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.
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 \( \text{f-homset}: f \in \text{Hom} (\text{Dom} f) (\text{Dom} g) \) by simp
  have \( \text{g-homset}: g \in \text{Hom} (\text{Dom} g) (\text{Cod} g) \) using \( g \) by (simp add: hom-def)
  have \( (\cdot) : \text{Hom} (\text{Dom} g) (\text{Cod} g) \to \text{Hom} (\text{Dom} f) (\text{Dom} g) \to \text{Hom} (\text{Dom} f) (\text{Cod} g) \) ..
  from this and \( \text{g-homset} \)
  have \( (\cdot) : \text{Hom} (\text{Dom} f) (\text{Dom} g) \to \text{Hom} (\text{Dom} f) (\text{Cod} g) \)
    by (rule funcset-mem)
  from this and \( \text{f-homset} \)
  have \( \text{gf-homset}: g \cdot f \in \text{Hom} (\text{Dom} f) (\text{Cod} g) \)
    by (rule funcset-mem)
  thus \( g \cdot f \in \text{Ar} \)
    by (simp add: hom-def)
  from \( \text{gf-homset} \) 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 $f \in$ extensional $A$
- shows $\{\text{set-dom}=A, \text{set-func}=f, \text{set-cod}=B\} \in \text{hom (set-cat U) A B}$
- using assms by (simp add: set-cat-def hom-def set-arrow-def)

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

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

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

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

**lemma** set-dom-cod-object-subset [intro]:
- assumes $f : f \in \text{ar (set-cat U)}$
- shows $\text{dom (set-cat U) f} \in \text{ob (set-cat U)}$
  - and $\text{cod (set-cat U) f} \in \text{ob (set-cat U)}$
  - and $\text{set-cod f} \subseteq U$
  - and $\text{set-dom f} \subseteq U$
- proof –
  - note [simp] = set-cat-def set-arrow-def
  - have $\text{dom (set-cat U) f} = \text{set-dom f}$ using $f$ by simp
  - also show $\ldots \subseteq U$ using $f$ by simp
  - finally show $\text{dom (set-cat U) f} \in \text{ob (set-cat U)}$ ..
  - have $\text{cod (set-cat U) f} = \text{set-cod f}$ using $f$ by simp
  - also show $\ldots \subseteq U$ using $f$ by simp
  - finally show $\text{cod (set-cat U) f} \in \text{ob (set-cat U)}$ ..
- qed

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

**lemma** set-homE [intro]:
- assumes $f : f \in \text{hom (set-cat U) A B}$
- 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 $1 : f \in \text{ar (set-cat U)}$
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 $f \in (set-dom f) \to (set-cod f)$
  using $f$ by (auto simp add: set-comp-def set-id-def)
  thus set-func $f \in A \to B$
    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 $U (set-cod f) \circ f = f$
proof -
  from $f \in ar (set-cat U)$ have set-cod $f \subseteq U$ ..
  hence 1: set-id $U (set-cod f) \circ f =$
        (set-dom $f$ = set-dom $f$,
         set-func = compose (set-dom $f$) ($\lambda x \in set-cod f. x$) (set-func $f$),
         set-cod = set-cod $f$)
  using $f$ by (simp add: set-comp-def set-id-def)
  have 2: compose (set-dom $f$) ($\lambda x \in set-cod f. x$) (set-func $f$) = set-func $f$
    using $f$ by (rule extensionality)
  show compose (set-dom $f$) ($\lambda x \in set-cod f. x$) (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 $f : (set-dom f) \to (set-cod f)$
  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 funset-mem)
show compose (set-dom $f$) ($\lambda x \in set-cod f. x$) (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 $U (set-cod f) \circ f =$
  (set-dom $f$ = set-dom $f$,
set-func = set-func f,
set-cod = set-cod f
\]
by (simp only: 2)
also have \ldots = f
by simp
finally show ?thesis.
qed

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 add: set-comp-def set-id-def)
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 \ldots = 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 ∈ ob (set-cat U)) have 1: A ⊆ U ..
