Category Theory to Yoneda's Lemma

Greg O'Keefe

March 17, 2025

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

theory *Cat*
imports *HOL−Library.FuncSet*
begin

1.1 Definitions

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

definition
hom :: [(*'o*,*'a*,*'m*) *category-scheme*, *'o*, *'o*] ⇒ *'a set*
(‹*Hom*ı - -› [*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 *Dom f* = *A* **using** *assms* **by** (*simp add*: *hom-def*)
  show *Cod f* = *B* **using** *assms* **by** (*simp add*: *hom-def*)
qed

lemma (**in** *category*) *id-arrow* [*intro*]:
  assumes *A* ∈ *Ob*
  shows *Id A* ∈ *Ar*
proof−
  from ‹*A* ∈ *Ob*› **have** *Id A* ∈ *Hom A A* **by** (*rule id-hom*)
  thus *Id A* ∈ *Ar* **by** (*simp add*: *hom-def*)
qed

lemma (**in** *category*) *id-dom-cod*:
  assumes *A* ∈ *Ob*
  shows *Dom* (*Id A*) = *A* **and** *Cod* (*Id A*) = *A*
proof−
  from ‹*A* ∈ *Ob*› **have** *1*: *Id A* ∈ *Hom A A* **..**
  then show *Dom* (*Id A*) = *A* **and** *Cod* (*Id A*) = *A*
    by (*simp-all add*: *hom-def*)
qed

lemma (**in** *category*) *compI* [*intro*]:
  assumes *f*: *f* ∈ *Ar* **and** *g*: *g* ∈ *Ar* **and** *Cod f* = *Dom g*
  shows *g* · *f* ∈ *Ar*
  and *Dom* (*g* · *f*) = *Dom f*
  and *Cod* (*g* · *f*) = *Cod g*
proof−
  have *f* ∈ *Hom* (*Dom f*) (*Cod f*) **using** *f* **by** (*simp add*: *hom-def*)
  with ‹*Cod f* = *Dom g*› **have** *f-homset*: *f* ∈ *Hom* (*Dom f*) (*Dom g*) **by** *simp*
  have *g-homset*: *g* ∈ *Hom* (*Dom g*) (*Cod g*) **using** *g* **by** (*simp add*: *hom-def*)
  have (·) : *Hom* (*Dom g*) (*Cod g*) → *Hom* (*Dom f*) (*Dom g*) → *Hom* (*Dom f*) (*Cod g*) **..**
  from *this* **and** *g-homset*
  have (·) *g* ∈ *Hom* (*Dom f*) (*Dom g*) → *Hom* (*Dom f*) (*Cod g*)
    by (*rule funcset-mem*)
  from *this* **and** *f-homset*
  have *gf-homset*: *g* · *f* ∈ *Hom* (*Dom f*) (*Cod g*)
    by (*rule funcset-mem*)
  thus *g* · *f* ∈ *Ar*
    by (*simp add*: *hom-def*)
  from *gf-homset* **show** *Dom* (*g* · *f*) = *Dom f* **and** *Cod* (*g* · *f*) = *Cod g*
    by (*simp-all add*: *hom-def*)
qed

end

2 Set is a Category

theory *SetCat*
imports *Cat*
begin

2.1 Definitions

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

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

definition
  set-id :: $['c\ set,\ 'c\ set] \Rightarrow 'c\ set\text{-}arrow$ **where**
  $set\text{-}id\ U = (\lambda s{\in}Pow\ U.\ (\!|set\text{-}dom{=}s,\ set\text{-}func{=}\lambda x{\in}s.\ x,\ set\text{-}cod{=}s|\!))$

definition
  set-comp :: $['c\ set\text{-}arrow,\ 'c\ set\text{-}arrow] \Rightarrow 'c\ set\text{-}arrow$ (**infix** ‹⊙› *70*) **where**
  $set\text{-}comp\ g\ f =$
  $(\!|$
    $set\text{-}dom = set\text{-}dom\ f,$
    $set\text{-}func = compose\ (set\text{-}dom\ f)\ (set\text{-}func\ g)\ (set\text{-}func\ f),$
    $set\text{-}cod = set\text{-}cod\ g$
  $|\!)$

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

2.2 Simple Rules and Lemmas

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

lemma *set-objectE* [*intro*]: $A \in ob\ (set\text{-}cat\ U) \Longrightarrow A \subseteq U$

