \, g \)
proof –
  have \( f \in \text{Hom} \, (\text{Dom} \, f) \, (\text{Cod} \, f) \) using \( f \) by (simp add: hom-def)
  with \( \text{Cod} \, f = \text{Dom} \, g \) have \( f \text{-homset: } f \in \text{Hom} \, (\text{Dom} \, f) \, (\text{Dom} \, g) \) by simp
  have \( g \text{-homset: } g \in \text{Hom} \, (\text{Dom} \, g) \, (\text{Cod} \, g) \) using \( g \) by (simp add: hom-def)
  have \( \cdot : \text{Hom} \, (\text{Dom} \, g) \, (\text{Cod} \, g) \to \text{Hom} \, (\text{Dom} \, f) \, (\text{Cod} \, g) \) ..
  from this and \( g \text{-homset} \) have \( \cdot \, g \in \text{Hom} \, (\text{Dom} \, f) \, (\text{Cod} \, g) \)
    by (rule funcset-mem)
  from this and \( f \text{-homset} \) have \( g \cdot f \in \text{Hom} \, (\text{Dom} \, f) \, (\text{Cod} \, g) \)
    by (rule funcset-mem)
  thus \( g \cdot f \in \text{Ar} \)
    by (simp add: hom-def)
  from \( g \cdot f \text{-homset} \) show \( \text{Dom} \, (g \cdot f) = \text{Dom} \, f \) and \( \text{Cod} \, (g \cdot f) = \text{Cod} \, g \)
    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 ⇒ 'c
    set-cod :: 'c set

definition set-arrow :: ['c set, 'c set-arrow] ⇒ bool where
    set-arrow U f ←→ set-dom f ⊆ U & set-cod f ⊆ U & (set-func f) (set-dom f) → (set-cod f) & set-func f ∈ extensional (set-dom f)

definition set-id :: ['c set, 'c set] ⇒ 'c set-arrow where
    set-id U = (λs∈Pow U. (|set-dom=s, set-func=λx∈s. x, set-cod=s|))

definition set-comp :: ['c set-arrow, 'c set-arrow] ⇒ 'c set-arrow (infix ⊙ 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 ⇒ ('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 ⊆ U ⇒ A ∈ ob (set-cat U)
    by (simp add: set-cat-def)

lemma set-objectE [intro]: A ∈ ob (set-cat U) ⇒ A ⊆ 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 \( \{\text{set-dom} = A, \text{set-func} = f, \text{set-cod} = B\} \in \text{hom} \ (\text{set-cat } U) \ A B \)
  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]: \( \text{comp} \ (\text{set-cat } U) \ g \ f = g \circ f \)
  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 \( \ldots \subseteq U \) 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 \( \ldots \subseteq U \) 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)

\[
\begin{align*}
\text{show 2:} & \quad \text{set-dom} f = A \\
\text{using} \ f \ & \text{by} \ (\text{simp add: set-cat-def hom-def set-arrow-def}) \\
\text{from} \ 1 \ & \text{have} \ \text{set-dom} f \subseteq U .. \\
\text{thus} \ A \subseteq U \ & \text{by} \ (\text{simp add: 2}) \\
\text{show 3:} & \quad \text{set-cod} f = B \\
\text{using} \ f \ & \text{by} \ (\text{simp add: set-cat-def hom-def set-arrow-def}) \\
\text{from} \ 1 \ & \text{have} \ \text{set-cod} f \subseteq U .. \\
\text{thus} \ B \subseteq U \ & \text{by} \ (\text{simp add: 3}) \\
\text{have} \ & \text{set-func} f \in (\text{set-dom} f) \rightarrow (\text{set-cod} f) \\
\text{using} \ f \ & \text{by} \ (\text{auto simp add: set-cat-def hom-def set-arrow-def}) \\
\text{thus} \ & \text{set-func} f \in A \rightarrow B \ & \text{by} \ (\text{simp add: 2 3}) \\
\end{align*}
\]

qed

2.3 Set is a Category

lemma set-id-left:
assumes \( f: f \in \text{ar} \ (\text{set-cat} U) \)
shows \( \text{set-id} U \ (\text{set-cod} f) \circ f = f \)

proof
from \( f \in \text{ar} \ (\text{set-cat} U) \) have \( \text{set-cod} f \subseteq U .. \)
hence 1: \( \text{set-id} U \ (\text{set-cod} f) \circ f = \)
\[
\begin{aligned}
\& \quad \text{set-dom}\Rightarrow\text{set-dom} f, \\
\& \quad \text{set-func}\Rightarrow\text{compose} \ (\text{set-dom} f) \ (\lambda x\in\text{set-cod} f. x) \ (\text{set-func} f), \\
\& \quad \text{set-cod}\Rightarrow\text{set-cod} f \\
\end{aligned}
\]
using \( f \) by (simp add: set-comp-def set-id-def)

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

proof (rule extensionalityI)

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

by (rule compose-extensional)

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

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

fix \( x \)

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

have \( \text{f-into-cod}: \text{set-func} f : (\text{set-dom} f) \rightarrow (\text{set-cod} f) \)

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

from \( \text{f-into-cod} \) and \( \text{x-in-dom} \)

have \( \text{f-x-in-cod}: \text{set-func} f \ x \in \text{set-cod} f \)

by (rule funset-mem)

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

by (simp add: x-in-dom f-x-in-cod compose-def)

qed

from 1 have \( \text{set-id} U \ (\text{set-cod} f) \circ f = \)
\[
\begin{aligned}
\& \quad \text{set-dom}\Rightarrow\text{set-dom} f, \\
\end{aligned}
\]