hence id (set-cat U) A = {set-dom=A, set-func=λx∈A. x, set-cod=A}
  by (simp add: set-cat-def set-id-def)
also have .. ∈ hom (set-cat U) A A
proof (rule set-homI)
  show (λx∈A. x) ∈ A → A
    by (rule funcsetI, auto)
  show (λx∈A. x) ∈ extensional A
    by (rule restrict-extensional)
qed (rule 1, rule 1)
finally show ?thesis.
qed

lemma set-comp-types:
  comp (set-cat U) ∈ hom (set-cat U) B C → hom (set-cat U) A B → hom (set-cat U) A C
proof (rule funcsetI)
  fix g
  assume g-BC: g ∈ hom (set-cat U) B C
  hence comp-cod: set-cod g = C ..
  show comp (set-cat U) g ∈ hom (set-cat U) A B → hom (set-cat U) A C
    by (simp add: set-cat-def set-comp-def comp-cod comp-dom)
  qed (rule funcsetI)
  fix f
  assume f-AB: f ∈ hom (set-cat U) A B
  hence comp-dom: set-dom f = A ..
  show comp (set-cat U) g f ∈ hom (set-cat U) A C
  proof
    have comp (set-cat U) g f =
      
        | set-dom = A,
        | set-func = compose (set-dom f) (set-func g) (set-func f),
        | set-cod = C

      |
    by (simp add: set-cat-def set-comp-def comp-cod comp-dom)
  also have .. ∈ hom (set-cat U) A C
  proof (rule set-homI)
    from f-AB show A ⊆ U ..
    from g-BC show C ⊆ U ..
    from f-AB have fs-f: set-func f: A → B ..
    from g-BC have fs-g: set-func g: B → C ..
    from fs-g and fs-f
    show compose (set-dom f) (set-func g) (set-func f) : A → C
      by (simp only: comp-dom) (rule funcset-compose)
    show compose (set-dom f) (set-func g) (set-func f) ∈ 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-domain h) (set-function f) (compose (set-domain h) (set-function g) (set-function h)) =
    compose (set-domain h) (compose (set-domain g) (set-function f) (set-function g)) (set-function h)
  proof (rule compose-assoc)
    show set-function h ∈ set-domain h → set-domain 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 = 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)
```

9
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 ('o1,'a1,'o2,'a2) functor =
   om :: 'o1 ⇒ 'o2
   am :: 'a1 ⇒ 'a2
abbreviation om-syn (- o [81]) where
   F o ≡ om F
abbreviation am-syn (- a [81]) where
   F a ≡ am F
locale two-cats = AA?: category AA + BB?: category BB
   for AA :: ('o1,'a1,'m1)category-scheme (structure)
   and BB :: ('o2,'a2,'m2)category-scheme (structure) +
fixes preserves-dom :: ('o1,'a1,'o2,'a2)functor ⇒ bool
   and preserves-cod :: ('o1,'a1,'o2,'a2)functor ⇒ bool
   and preserves-id :: ('o1,'a1,'o2,'a2)functor ⇒ bool
   and preserves-comp :: ('o1,'a1,'o2,'a2)functor ⇒ bool
defines preserves-dom G ≡ ∀ f∈Ar AA. G o (Dom AA f) = Dom BB (G a f)
   and preserves-cod G ≡ ∀ f∈Ar AA. G o (Cod AA f) = Cod BB (G a f)
   and preserves-id G ≡ ∀ A∈Ob AA. G a (Id AA A) = Id BB (G o A)
   and preserves-comp G ≡ ∀ f∈Ar AA. ∀ g∈Ar AA. Cod AA f = Dom AA g −→ G a (g · AA f) = (G a g) · BB (G a f)
locale functor = two-cats +
fixes F (structure)
assumes F-preserves-arrows: F a : Ar AA → Ar BB
   and F-preserves-objects: F o : Ob AA → Ob BB
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 $F$

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 = preserves-dom-def preserves-cod-def preserves-id-def preserves-comp-def

end

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

abbreviation
Functor (Functor - : - → - [81]) where
Functor $F$ : $AA$ → $BB$ ≡ functor $AA$ $BB$ $F$

3.2 Simple Lemmas

For example:

lemma (in functor) Functor $F$ : $AA$ → $BB$ ..