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

lemma *set-homI* [*intro*]:
assumes $A \subseteq U$
and $B \subseteq U$
and $f : A \rightarrow B$
and $f \in$ *extensional* A
shows $(\!|$*set-dom*$=A$, *set-func*$=f$, *set-cod*$=B|\!) \in$ *hom* (*set-cat* U) A B
using *assms* **by** (*simp add*: *set-cat-def hom-def set-arrow-def*)

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

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

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

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

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

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

lemma *set-homE* [*intro*]:
assumes f: $f \in$ *hom* (*set-cat* U) A B
shows $A \subseteq U$
and $B \subseteq U$
and *set-dom* $f = A$
and *set-func* $f : A \rightarrow B$
and *set-cod* $f = B$
proof−
have *1*: $f \in$ *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* ⊆ *U* **..**
thus *A* ⊆ *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* ⊆ *U* **..**
thus *B* ⊆ *U* **by** (*simp add*: *3*)
have *set-func f* ∈ (*set-dom f*) → (*set-cod f*)
using *f* **by** (*auto simp add*: *set-cat-def hom-def set-arrow-def*)
thus *set-func f* ∈ *A* → *B*
by (*simp add*: *2 3*)
qed

2.3 Set is a Category

lemma *set-id-left*:
assumes *f*: *f* ∈ *ar* (*set-cat U*)
shows *set-id U* (*set-cod f*) ⊙ *f* = *f*
proof−
from ‹*f* ∈ *ar* (*set-cat U*)› **have** *set-cod f* ⊆ *U* **..**
hence *1*: *set-id U* (*set-cod f*) ⊙ *f* =
⦇
set-dom=*set-dom f*,
set-func=*compose* (*set-dom f*) (λ*x*∈*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*) (λ*x*∈*set-cod f*. *x*) (*set-func f*) = *set-func f*
proof (*rule extensionalityI*)
show *compose* (*set-dom f*) (λ*x*∈*set-cod f*. *x*) (*set-func f*) ∈ *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*
have *f-into-cod*: *set-func f* : (*set-dom f*) → (*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* ∈ *set-cod f*
by (*rule funcset-mem*)
show *compose* (*set-dom f*) (λ*x*∈*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*) ⊙ *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-right*:
assumes f: $f \in$ *ar* (*set-cat* U)
shows $f \odot$ (*set-id* U (*set-dom* f)) = f
proof−
from ‹$f \in$ *ar* (*set-cat* U)› **have** *set-dom* $f \subseteq U$ **..**
hence *1*: $f \odot$ (*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) $\in$ *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
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** $f \odot$ (*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 $A \in$ *ob* (*set-cat* U)
shows *id* (*set-cat* U) $A \in$ *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
  proof (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
```

qed
qed

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

lemma *set-comp-associative*:
fixes *f* **and** *g* **and** *h*
assumes *f*: $f \in ar$ (*set-cat U*)
and *g*: $g \in ar$ (*set-cat U*)
and *h*: $h \in ar$ (*set-cat U*)
and *hg*: *cod* (*set-cat U*) *h* = *dom* (*set-cat U*) *g*
and *gf*: *cod* (*set-cat U*) *g* = *dom* (*set-cat U*) *f*
shows *comp* (*set-cat U*) *f* (*comp* (*set-cat U*) *g h*) =
comp (*set-cat U*) (*comp* (*set-cat U*) *f g*) *h*
proof (*simp add*: *set-cat-def set-comp-def*)
show *compose* (*set-dom h*) (*set-func f*) (*compose* (*set-dom h*) (*set-func g*) (*set-func h*)) =
compose (*set-dom h*) (*compose* (*set-dom g*) (*set-func f*) (*set-func g*)) (*set-func h*)