6
lemma set-id-right:
  assumes f: f ∈ ar (set-cat U)
  shows f ⊙ (set-id U (set-dom f)) = f
proof-
  from f ∈ ar (set-cat U) have set-dom f ⊆ U ..
  hence 1: f ⊙ (set-id U (set-dom f)) =
    ⟨ set-dom = set-dom f,
      set-func = compose (set-dom f) (set-func f) (λx∈set-dom f. x),
      set-cod = set-cod f ⟩
    using f by (simp only: 2)
  have 2: compose (set-dom f) (set-func f) (λx∈set-dom f. x) = set-func f
  proof (rule extensionalityI)
    show compose (set-dom f) (set-func f) (λx∈set-dom f. x) ∈ extensional (set-dom f)
      by (rule compose-extensional)
    show set-func f ∈ extensional (set-dom f)
      using f by (simp add: set-cat-def set-arrow-def)
    fix x
    assume x-in-dom: x ∈ set-dom f
    thus compose (set-dom f) (set-func f) (λx∈set-dom f. x) x = set-func f x
      by (simp add: compose-def)
  qed
from 1 have f ⊙ (set-id U (set-dom f)) =
  ⟨ set-dom = set-dom f,
    set-func = set-func f,
    set-cod = set-cod f ⟩
  by (simp only: 2)
also have ... = f
  by simp
finally show ?thesis .
qed

lemma set-id-hom:
  assumes A ∈ ob (set-cat U)
  shows id (set-cat U) A ∈ hom (set-cat U) A A
proof–
from \((A \in \text{ob (set-cat } U))\) have \(1: A \subseteq U\) ..
hence id \((\text{set-cat } U)\) \(A = \{|\text{set-dom} = A, \text{set-func} = \lambda x \in A. x, \text{set-cod} = A\}\)
by (simp add: set-cat-def set-id-def)
also have \(... \in \text{hom (set-cat } U)\) \(A A\)
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:
  \(\text{comp (set-cat } U) \in \text{hom (set-cat } U)\) \(B C \rightarrow \text{hom (set-cat } U)\) \(A B \rightarrow \text{hom (set-cat } U)\) \(A C\)
proof (rule funcsetI)
  fix \(g\)
  assume g-BC: \(g \in \text{hom (set-cat } U)\) \(B C\)
  hence comp-cod: \(\text{set-cod } g = C\) ..
  show \(\text{comp (set-cat } U)\) \(g \in \text{hom (set-cat } U)\) \(A B \rightarrow \text{hom (set-cat } U)\) \(A C\)
proof (rule funcsetI)
  fix \(f\)
  assume f-AB: \(f \in \text{hom (set-cat } U)\) \(A B\)
  hence comp-dom: \(\text{set-dom } f = A\) ..
  show \(\text{comp (set-cat } U)\) \(g f \in \text{hom (set-cat } U)\) \(A C\)
proof
  have \(\text{comp (set-cat } U)\) \(g f =\)
    
set-dom = A,
set-func = \text{compose (set-dom } f\) \(\text{ (set-func } g\) \(\text{ (set-func } f\),
set-cod = C

by (simp add: set-cat-def set-comp-def comp-cod comp-dom)
also have \(... \in \text{hom (set-cat } U)\) \(A C\)
proof (rule set-homI)
from f-AB show \(A \subseteq U\) ..
from g-BC show \(C \subseteq U\) ..
from f-AB have fs-f: \(\text{set-func } f: A \rightarrow B\) ..
from g-BC have fs-g: \(\text{set-func } g: B \rightarrow C\) ..
from fs-g and fs-f
  show \(\text{compose (set-dom } f\) \(\text{ (set-func } g\) \(\text{ (set-func } f\) : A \rightarrow C\)
  by (simp only: comp-dom) (rule funcset-compose)
show \(\text{compose (set-dom } f\) \(\text{ (set-func } g\) \(\text{ (set-func } f\) \(\in \text{extensional } A\)
  by (simp only: comp-dom) (rule compose-extensional)
qed
finally show ?thesis .
qed
We reason explicitly about the function component of the composite arrow, leaving the rest to the simplifier.

**lemma** set-comp-associative:

```plaintext
fixes f and g and h
assumes f: f ∈ ar (set-cat U)
and g: g ∈ ar (set-cat U)
and h: h ∈ 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)
    show set-func h ∈ set-dom h → set-dom g
      using h hg by (simp add: set-cat-def set-arrow-def)
  qed
qed
```

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

```plaintext
proof (rule category.intro)
  fix f
  assume f: f ∈ ar (set-cat U)
  show dom (set-cat U) f ∈ ob (set-cat U) using f ..
  show cod (set-cat U) f ∈ ob (set-cat U) using f ..
  show comp (set-cat U) (id (set-cat U) (cod (set-cat U) 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 ∈ ob (set-cat U)
  then show id (set-cat U) A ∈ hom (set-cat U) A A
    by (rule set-id-hom)
next
  fix A and B and C
  show comp (set-cat U) ∈ hom (set-cat U) B C → hom (set-cat U) A B → hom (set-cat U) A C
    by (rule set-comp-types)
next
  fix f and g and h
assume f ∈ ar (set-cat U)
```
and \( g \in \text{ar (set-cat } U \text{)} \)
and \( h \in \text{ar (set-cat } U \text{)} \)
and \( \text{cod (set-cat } U \text{) } h = \text{dom (set-cat } U \text{) } g \)
and \( \text{cod (set-cat } U \text{) } g = \text{dom (set-cat } U \text{) } f \)
then show \( \text{comp (set-cat } U \text{) } f (\text{comp (set-cat } U \text{) } g \cdot h) = \text{comp (set-cat } U \text{) } (\text{comp (set-cat } U \text{) } f \cdot g) \cdot h \)
by (rule set-comp-associative)

qed

end

3 Functors

theory Functors
imports Cat
begin

3.1 Definitions

record \((\cdot o1, \cdot a1, \cdot o2, \cdot a2)\) functor =
  om :: \(\cdot o1 \Rightarrow \cdot o2\)
  am :: \(\cdot a1 \Rightarrow \cdot a2\)

abbreviation om-syn \((\cdot o [81])\) where
  \(F\cdot o \equiv \text{om } F\)