lemma functors-preserve-arrows [intro]:
assumes Functor $F$ : $AA$ → $BB$
and $f$ ∈ ar $AA$
shows $F_a f$ ∈ ar $BB$
proof −
from (Functor $F$ : $AA$ → $BB$)
have $F_a :$ ar $AA$ → ar $BB$
by (simp add: functor-def functor-axioms-def)
from this and ($f$ ∈ ar $AA$)
show $?thesis$ by (rule funcsset-mem)
qed

lemma (in functor) functors-preserve-homsets:
assumes 1: $A$ ∈ Ob $AA$
and 2: $B$ ∈ Ob $AA$
and 3: $f$ ∈ Hom $AA$ $A$ $B$
shows $F_a f$ ∈ Hom $BB$ ($F_o A$) ($F_o B$)
proof −
from 3
have 4: $f$ ∈ Ar
by (simp add: hom-def)
with $F$-preserves-arrows
have 5: $F_a f$ ∈ Ar $BB$
by (rule funcsset-mem)
from 4 and \( F \)-preserves-dom
have \( \text{Dom}_{BB} (F_a f) = F_o (\text{Dom}_{AA} f) \)
by \((\text{simp add: preserves-dom-def})\)
also from 3 have \( \ldots = F_o A \)
by \((\text{simp add: hom-def})\)
finally have 6: \( \text{Dom}_{BB} (F_a f) = F_o A \).
from 4 and \( F \)-preserves-cod
have \( \text{Cod}_{BB} (F_a f) = F_o (\text{Cod}_{AA} f) \)
by \((\text{simp add: preserves-cod-def})\)
also from 3 have \( \ldots = F_o B \)
by \((\text{simp add: hom-def})\)
finally have 7: \( \text{Cod}_{BB} (F_a f) = F_o B \).
from 5 and 6 and 7
show ?thesis
by \((\text{simp add: hom-def})\)
qed

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

3.3 Identity Functor

definition
\( \text{id-func} :: (\text{′o,′a,′m}) \text{category-scheme } \Rightarrow (\text{′o,′a,′o,′a}) \text{ functor where} \)
\( \text{id-func } CC = (\text{om}= (\lambda A \in \text{ob } CC. A), \text{am}= (\lambda f \in \text{ar } CC. f)) \)

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

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

lemma (in one-cat) id-func-preserves-objects:
shows \( (id\text{-func } AA)_a : \text{Ob} \to \text{Ob} \)
by \((\text{unfold id-func-def, rule funcsetI, simp})\)
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)₀ (Dom f) = Dom f
    by (simp add: id-func-def) auto
  have (id-func AA)₀ f = f
    using f by (simp add: id-func-def)
  hence rhs: Dom (id-func AA)₀ f = Dom f
    by simp
  from lhs and rhs show (id-func AA)₀ (Dom f) = Dom (id-func AA)₀ 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)₀ (Cod f) = Cod f
    by (simp add: id-func-def) auto
  have (id-func AA)₀ f = f
    using f by (simp add: id-func-def)
  hence rhs: Cod (id-func AA)₀ f = Cod f
    by simp
  from lhs and rhs show (id-func AA)₀ (Cod f) = Cod (id-func AA)₀ 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)₀ (Id A) = Id A
    by (simp add: id-func-def) auto
  have (id-func AA)₀ A = A
    using A by (simp add: id-func-def)
  hence rhs: Id ((id-func AA)₀ A) = Id A
    by simp
  from lhs and rhs show (id-func AA)₀ (Id A) = Id ((id-func AA)₀ A)
    by simp
qed
lemma (in one-cat) id-func-preserves-comp:
  unfolding presieves-comp (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 :: ('a,'a,'m)category-scheme (structure)
  and BB (structure) +
  fixes U and Set
  defines U ≡ (UNIV::'a set)
  defines Set ≡ set-cat U

14
assumes BB-Set: BB = Set
fixes homf (\(\text{Hom}(\cdot,\cdot)\))
defines homf A \equiv \emptyset
om = (\lambda B \in \text{Ob}. \text{Hom} A B),
am = (\lambda f \in \text{Ar}. \{\text{set-dom} = \text{Hom} A (\text{Dom} f), \text{set-func} = (\lambda g \in \text{Hom} A (\text{Dom} f). f \cdot g), \text{set-cod} = \text{Hom} A (\text{Cod} f)\})\)

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

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

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

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

lemma (in into-set) \text{homf-preserves-id}:
assumes \( B \): \( B \in \text{Ob} \)
shows \( \text{Hom}(A,-) \circ (\text{Id } B) = \text{id } \text{Set} \circ (\text{Hom}(A,-) \circ B) \)
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 \( \text{Hom}(A,-) \circ (\text{Id } B) = \emptyset \)
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 \( \ldots = \text{id } \text{Set} \circ (\text{Hom}(A,-) \circ B) \)
using \( B \) by \( \text{simp add: Set-def U-def set-cat-def set-id-def homf-def} \)
finally show \(?thesis\).