proof (*rule compose-assoc*)
show *set-func h* $\in$ *set-dom h* $\to$ *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*)
proof (*rule category.intro*)
fix *f*
assume *f*: $f \in ar$ (*set-cat U*)
show *dom* (*set-cat U*) $f \in ob$ (*set-cat U*) **using** *f* **..**
show *cod* (*set-cat U*) $f \in ob$ (*set-cat U*) **using** *f* **..**
show *comp* (*set-cat U*) (*id* (*set-cat U*) (*cod* (*set-cat U*) *f*)) *f* = *f*
using *f* **by** (*simp add*: *set-id-left*)
show *comp* (*set-cat U*) *f* (*id* (*set-cat U*) (*dom* (*set-cat U*) *f*)) = *f*
using *f* **by** (*simp add*: *set-id-right*)
next
fix *A*
assume $A \in ob$ (*set-cat U*)
then show *id* (*set-cat U*) $A \in hom$ (*set-cat U*) *A A*
by (*rule set-id-hom*)
next
fix *A* **and** *B* **and** *C*
show *comp* (*set-cat U*) $\in$ *hom* (*set-cat U*) *B C* $\to$ *hom* (*set-cat U*) *A B* $\to$ *hom* (*set-cat U*) *A C*
by (*rule set-comp-types*)
next
fix *f* **and** *g* **and** *h*
assume $f \in ar$ (*set-cat U*)

and $g \in ar\ (set\text{-}cat\ U)$
and $h \in ar\ (set\text{-}cat\ U)$
and $cod\ (set\text{-}cat\ U)\ h = dom\ (set\text{-}cat\ U)\ g$
and $cod\ (set\text{-}cat\ U)\ g = dom\ (set\text{-}cat\ U)\ f$
then show $comp\ (set\text{-}cat\ U)\ f\ (comp\ (set\text{-}cat\ U)\ g\ h) =$
$comp\ (set\text{-}cat\ U)\ (comp\ (set\text{-}cat\ U)\ f\ g)\ h$
by (*rule set-comp-associative*)
qed

end

3 Functors

theory *Functors*
imports *Cat*
begin

3.1 Definitions

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

abbreviation
om-syn (‹- $_{\mathrm{o}}$› [*81*]) **where**
$F_{\mathrm{o}} \equiv om\ F$

abbreviation
am-syn (‹- $_{\mathrm{a}}$› [*81*]) **where**
$F_{\mathrm{a}} \equiv am\ F$

locale *two-cats* = *AA?*: *category AA* + *BB?*: *category BB*
for $AA :: ('o1,'a1,'m1)category\text{-}scheme$ (**structure**)
and $BB :: ('o2,'a2,'m2)category\text{-}scheme$ (**structure**) +
fixes $preserves\text{-}dom :: ('o1,'a1,'o2,'a2)functor \Rightarrow bool$
and $preserves\text{-}cod :: ('o1,'a1,'o2,'a2)functor \Rightarrow bool$
and $preserves\text{-}id :: ('o1,'a1,'o2,'a2)functor \Rightarrow bool$
and $preserves\text{-}comp :: ('o1,'a1,'o2,'a2)functor \Rightarrow bool$
defines $preserves\text{-}dom\ G \equiv \forall f{\in}Ar_{AA}.\ G_{\mathrm{o}}\ (Dom_{AA}\ f) = Dom_{BB}\ (G_{\mathrm{a}}\ f)$
and $preserves\text{-}cod\ G \equiv \forall f{\in}Ar_{AA}.\ G_{\mathrm{o}}\ (Cod_{AA}\ f) = Cod_{BB}\ (G_{\mathrm{a}}\ f)$
and $preserves\text{-}id\ G \equiv \forall A{\in}Ob_{AA}.\ G_{\mathrm{a}}\ (Id_{AA}\ A) = Id_{BB}\ (G_{\mathrm{o}}\ A)$
and $preserves\text{-}comp\ G \equiv$
$\forall f{\in}Ar_{AA}.\ \forall g{\in}Ar_{AA}.\ Cod_{AA}\ f = Dom_{AA}\ g \longrightarrow G_{\mathrm{a}}\ (g \cdot_{AA} f) = (G_{\mathrm{a}}\ g)$
$\cdot_{BB}\ (G_{\mathrm{a}}\ f)$

locale *functor* = *two-cats* +
fixes *F* (**structure**)
assumes *F-preserves-arrows*: $F_{\mathrm{a}} : Ar_{AA} \rightarrow Ar_{BB}$
and *F-preserves-objects*: $F_{\mathrm{o}} : Ob_{AA} \rightarrow Ob_{BB}$

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* - : - $\longrightarrow$ -› [*81*]) **where**
Functor F : *AA* $\longrightarrow$ *BB* $\equiv$ *functor AA BB F*

3.2 Simple Lemmas

For example:

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

lemma *functors-preserve-arrows* [*intro*]:
assumes *Functor F* : *AA* $\longrightarrow$ *BB*
and $f \in$ *ar AA*
shows $F_{\mathrm{a}}\, f \in$ *ar BB*
proof−