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

locale two-cats = AA?: category AA + BB?: category BB
for AA :: \((\cdot o1, \cdot a1, \cdot o2, \cdot a2)\)category-scheme (structure)
and BB :: \((\cdot o2, \cdot a2, \cdot o2, \cdot a2)\)category-scheme (structure) +
fixes preserves-dom :: \((\cdot o1, \cdot a1, \cdot o2, \cdot a2)\)functor \(\Rightarrow\) bool
and preserves-cod :: \((\cdot o1, \cdot a1, \cdot o2, \cdot a2)\)functor \(\Rightarrow\) bool
and preserves-id :: \((\cdot a1, \cdot a1, \cdot a2, \cdot a2)\)functor \(\Rightarrow\) bool
and preserves-comp :: \((\cdot o1, \cdot a1, \cdot o2, \cdot a2)\)functor \(\Rightarrow\) bool

defines preserves-dom G \(\equiv \forall f \in \text{Ar } AA\cdot \text{G}_o (\text{Dom } AA \cdot f) = \text{Dom } BB \cdot (\text{G}_a \cdot f)\)
and preserves-cod G \(\equiv \forall f \in \text{Ar } AA\cdot \text{G}_o (\text{Cod } AA \cdot f) = \text{Cod } BB \cdot (\text{G}_a \cdot f)\)
and preserves-id G \(\equiv \forall A \in \text{Ob } AA\cdot \text{G}_a (\text{Id } AA \cdot A) = \text{Id } BB \cdot (\text{G}_o \cdot A)\)
and preserves-comp G \(\equiv \forall f \in \text{Ar } AA\cdot \forall g \in \text{Ar } AA\cdot \text{Cod } AA \cdot f = \text{Dom } AA \cdot g \rightarrow (\text{G}_a \cdot g \cdot AA \cdot f) = (\text{G}_a \cdot f)\)

locale functor = two-cats +
fixes F (structure)
assumes F-preserves-arrows: \(F\cdot a : \text{Ar } AA \rightarrow \text{Ar } BB\)
and F-preserves-objects: \(F\cdot o : \text{Ob } AA \rightarrow \text{Ob } BB\)
and $F$-preserves-dom: $\text{preserves-dom } F$

and $F$-preserves-cod: $\text{preserves-cod } F$

and $F$-preserves-id: $\text{preserves-id } F$

and $F$-preserves-comp: $\text{preserves-comp } F$

begin

lemmas $F$-axioms = $F$-preserves-arrows $F$-preserves-objects $F$-preserves-dom $F$-preserves-cod $F$-preserves-id $F$-preserves-comp

lemmas func-pred-defs = $\text{preserves-dom-def } \text{preserves-cod-def } \text{preserves-id-def } \text{preserves-comp-def}$

end

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

abbreviation

$\text{Functor } (\text{Functor } - : - \rightarrow - [81])$ where

$\text{Functor } F : AA \rightarrow BB \equiv \text{functor } AA BB F$

3.2 Simple Lemmas

For example:

lemma (in functor) $\text{Functor } F : AA \rightarrow BB$ ..

lemma $\text{functors-preserve-arrows}$ [intro]:

assumes $\text{Functor } F : AA \rightarrow BB$

shows $F_a f \in ar BB$

proof -

from $\langle \text{Functor } F : AA \rightarrow BB \rangle$

have $F_a : ar AA \rightarrow ar BB$

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

from this and $f \in ar AA$

show ?thesis by (rule funcsset-mem)

qed

lemma (in functor) $\text{functors-preserve-homsets}$:

assumes 1: $A \in \text{Ob}_{AA}$

and 2: $B \in \text{Ob}_{AA}$

and 3: $f \in \text{Hom}_{AA} A B$

shows $F_a f \in \text{Hom}_{BB} (F_o A) (F_o B)$

proof -

from 3

have 4: $f \in Ar$

by (simp add: hom-def)

with $F$-preserves-arrows

have 5: $F_a f \in Ar_{BB}$

by (rule funcsset-mem)
from \(4\) and \(F\)-preserves-dom

have \(Dom_{BB} (F_a f) = F_o (Dom_{AA} f)\)
  by (simp add: preserves-dom-def)

also from \(3\) have \(\ldots = F_o A\)
  by (simp add: hom-def)

finally have \(6\): \(Dom_{BB} (F_a f) = F_o A\)
  by (simp add: hom-def)

from \(4\) and \(F\)-preserves-cod

have \(Cod_{BB} (F_a f) = F_o (Cod_{AA} f)\)
  by (simp add: preserves-cod-def)

also from \(3\) have \(\ldots = F_o B\)
  by (simp add: hom-def)

finally have \(7\): \(Cod_{BB} (F_a f) = F_o B\)
  by (simp add: hom-def)

from \(5\) and \(6\) and \(7\)

show ?thesis
  by (simp add: hom-def)

qed

lemma functors-preserve-objects [intro]:
  assumes Functor \(F : AA \to BB\)
    and \(A \in ob AA\)
  shows \(F_o A \in ob BB\)

proof –
  from \(\langle\text{Functor } F : AA \to BB\rangle\)
  have \(F_o : ob AA \to ob BB\)
    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
  \(\text{id-func} :: ([',']o,[',']a,[',']m)\text{-category-scheme}\Rightarrow ([',']o,[',']a,[',']o,[',']a)~\text{functor}\)
  where
  \(\text{id-func }CC = ([o]\Rightarrow\lambda A\in ob~CC.~A),~[a]\Rightarrow\lambda f\in ar~CC.~f))\)

locale one-cat = two-cats +
  assumes endo: \(BB = AA\)

lemma (in one-cat) id-func-preserved-arrows:
  shows \((\text{id-func }AA)_A : Ar \to Ar\)
  by (unfold id-func-def, rule funcsetI, simp)