qed

lemma (in into-set) \text{homf-preserves-comp}:
assumes \( f \): \( f \in \text{Ar} \)
and \( g \): \( g \in \text{Ar} \)
and \( fg \): \( \text{Cod } f = \text{Dom } g \)
shows \( \text{Hom}(A,-) \circ (g \cdot f) = (\text{Hom}(A,-) \circ g) \cdot (\text{Hom}(A,-) \circ f) \)
proof –
have 1: \( g \cdot f \in \text{Ar} \) using \( \text{assms} \) ..
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 \(\text{lhs}: \Hom_{A\cdot} (g \cdot f) = \emptyset\):
  \begin{align*}
  \text{set-dom} &= \Hom A (\text{Dom } f), \\
  \text{set-func} &= (\lambda h \in \Hom A (\text{Dom } f). (g \cdot f) \cdot h), \\
  \text{set-cod} &= \Hom A (\text{Cod } g)
  \end{align*}

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

have 4: set-dom \((\Hom_{A\cdot} g) \circ (\Hom_{A\cdot} f)\) = \Hom A (\text{Dom } f)

using \(f\) by (simp add: set-comp-def homf-def)

have 5: set-cod \((\Hom_{A\cdot} g) \circ (\Hom_{A\cdot} f)\) = \Hom A (\text{Cod } g)

using \(g\) by (simp add: set-comp-def homf-def)

have set-func \((\Hom_{A\cdot} g) \circ (\Hom_{A\cdot} f)\)
  = compose (\Hom A (\text{Dom } f)) (\lambda y \in \Hom A (\text{Dom } g). g \cdot y) (\lambda x \in \Hom A (\text{Dom } f). f \cdot x)

using \(f g\) by (simp add: set-comp-def homf-def)

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

  rule extensionalityI,
  rule compose-extensional,
  rule restrict-extensional,
  simp)

fix \(h\)

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

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

proof–

  from 10 have \(h \in A\cdot r\) by (simp add: hom-def)

have 100: \((\cdot) : \Hom (\text{Dom } f) (\text{Dom } g) \rightarrow \Hom A (\text{Dom } f) \rightarrow \Hom A (\text{Dom } g)\)

  by (rule AA.comp-types)

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

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

from 100 and 101

have \((\cdot) : \Hom A (\text{Dom } f) \rightarrow \Hom A (\text{Dom } g)\)

  by (rule funcset-mem)

from this and 10

show \(f \cdot h \in \Hom A (\text{Dom } g)\)

  by (rule funcset-mem)

qed

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

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

and \(f \cdot h \in A\cdot r\)

by (simp-all add: hom-def)

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

using \(f g fg\) 10 by (simp add: compose-def 10 11 hom-def)

qed

finally have 6: set-func \((\Hom_{A\cdot} g) \circ (\Hom_{A\cdot} f)\)
  = \((\lambda h \in \Hom A (\text{Dom } f). (g \cdot f) \cdot h)\).