lemma (in one-cat) id-func-preserved-objects:
  shows \((\text{id-func }AA)_A : Ob \to Ob\)
  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 ∈ Ar
  hence lhs: (id-func AA)_a (Dom f) = Dom f
    by (simp add: id-func-def) auto
  have (id-func AA)_a f = f
    using f by (simp add: id-func-def)
  hence rhs: Dom (id-func AA)_a f = Dom f
    by simp
  from lhs and rhs show (id-func AA)_o (Dom f) = Dom (id-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 ∈ Ar
  hence lhs: (id-func AA)_o (Cod f) = Cod f
    by (simp add: id-func-def) auto
  have (id-func AA)_a f = f
    using f by (simp add: id-func-def)
  hence rhs: Cod (id-func AA)_a f = Cod f
    by simp
  from lhs and rhs show (id-func AA)_o (Cod f) = Cod (id-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 ∈ Ob
  hence lhs: (id-func AA)_a (Id A) = Id A
    by (simp add: id-func-def) auto
  have (id-func AA)_o A = A
    using A by (simp add: id-func-def)
  hence rhs: Id ((id-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:
  unfolding (id-func AA)
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 AA BB
  for AA :: ('o,'a,'m)category-scheme (structure)
  and BB (structure) +
fixes U and Set
defines U ≡ (UNIV::'a set)
defines Set ≡ set-cat U
assumes \(BB-Set: BB = Set\)
defines \(homf \ (\Hom(\cdot,\cdot))\)
fixes \(homf A \equiv \emptyset\)
\(om = (\lambda B \in Ob. \Hom A B),\)
\(am = (\lambda f \in Ar. \{\text{set-dom} = \Hom A (\Dom f), \text{set-func} = (\lambda g \in \Hom A (\Dom f). f \cdot g), \text{set-cod} = \Hom A (\Cod f)\})\)

lemma (in into-set) \(homf\)-preserves-arrows:
\(\Hom(A,\cdot)_A : Ar \rightarrow ar \ Set\)
proof (rule funcsetI)
  fix \(f\)
  assume \(f: f \in Ar\)
  thus \(\Hom(A,\cdot)_A f \in ar \ Set\)
proof (simp add: \(\text{homf-def \ Set-def \ set-cat-def \ set-arrow-def \ U-def\)}\)
  have 1: \((\cdot) : \Hom (\Dom f) (\Cod f) \rightarrow \Hom A (\Dom f) \rightarrow \Hom A (\Cod f)\) ..
  have 2: \(f \in \Hom (\Dom f) (\Cod f)\) using \(f\) by (simp add: \(\text{hom-def}\))
  from 1 and 2 have 3: \((\cdot) : \Hom A (\Dom f) \rightarrow \Hom A (\Cod f)\)
    by (rule funcset-mem)
  show \((\lambda g \in \Hom A (\Dom f). f \cdot g) : \Hom A (\Dom f) \rightarrow \Hom A (\Cod f)\)
  proof (rule funcsetI)
    fix \(g'\)
    assume \(g' \in \Hom A (\Dom f)\)
    from 3 and this show \((\lambda g \in \Hom A (\Dom f). f \cdot g) g' \in \Hom A (\Cod f)\)
      by simp (rule funcset-mem)
  qed
qed

lemma (in into-set) \(homf\)-preserves-objects:
\(\Hom(A,\cdot)_{\cdot} : Ob \rightarrow ob \ Set\)
proof (rule funcsetI)
  fix \(B\)
  assume \(B: B \in Ob\)
  have \(\Hom(A,\cdot)_{\cdot} B = \Hom A B\)
    using \(B\) by (simp add: \(\text{homf-def}\))
  moreover have \(\ldots \in ob \ Set\)
    by (simp add: \(U-def \ Set-def \ set-cat-def\))
  ultimately show \(\Hom(A,\cdot)_{\cdot} B \in ob \ Set \ by \ simp\)
qed

lemma (in into-set) \(homf\)-preserves-dom:
  assumes \(f: f \in Ar\)
  shows \(\Hom(A,\cdot)_{\cdot} (\Dom f) = \Dom \ Set \ (\Hom(A,\cdot)_A f)\)
proof
  have \(\Dom f \in Ob\) using \(f\) ..
hence 1: \( \text{Hom}(A, \cdot)_\circ (\text{Dom } f) = \text{Hom } A (\text{Dom } f) \)
  using \( f \) by \((\text{simp add: homf-def})\)
have 2: \( \text{dom } \text{Set} (\text{Hom}(A, \cdot)_a f) = \text{Hom } A (\text{Dom } f) \)
  using \( f \) by \((\text{simp add: Set-def homf-def})\)
from 1 and 2 show \( ?\text{thesis} \) by simp
qed

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

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

lemma (in into-set) homf-preserves-comp:
  assumes \( f: f \in \text{Ar} \)
  and \( g: g \in \text{Ar} \)
  and \( fg: \text{Cod } f = \text{Dom } g \)
  shows \( \text{Hom}(A, \cdot)_a (g \cdot f) = (\text{Hom}(A, \cdot)_a g) \circ (\text{Hom}(A, \cdot)_a f) \)
proof–
  have 1: \( g \cdot f \in \text{Ar} \) using \( \text{assms} \) ..
have 2: \( \text{Dom} \ (g \cdot f) = \text{Dom} f \) using \( fgfg \).

have 3: \( \text{Cod} \ (g \cdot f) = \text{Cod} g \) using \( fgfg \).

have lhs: \( \text{Hom}(A,\cdot)_a \ (g \cdot f) = \emptyset \)

set-func=(\( \lambda h \in \text{Hom} \ (\text{Dom} f), set-func=(\lambda h \in \text{Hom} \ (\text{Dom} f). \ (g \cdot f) \cdot h), set-cod=\text{Hom} \ (\text{Cod} g) \)

by (simp add: homf-def 1 2 3)

have 4: set-dom \((\text{Hom}(A,\cdot)_a g) \odot (\text{Hom}(A,\cdot)_a f)\) = \text{Hom} \ (\text{Dom} f)
using f by (simp add: set-comp-def homf-def)

have 5: set-cod \((\text{Hom}(A,\cdot)_a g) \odot (\text{Hom}(A,\cdot)_a f)\) = \text{Hom} \ (\text{Cod} g)
using g by (simp add: set-comp-def homf-def)

have set-func \((\text{Hom}(A,\cdot)_a g) \odot (\text{Hom}(A,\cdot)_a f)\)
= compose \((\text{Hom} \ (\text{Dom} f)) (\lambda y \in \text{Hom} \ (\text{Dom} g). \ g \cdot y) (\lambda x \in \text{Hom} \ (\text{Dom} f). \ f \cdot x)\)
using fg by (simp add: set-comp-def homf-def)

also have \( \ldots = (\lambda h \in \text{Hom} \ (\text{Dom} f). \ (g \cdot f) \cdot h) \)

g

proof \{
  rule extensionalityI,
  rule compose-extensional,
  rule restrict-extensional,
  simp\)

fix h

assume 10: \( h \in \text{Hom} \ (\text{Dom} f) \)

hence 11: \( f \cdot h \in \text{Hom} \ (\text{Dom} g) \)

proof–

from 10 have h \( \in Ar \) by (simp add: hom-def)

have 100: \( \cdot : \text{Hom} \ (\text{Dom} f) (\text{Dom} g) \rightarrow \text{Hom} \ (\text{Dom} f) \rightarrow \text{Hom} \ (\text{Dom} g) \)
by (rule AA.comp-types)

have \( f \in \text{Hom} \ (\text{Dom} f) (\text{Cod} f) \) using f by (simp add: hom-def)

hence 101: \( f \in \text{Hom} \ (\text{Dom} f) (\text{Dom} g) \) using fg by simp

from 100 and 101

have \( \cdot \ : \text{Hom} \ (\text{Dom} f) \rightarrow \text{Hom} \ (\text{Dom} g) \)
by (rule funcset-mem)

from this and 10

show \( f \cdot h \in \text{Hom} \ (\text{Dom} g) \)
by (rule funcset-mem)

qed

hence \( \text{Cod} \ (f \cdot h) = \text{Dom} g \)

and \( \text{Dom} \ (f \cdot h) = A \)

and \( f \cdot h \in Ar \)
by (simp-all add: hom-def)

thus compose \((\text{Hom} \ (\text{Dom} f)) (\lambda y \in \text{Hom} \ (\text{Dom} g). \ g \cdot y) (\lambda x \in \text{Hom} \ (\text{Dom} f). \ f \cdot x)\) \( h = \)
(\( g \cdot f \) \cdot h)

using fgfg 10 by (simp add: compose-def 10 11 hom-def)

qed

finally have 6: set-func \((\text{Hom}(A,\cdot)_a g) \odot (\text{Hom}(A,\cdot)_a f)\)
= \( (\lambda h \in \text{Hom} \ (\text{Dom} f). \ (g \cdot f) \cdot h) \).
from 4 and 5 and 6
have rhs: \((\text{Hom}(A,\cdot)_a \ g) \circ (\text{Hom}(A,\cdot)_a \ f) = \emptyset\)
set-dom=\text{Hom} \ A (\text{Dom} \ f),
set-func=(\lambda h\in\text{Hom} \ A (\text{Dom} \ f). \ (g \cdot f) \cdot h),
set-cod=\text{Hom} \ A (\text{Cod} \ g))
by simp
show ?thesis
by (simp add: lhs rhs)
qed

theorem (in into-set) homf-into-set:
Functor \text{Hom}(A,\cdot) : AA \rightarrow Set
proof (intro functor.intro functor-axioms.intro)
show \text{Hom}(A,\cdot)_a : Ar \rightarrow ar Set
by (rule homf-preserves-arrows)
show \text{Hom}(A,\cdot)_0 : Ob \rightarrow ob Set
by (rule homf-preserves-objects)
show \forall f\in Ar. \text{Hom}(A,\cdot)_o (\text{Dom} \ f) = \text{dom} Set (\text{Hom}(A,\cdot)_a \ f)
by (intro ballI) (rule homf-preserves-dom)
show \forall f\in Ar. \text{Hom}(A,\cdot)_o (\text{Cod} \ f) = \text{cod} Set (\text{Hom}(A,\cdot)_a \ f)
by (intro ballI) (rule homf-preserves-cod)
show \forall B\in Ob. \text{Hom}(A,\cdot)_a (\text{Id} \ B) = \text{id} Set (\text{Hom}(A,\cdot)_o \ B)
by (intro ballI) (rule homf-preserves-id)
show \forall f\in Ar. \forall g\in Ar.
\text{Cod} \ f = \text{Dom} \ g \implies
\text{Hom}(A,\cdot)_a (g \cdot f) = \text{comp} Set (\text{Hom}(A,\cdot)_a \ g) (\text{Hom}(A,\cdot)_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)
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 \rightarrow BB
and Functor G : AA \rightarrow BB
and u : ob AA \rightarrow ar BB
and \( u \in \text{extensional} (\text{ob } AA) \)
and \( \forall A \in \text{Ob} \). \( u A \in \text{Hom}_{BB} (F \circ A) (G \circ A) \)
and \( \forall A \in \text{Ob} \). \( \forall B \in \text{Ob} \). \( \forall f \in \text{Hom} A B \). \( (G \circ f) \cdot_{BB} (u A) = (u B) \cdot_{BB} (F \circ f) \)

abbreviation
\[
\text{nt-syn} \ (\ - \Rightarrow \ - \ ) \ (\text{in } \text{Func} \ (\ AA, \ BB)) \ (\text{where})
\]
\( u : F \Rightarrow G \ \text{in } \text{Func} (\ AA, \ BB) \equiv \ \text{natural-transformation} \ AA \ BB \ F G \ u \)

locale \( \text{endoNT} = \text{natural-transformation} + \text{one-cat} \)

theorem (in \( \text{endoNT} \)) \( \text{id-restrict-natural} \):
\( (\lambda A \in \text{Ob}. \ I d \ A) : (\text{id-func } AA) \Rightarrow (\text{id-func } AA) \ \text{in } \text{Func} (\ AA, AA) \)

proof (intro \text{natural-transformation.intro} \text{natural-transformation-axioms.intro} two-cats.intro ballI)

show \( (\lambda A \in \text{Ob}. \ I d \ A) : \text{Ob} \Rightarrow \text{Ar} \)
by (rule \text{funcsetI}) auto

show \( (\lambda A \in \text{Ob}. \ I d \ A) \in \text{extensional} (\text{Ob}) \)
by (rule \text{restrict-extensional})

fix \( A \)

assume \( A : A \in \text{Ob} \)

hence \( I d \ A \in \text{Hom} A A \).