from 4 and 5 and 6
have rhs: \((\text{Hom}(A,-)_a g) \circ (\text{Hom}(A,-)_a f) = \emptyset\)
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 \(\text{Hom}(A,-) : AA \rightarrow \text{Set}\)
proof (intro functor.intro functor-axioms.intro)
show \(\text{Hom}(A,-)_a : \text{Ar} \rightarrow \text{ar} \text{Set}\)
  by (rule homf-preserves-arrows)
show \(\text{Hom}(A,-)_o : \text{Ob} \rightarrow \text{ob} \text{Set}\)
  by (rule homf-preserves-objects)
show \(\forall f \in \text{Ar}. \text{Hom}(A,-)_o (\text{Dom} f) = \text{dom} Set (\text{Hom}(A,-)_a f)\)
  by (intro ballI) (rule homf-preserves-dom)
show \(\forall f \in \text{Ar}. \text{Hom}(A,-)_o (\text{Cod} f) = \text{cod} Set (\text{Hom}(A,-)_a f)\)
  by (intro ballI) (rule homf-preserves-cod)
show \(\forall B \in \text{Ob}. \text{Hom}(A,-)_a (\text{Id} B) = \text{id} Set (\text{Hom}(A,-)_o B)\)
  by (intro ballI) (rule homf-preserves-id)
show \(\forall f \in \text{Ar}. \forall g \in \text{Ar}. \text{Cod} f = \text{Dom} g \rightarrow \text{Hom}(A,-)_a (g \cdot f) = \text{comp} Set (\text{Hom}(A,-)_a g) (\text{Hom}(A,-)_a f)\)
  by (intro ballI impI, simp add: Set-def set-cat-def) (rule homf-preserves-comp)
show two-cats \(AA \rightarrow \text{Set}\)
proof intro-locales
  show category Set
  by (unfold Set-def, rule set-cat-cat)
qed
qed
end

5 Natural Transformations

theory NatTrans
imports Functors
begin

locale natural-transformation = two-cats +
fixes \(F\) and \(G\) and \(u\)
assumes Functor \(F : AA \rightarrow BB\)
and Functor \(G : AA \rightarrow BB\)
and \(u : \text{ob} AA \rightarrow \text{ar} BB\)
and \( u \in \text{extensional} \) (\( \text{ob} AA \))
and \( \forall A \in \text{Ob} \). \( u A \in \text{Hom}_{BB}(F A) \) \( (G A) \)
and \( \forall A \in \text{Ob}. \forall B \in \text{Ob}. \forall f \in \text{Hom} A B. (G_a f) \cdot_{BB} (u A) = (u B) \cdot_{BB} (F A f) \)

abbreviation
\( \text{nt-syn} \) \( (- : - \Rightarrow - \in \text{Func} \text{'}\text{'}) \) \( [81] \) where
\( u : F \Rightarrow G \) 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 \( \text{endoNT} \)) \( \text{id-restrict-natural} \):
\( (\lambda A \in \text{Ob}. \text{Id} A) : (\text{id-func } AA) \Rightarrow (\text{id-func } AA) \) in \( \text{Func}(AA, AA) \)

proof (intro \( \text{natural-transformation.intro natural-transformation-axioms.intro two-cats.intro ballI} \))
show \( (\lambda A \in \text{Ob}. \text{Id} A) : \text{Ob} \Rightarrow \text{Ar} \)
by \( \text{(rule funcsetI)}.\) \( \text{auto} \)
show \( (\lambda A \in \text{Ob}. \text{Id} A) \) \( \in \text{extensional} \) (\( \text{Ob} \))
by \( \text{(rule restrict-extensional)} \)
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 \( \text{(simp add: id-func-def)} \)
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 \( \text{(simp-all add: hom-def)} \)
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 \( \text{(simp add: id-func-def)} \)
qed \( \text{(auto intro: id-func-functor, unfold-locales, unfold-locales)} \)

end

6 Yoneda Lemma

theory Yoneda
imports HomFunctors NatTrans
begin

6.1 The Sandwich Natural Transformation

locale Yoneda = functor + into-set +
assumes \( \text{TERM} \) \( (AA :: \langle \text{'a}, \text{'a}, \text{'m} \rangle \text{category-scheme}) \)
fixes sandwich :: \( \langle \text{'a}, \text{'a}, \text{'o} \rangle \Rightarrow \text{'a set-arrow} \) \( (\sigma(\text{'-','-})) \)
defines \( \text{sandwich } A a \equiv (\lambda B \in \text{Ob}. \) \( \langle \text{\} } \)
\textit{set-dom}=\text{Hom }A \, B,
\textit{set-func}=(\lambda f \in \text{Hom }A \, B. \text{set-func} (F_a \, f) \, a),