thus \( (\lambda X \in \text{Ob}. \ I d \ X) \ A \in \text{Hom} ((\text{id-func } AA)_0 \ A) \ ((\text{id-func } AA)_0 \ A) \)
using \( A \) by (simp add: \text{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: \text{hom-def})

thus \( (\text{id-func } AA)_a \ f \cdot (\lambda A \in \text{Ob}. \ I d \ A) \ A \)
\( = (\lambda A \in \text{Ob}. \ I d \ A) \ B \cdot (\text{id-func } AA)_a \ f \)
by (simp add: \text{id-func-def})

qed (auto intro: \text{id-func-functor}, \text{unfold-locales}, \text{unfold-locales})

end

6 Yonedas Lemma

theory \( \text{Yoneda} \)

imports \text{HomFunctors} NatTrans

begin

6.1 The Sandwich Natural Transformation

locale \( \text{Yoneda} = \text{functor} + \text{into-set} + \)

assumes \text{TERM} \( (AA :: (\'a, \'a, \'m) \text{category-scheme}) \)

fixes \( \text{sandwich} :: [\'a, \'a] \Rightarrow \text{'a set-arrow} \ (\sigma[\cdot, \cdot]) \)

defines \( \text{sandwich} A a \equiv (\lambda B \in \text{Ob}. \)
set-dom=\text{Hom} \ A \ B, \\
set-func=(\lambda f \in \text{Hom} \ A \ B. \ \text{set-func} (\text{F} a \ f) \ a), \\
set-cod=\text{F} o \ B \\
\} \\
\text{fixes} \ \text{unsandwich} :: \ 'a \Rightarrow 'a \ \text{set-arrow} \\
\text{defines} \ \text{unsandwich} \ A \ u \equiv \text{set-func} (u \ A) (\text{Id} \ A) \\

\text{lemma (in Yoneda) F-into-set:} \\
\text{Functor} \ F : AA \rightarrow \text{Set} \\
\text{proof} - \\
\text{from F-axioms have} \ \text{Functor} \ F : AA \rightarrow BB \ \text{by intro-locales} \\
\text{thus} \ \text{?thesis} \\
\text{by} (\text{simp only}: \ BB-\text{Set}) \\
\text{qed} \\

\text{lemma (in Yoneda) F-comp-func:} \\
\text{assumes} 1: A \in \text{Ob} \ \text{and} 2: B \in \text{Ob} \ \text{and} 3: C \in \text{Ob} \\
\text{and} 4: g \in \text{Hom} \ A \ B \ \text{and} 5: f \in \text{Hom} \ B \ C \\
\text{shows} \ \text{set-func} (\text{F} a (f \cdot g)) = \text{compose} (\text{F} o A) (\text{set-func} (\text{F} a f)) (\text{set-func} (\text{F} a g)) \\
\text{proof} - \\
\text{from 4 and 5} \\
\text{have} 7: \text{Cod} \ g = \text{Dom} \ f \\
\text{and} 8: g \in \text{Ar} \\
\text{and} 9: f \in \text{Ar} \\
\text{and} 10: \text{Dom} \ g = A \\
\text{by} (\text{simp-all add}: \ \text{hom-def}) \\
\text{from F-preserves-dom and 8 and 10} \\
\text{have} 11: \text{set-dom} (\text{F} a g) = \text{F} o A \\
\text{by} (\text{simp add: preserves-dom-def BB-Set Set-def}) \text{ auto} \\
\text{from F-preserves-comp and 7 and 8 and 9} \\
\text{have} F a (f \cdot g) = (F a f) \cdot BB (F a g) \\
\text{by} (\text{simp add: preserves-comp-def}) \\
\text{hence} \ \text{set-func} (F a (f \cdot g)) = \text{set-func} ((F a f) \circ (F a g)) \\
\text{by} (\text{simp add: BB-Set Set-def}) \\
\text{also have} \ldots = \text{compose} (F o A) (\text{set-func} (F a f)) (\text{set-func} (F a g)) \\
\text{by} (\text{simp add: set-comp-def 11}) \\
\text{finally show} \ \text{?thesis} . \\
\text{qed} \\

\text{lemma (in Yoneda) sandwich-funcset:} \\
\text{assumes} A: A \in \text{Ob} \\
\text{and} a \in \text{F} o A \\
\text{shows} \ \sigma(A,a) : \text{Ob} \rightarrow \text{ar Set} \\
\text{proof (rule funcsetI)} \\
\text{fix} B \\
\text{assume} B: B \in \text{Ob} \\
\text{thus} \ \sigma(A,a) \ B \in \text{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_{BB} (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 (a \in F o A)
    by (rule funcset-mem)
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 (a \in F o A) by (rule sandwich-funcset)
hence \sigma(A,a) B \in ar Set
    using B by (rule funcset-mem)
thus \sigma(A,a) B \in hom Set (Hom A B) (F o B)
    using (a \in F o A)
    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 aFu: a \in F o A
  and fBC: f \in Hom B C
  shows (F_a f) \circ (\sigma(A,a) B) = (\sigma(A,a) C) \circ (Hom(A,-)a f)
proof –
from \(fBC\) have 1: \(f \in Ar\) and 2: \(\text{Dom } f = B\) and 3: \(\text{Cod } f = C\)

by (simp-all add: hom-def)

from \(BOb\) have \(\text{set-dom } ((F_a f) \circ (\sigma(A,a) B)) = \text{Hom } A B\)

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

also have \(\ldots = \text{set-dom } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f))\)

by (simp add: set-comp-def homf-def 1 2)

finally have \(\text{set-dom-eq:}\)

\[
\text{set-dom } ((F_a f) \circ (\sigma(A,a) B)) = \text{set-dom } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f)).
\]

from \(BOb\) \(COb\) \(fBC\) have \((F_a f) \in \text{Hom}_{BB} (F_o B) (F_o C)\)

by (rule functors-preserve-homsets)

hence \(\text{set-cod } ((F_a f) \circ (\sigma(A,a) B)) = F_o C\)

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

also from \(COb\)

have \(\ldots = \text{set-cod } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f))\)

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

finally have \(\text{set-cod-eq:}\)

\[
\text{set-cod } ((F_a f) \circ (\sigma(A,a) B)) = \text{set-cod } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f)).
\]

from \(AOh\) and \(BOb\) and \(COb\) and \(fBC\) and \(aFa\)

have \(\text{set-func-lhs:}\)

\[
\text{set-func } ((F_a f) \circ (\sigma(A,a) B)) =
(\lambda g \in \text{Hom } A B. \text{set-func } (F_a (f \cdot g)) a)
\]

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 \((\cdot,:) : \text{Hom } B C \to \text{Hom } A B \to \text{Hom } A C\) ..

from this and \(fBC\)

have \(\text{optType}: (\cdot,:) : \text{Hom } A B \to \text{Hom } A C\)

by (rule funcset-mem)

from 1 and 2

have \(\text{set-func } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f)) =
(\lambda g \in \text{Hom } A B. \text{set-func } (\sigma(A,a) C) (f \cdot g))\)

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 \(\text{optType}\)

have \(\ldots = (\lambda g \in \text{Hom } A B. \text{set-func } (F_a (f \cdot g)) a)\)

apply (simp add: sandwich-def)

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

by (simp add:Pi-def)

finally have \(\text{set-func-rhs:}\)

\[
\text{set-func } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f)) =
(\lambda g \in \text{Hom } A B. \text{set-func } (F_a (f \cdot g)) a).
\]

from \(\text{set-func-lhs}\) and \(\text{set-func-rhs}\) have

\[
\text{set-func } ((F_a f) \circ (\sigma(A,a) B)) =
\text{set-func } ((\sigma(A,a) C) \circ (\text{Hom}(A,-) a f))
\]

by simp
with set-dom-eq and set-cod-eq show θthesis
  by simp
qed

lemma (in Yoneda) sandwich-natural:
  assumes A ∈ Ob
  and a ∈ F o A
  shows σ(A,a) : Hom(A,-) ⇒ F in Func(AA,Set)
proof (intro natural-transformation.intro natural-transformation-axioms.intro two-cats.intro)
  show category AA ..
  show category Set
    by (simp only: Set-def)
  show Functor Hom(A,-) : AA −→ Set
    by (rule homf-into-set)
  show Functor F : AA −→ Set
    by (rule F-into-set)
  show ∀ B ∈ Ob. σ(A,a) B ∈ hom Set (Hom(A,-) a B) (F o B)
    using assms by (auto simp add: homf-def intro: sandwich-type)
  show σ(A,a) : Ob → ar Set
    using assms by (rule sandwich-funcset)
  show σ(A,a) ∈ extensional (Ob)
    unfolding sandwich-def by (rule restrict-extensional)
  show ∀ B ∈ Ob. ∀ C ∈ Ob. ∀ f ∈ Hom B C.
    comp Set (F a f) (σ(A,a) B) = comp Set (σ(A,a) C) (Hom(A,-) a f)
    using assms by (auto simp add: Set-def intro: sandwich-commutes)
qed

6.2 Sandwich Components are Bijective

lemma (in Yoneda) unsandwich-left-inverse:
  assumes 1: A ∈ Ob
  and 2: a ∈ F o A
  shows σ←(A,σ(A,a)) = a
proof–
  from 1 have Id A ∈ Hom A A ..
  with 1
  have 3: σ←(A,σ(A,a)) = set-func (F a (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 ∈ Ob BB
    by (rule funcset-mem)
  hence F o A ⊆ U
    by (simp add: BB-Set Set-def set-cat-def)
with 2
  have 5: set-func (id Set (F o A)) a = a

23
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 ∈ Ob
  and 2: u : Hom(A,-) ⇒ F in Func(AA,Set)
  shows σ(A,σ⁻¹(A,u)) = u
proof (rule extensionalityI)
  show σ(A,σ⁻¹(A,u)) ∈ extensional (Ob)
    by (unfold sandwich-def, rule restrict-extensional)
  from 2 show u ∈ extensional (Ob)
    by (simp add: natural-transformation-def natural-transformation-axioms-def)
fix B
assume 3: B ∈ Ob
with 1
  have one: σ(A,σ⁻¹(A,u)) B = []
    set-dom = Hom A B,
    set-func = (λf∈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,-)₀ A = Hom A A
    by (simp add: homf-def)
with 1 and 2 have (u A) ∈ hom Set (Hom A A) (F₀ 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 ∈ set-dom (u A)
  by (simp)(rule)
  have two: (λf∈Hom A B. (set-func (F a f)) (set-func (u A) (Id A)))
    = (λf∈Hom A B. (set-func ((F a f) ⊗ (u A)) (Id A)))
    by (rule extensionalityI, rule restrict-extensional, rule restrict-extensional, simp add: set-comp-def compose-def applicable)
  from 2 have (∀ X∈Ob. ∀ Y∈Ob. ∀ f∈Hom X Y. (F a f) · BB (u X) = (u Y) · BB (Hom(A,-)₀ f))
    by (simp add: natural-transformation-def natural-transformation-axioms-def BB-Set)
with 1 and 3 have three: (λf∈Hom A B. (set-func ((F a f) ⊗ (u A)) (Id A)))
  = (λf∈Hom A B. (set-func ((u B) ⊗ (Hom(A,-)₀ f)) (Id A)))
  apply (simp add: BB-Set Set-def)
  apply (rule extensionalityI)
  apply (rule restrict-extensional, rule restrict-extensional)
  by simp