\textit{set-cod}=F_o \, B

\text{fixes } \text{unsandwich} :: [\sigma^{-'}(\cdot, \cdot)]
\text{defines } \text{unsandwich} \, A \, u \equiv \text{set-func}(u \, A) \, (\text{Id} \, A)

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

\textbf{lemma (in Yoneda) F-comp-func:}
\begin{itemize}
\item[1.] \(A \in \text{Ob}\)
\item[2.] \(B \in \text{Ob}\)
\item[3.] \(C \in \text{Ob}\)
\item[4.] \(g \in \text{Hom }A \, B\)
\item[5.] \(f \in \text{Hom }B \, C\)
\end{itemize}
\item[shows] \(\text{set-func}(F_a \, (f \cdot g)) = \text{compose}(F_o \, A) \, (\text{set-func}(F_a \, f)) \, (\text{set-func}(F_a \, g))\)
\begin{proof}
\text{from 4 and 5 have}
\item[7.] \(\text{Cod } g = \text{Dom } f\)
\item[8.] \(g \in \text{Ar}\)
\item[9.] \(f \in \text{Ar}\)
\item[10.] \(\text{Dom } g = A\)
\text{by (simp-all add: hom-def)}
\text{from F-preserves-dom and 8 and 10 have}
\item[11.] \(\text{set-dom}(F_a \, g) = F_o \, A\)
\text{by (simp add: preserves-dom-def BB-Set Set-def) auto}
\text{from F-preserves-comp and 7 and 8 and 9 have}
\item[12.] \((F_a \, (f \cdot g)) = (F_a \, f) \cdot BB \, (F_a \, g)\)
\text{by (simp add: preserves-comp-def)}
\text{hence set-func} \((F_a \, (f \cdot g)) = \text{set-func} \, ((F_a \, f) \circ (F_a \, g))\)
\text{by (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 (simp add: set-comp-def 11)}
\text{finally show } \text{?thesis}
\end{proof}
\textbf{qed}

\textbf{lemma (in Yoneda) sandwich-funcset:}
\begin{itemize}
\item[\text{assumes}] \(A : A \in \text{Ob}\)
\item[\text{and}] \(a \in F_o \, A\)
\end{itemize}
\item[\text{shows}] \(\sigma(A,a) : \text{Ob} \rightarrow \text{ar Set}\)
\begin{proof} \text{(rule funcsetI)}
\text{fix } B
\text{assume } B : B \in \text{Ob}
\text{thus } \sigma(A,a) \, B \in \text{ar Set}
\end{proof}
proof (simp add: Set-def sandwich-def set-cat-def)
show set-arrow U ≡
  set-dom = Hom A B,
  set-func = λf∈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 ⊆ U and F o B ⊆ U
by (simp-all add: U-def)
show (λf∈Hom A B. set-func (F a f) a) ∈ Hom A B → F o B
proof (rule funcsetI, simp)
fix f
assume f: f ∈ Hom A B
with A B have F a f ∈ Hom B (F o A) (F o B)
  by (rule functors-preserve-homsets)
hence F a f ∈ 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) → (F o B)
  by (simp add: Set-def set-cat-def set-arrow-def)
thus set-func (F a f) a ∈ F o B
using (a ∈ F o A)
  by (rule funcset-mem)
qed
qed
qed

lemma (in Yoneda) sandwich-type:
assumes A: A ∈ Ob and B: B ∈ Ob
and a ∈ F o A
shows σ(A,a) B ∈ hom Set (Hom A B) (F o B)
proof –
  have σ(A,a) ∈ Ob → Ar Set
    using A and (a ∈ F o A) by (rule sandwich-funcset)
hence σ(A,a) B ∈ 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 ∈ Ob and BOb: B ∈ Ob and COb: C ∈ Ob
and aFu: a ∈ F o A
and fBC: f ∈ Hom B C
shows (F_a f) ◦ (σ(A,a) B) = (σ(A,a) C) ◦ (Hom(A-,a) f)
proof –
from \(fBC\) have 1: \(f \in A r\) 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: simp-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 \(AOb\) 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 \((\cdot) : \text{Hom} \ B \ C \rightarrow \text{Hom} \ A \ B \rightarrow \text{Hom} \ A \ C\)
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 \(\text{opfType}\)

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 \$thesis$
  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,-)o B) (F o B)
    using assms by (auto simp add: homf-def)
  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)
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⁻¹(A,σ(A,a)) = a
proof –
  from 1 have Id A ∈ Hom A A ..
  with 1
  have 3: σ⁻¹(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
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_0 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) (F_0 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,-)_a 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,-)_a f)) (Id A)))
  apply (simp add: BB-Set Set-def)
  apply (rule extensionalityI)
  apply (rule restrict-extensional, rule restrict-extensional)
  by simp
have ∀f ∈ Hom A B. set-dom (Hom(A,·)a f) = Hom A A
  by (intro ballI, simp add: homf-def hom-def)

have rootz: ∀f ∈ Hom A B ⇒ set-dom (Hom(A,·)a f) = Hom A A
  by (simp add: homf-def hom-def)

from 1 have rooly: Id A ∈ Hom A A.

have roolx: ∀f. f ∈ Hom A B ⇒ f ∈ Ar
  by (simp add: hom-def)

have roolw: ∀f. f ∈ Hom A B ⇒ Id A ∈ Hom A (Dom f)

proof
  fix f
  assume f ∈ Hom A B
  hence Dom f = A by (simp add: hom-def)
  thus Id A ∈ Hom A (Dom f)
    by (simp add: rooly)
qed

have annoying: ∀f. f ∈ Hom A B ⇒ Id A = Id (Dom f)
  by (simp add: hom-def)

have (λf∈Hom A B. (set-func ((u B) ⊙ (Hom(A,·))) (Id A))
  = (λf∈Hom A B. (compose (Hom A A) (set-func (u B)) (set-func (Hom(A,·))) f)) (Id A))
  apply (rule extensionalityI)
  apply (rule restrict-extensional, rule restrict-extensional)
  by (simp add: compose-def set-comp-def roolz roolx roolw)

also have ... = (λf∈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 roolz roolx)
  apply (simp only: annoying)
  apply (simp add: roolx id-right)
  done

finally have four:
  (λf∈Hom A B. (set-func ((u B) ⊙ (Hom(A,·))) f)) (Id A))
  from 2 and 3

have uBhom: u B ∈ hom Set (Hom(A,·)o B) (F 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) = F 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-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_0 A ≡ { u. u : Hom(A,-) ⇒ F in Func(AA,Set) }
unfolding equinumerous-def bij-betw-eq inj-on-def
proof (intro exI conjI beI ballI impI)
  — Sandwich is injective
  fix x and y
  assume 2: x ∈ F_0 A and 3: y ∈ F_0 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 \( u \)
assumee \( u \in \{ y. y : \text{Hom}(A,-) \Rightarrow F \text{ in Func } (AA,Set) \} \)
hence \( 2: u : \text{Hom}(A,-) \Rightarrow F \text{ in Func } (AA,Set) \)
by simp
with \( 1 \) show \( \sigma(A,\sigma^-(A,u)) = u \)
by (rule unsandwich-right-inverse)

Sandwich is into \( F A \)

from \( 1 \) and \( 2 \)
have \( u A \in \text{hom Set } (\text{Hom } A A) (F_O A) \)
by (simp add: natural-transformation-def natural-transformation-axioms-def homf-def)
hence \( u A \in \text{ar Set and dom Set } (u A) = \text{Hom } A A \text{ and cod Set } (u A) = F_O A \)
by (simp-all add: hom-def)
hence \( uAfuncset: \text{set-func } (u A) : (\text{Hom } A A) \Rightarrow (F_O A) \)
by (simp add: Set-def set-cat-def set-arrow-def)
from \( 1 \) have \( \text{Id } A \in \text{Hom } A A \) ..
with \( uAfuncset \)
show \( \sigma^+(A,u) \in F_O A \)
by (simp add: unsandwich-def, rule funcset-mem)

next
fix \( x \)
assume \( x \in F_O A \)
with \( 1 \) have \( \sigma(A,x) : \text{Hom}(A,-) \Rightarrow F \text{ in Func } (AA,Set) \)
by (rule sandwich-natural)
thus \( \sigma(A,x) \in \{ y. y : \text{Hom}(A,-) \Rightarrow F \text{ in Func } (AA,Set) \} \)
by simp
qed

end

References