24
have $\forall f \in \text{Hom } A, B. \text{set-dom } (\text{Hom}(A,-)_a f) = \text{Hom } A$
by (intro ballI, simp add: homf-def hom-def)

have rootz: $\forall f : \text{Hom } A, B \Longrightarrow \text{set-dom } (\text{Hom}(A,-)_a f) = \text{Hom } A$
by (simp add: homf-def hom-def)

from 1 have rooly: $\text{Id } A \in \text{Hom } A, A$
by (simp add: homf-def)

have roolz: $\forall f : \text{Hom } A, B \Longrightarrow f \in \text{Ar}$
by (simp add: hom-def)

have roolx: $\forall f : \text{Hom } A, B \Longrightarrow \text{Id } A \in \text{Hom } A, (\text{Dom } f)$
proof
  fix $f$
  assume $f \in \text{Hom } A, B$
  hence $\text{Dom } f = A$
by (simp add: hom-def)
  thus $\text{Id } A \in \text{Hom } A, (\text{Dom } f)$
by (simp add: rooly)
qed

have annoying: $\forall f : \text{Hom } A, B \Longrightarrow \text{Id } A = \text{Id } (\text{Dom } f)$
by (simp add: hom-def)

have $(\forall f : \text{Hom } A, B. (\text{set-func } ((u B) \circ (\text{Hom}(A,-)_a f)) (\text{Id } A))$
= $(\forall f : \text{Hom } A, B. (\text{compose } (\text{Hom } A, A) (\text{set-func } (u B)) (\text{set-func } (\text{Hom}(A,-)_a f)) \text{(Id } A)))$
apply (rule extensionalityI)
apply (rule restrict-extensional, rule restrict-extensional)
by (simp add: compose-def set-comp-def roolz rooly)
also have $\ldots = (\forall f : \text{Hom } A, B. (\text{set-func } (u B) f))$
apply (rule extensionalityI)
apply (rule restrict-extensional, rule restrict-extensional)
apply (simp add: compose-def homf-def roolz rooly)
apply (simp only: annoying)
apply (simp add: roolz id-right)
done

finally have four:
$(\forall f : \text{Hom } A, B. (\text{set-func } ((u B) \circ (\text{Hom}(A,-)_a f)) (\text{Id } A))$
= $(\forall f : \text{Hom } A, B. (\text{set-func } (u B) f))$
from 2 and 3
have uBhom: $u B \in \text{hom Set } (\text{Hom}(A,-)_a B) (F_\alpha B)$
by (simp add: natural-transformation-def natural-transformation-axioms-def)

with 3
have five: $\text{set-dom } (u B) = \text{Hom } A, B$
by (simp add: hom-def homf-def Set-def set-cat-def)
from uBhom
have six: $\text{set-cod } (u B) = F_\alpha B$
by (simp add: hom-def homf-def Set-def set-cat-def)

have seven: $\text{restrict } (\text{set-func } (u B)) (\text{Hom } A, B) = \text{set-func } (u B)$
apply (rule extensionalityI)
apply (rule restrict-extensional)
proof
from uBhom have $u B \in \text{ar Set}$
by (simp add: hom-def)
故 $\text{almost } \text{set-func } (u B) \in \text{extensional } (\text{set-dom } (u B))$
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,x)) 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-betw f A B)

lemma bij-betw-eq:
  bij-betw f A B ←→
  inj-on f A ∧ (∀ y ∈ B. ∃ x ∈ A. f(x)=y) ∧ (∀ x ∈ A. f x ∈ B)

unfolding bij-betw-def by auto

theorem (in Yoneda) Yoneda:
  assumes 1: A ∈ Ob
  shows F o A ∋ { u. u : Hom(A,-) ⇒ F in Func(AA,Set)}

unfolding equinumerous-def bij-betw-eq inj-on-def

proof (intro exI conjI bezI ballI impI)
  — Sandwich is injective
  fix x and y
  assume 2: x ∈ F o A and 3: y ∈ F o A
  and 4: σ(A,x) = σ(A,y)
  hence σ⁺(A,σ(A,x)) = σ⁺(A,σ(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 ∈ {y. y : Hom(A, -) ⇒ F in Func (AA, Set)}
hence 2: u : Hom(A, -) ⇒ F in Func (AA, Set)
  by simp
with 1 show σ(A, σ⁻¹(A, u)) = u
  by (rule unsandwich-right-inverse)
— Sandwich is into F A
from 1 and 2
have u A ∈ hom Set (Hom A A) (F o A)
  by (simp add: natural-transformation-def natural-transformation-axioms-def homf-def)
hence u A ∈ ar Set and dom Set (u A) = Hom A A and cod Set (u A) = F o A
  by (simp-all add: hom-def)
hence uAfuncset: set-func (u A) : (Hom A A) → (F o A)
  by (simp add: Set-def set-cat-def set-arrow-def)
from 1 have Id A ∈ Hom A A ..
with uAfuncset
show σ⁻¹(A, u) ∈ F o A
  by (simp add: unsandwich-def, rule funcset-mem)
next
fix x
assume x ∈ F o A
with 1 have σ(A, x) : Hom(A, -) ⇒ F in Func (AA, Set)
  by (rule sandwich-natural)
thus σ(A, x) ∈ {y. y : Hom(A, -) ⇒ F in Func (AA, Set)}
  by simp
qed

end

References