from ‹*Functor F* : *AA* $\longrightarrow$ *BB*›
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 funcset-mem*)
qed

lemma (**in** *functor*) *functors-preserve-homsets*:
assumes *1*: $A \in Ob_{AA}$
and *2*: $B \in Ob_{AA}$
and *3*: $f \in Hom_{AA}\ A\ B$
shows $F_{\mathrm{a}}\, f \in Hom_{BB}\ (F_{\mathrm{o}}\ A)\ (F_{\mathrm{o}}\ B)$
proof−
from *3*
have *4*: $f \in Ar$
by (*simp add*: *hom-def*)
with *F-preserves-arrows*
have *5*: $F_{\mathrm{a}}\, f \in Ar_{BB}$

by (*rule funcset-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** ... = F_o *A*
by (*simp add*: *hom-def*)
finally have *6*: Dom_{BB} (F_a *f*) = F_o *A* **.**
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** ... = F_o *B*
by (*simp add*: *hom-def*)
finally have *7*: Cod_{BB} (F_a *f*) = F_o *B* **.**
from *5* **and** *6* **and** *7*
show *?thesis*
by (*simp add*: *hom-def*)
qed

lemma *functors-preserve-objects* [*intro*]:
assumes *Functor F* : *AA* ⟶ *BB*
and *A* ∈ *ob AA*
shows F_o *A* ∈ *ob BB*
proof−
from ‹*Functor F* : *AA* ⟶ *BB*›
have F_o : *ob AA* → *ob BB*
by (*simp add*: *functor-def functor-axioms-def*)
from *this* **and** ‹*A* ∈ *ob AA*›
show *?thesis* **by** (*rule funcset-mem*)
qed

3.3 Identity Functor

definition
id-func :: (*'o*,*'a*,*'m*) *category-scheme* ⇒ (*'o*,*'a*,*'o*,*'a*) *functor* **where**
id-func CC = ⦇*om*=(λ*A*∈*ob CC*. *A*), *am*=(λ*f*∈*ar CC*. *f*)⦈

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

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

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

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

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

lemma (**in** *one-cat*) *id-func-preserves-id*:
  preserves-id (*id-func AA*)
unfolding *preserves-id-def endo*
proof
  fix *A*
  assume *A*: $A \in Ob$
  hence *lhs*: $(id\text{-}func\ AA)_{\mathrm{a}}\ (Id\ A) = Id\ A$
    by (*simp add*: *id-func-def*) *auto*
  have $(id\text{-}func\ AA)_{\mathrm{o}}\ A = A$
    using *A* **by** (*simp add*: *id-func-def*)
  hence *rhs*: $Id\ ((id\text{-}func\ AA)_{\mathrm{o}}\ A) = Id\ A$
    by *simp*
  from *lhs* **and** *rhs* **show** $(id\text{-}func\ AA)_{\mathrm{a}}\ (Id\ A) = Id\ ((id\text{-}func\ AA)_{\mathrm{o}}\ A)$
    by *simp*
qed

lemma (**in** *one-cat*) *id-func-preserves-comp*:
  preserves-comp (*id-func* AA)
unfolding *preserves-comp-def endo*
proof (*intro ballI impI*)
  fix f **and** g
  assume f: $f \in Ar$ **and** g: $g \in Ar$ **and** $Cod\ f = Dom\ g$
  then have $g \cdot f \in Ar$ **..**
  hence *lhs*: $(id\text{-}func\ AA)_{\mathrm{a}}\ (g \cdot f) = g \cdot f$
    by (*simp add*: *id-func-def*)
  have *id-f*: $(id\text{-}func\ AA)_{\mathrm{a}}\ f = f$
    using f **by** (*simp add*: *id-func-def*)
  have *id-g*: $(id\text{-}func\ AA)_{\mathrm{a}}\ g = g$
    using g **by** (*simp add*: *id-func-def*)
  hence *rhs*: $(id\text{-}func\ AA)_{\mathrm{a}}\ g \cdot (id\text{-}func\ AA)_{\mathrm{a}}\ f = g \cdot f$
    by (*simp add*: *id-f id-g*)
  from *lhs* **and** *rhs*
  show $(id\text{-}func\ AA)_{\mathrm{a}}\ (g \cdot f) = (id\text{-}func\ AA)_{\mathrm{a}}\ g \cdot (id\text{-}func\ AA)_{\mathrm{a}}\ f$
    by *simp*
qed

theorem (**in** *one-cat*) *id-func-functor*:
  Functor (*id-func* AA) : $AA \longrightarrow AA$
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 $U \equiv (UNIV{::}'a\ set)$

defines *Set* ≡ *set-cat U*
assumes *BB-Set*: *BB* = *Set*
fixes *homf* (‹*Hom*′(-,′-′)›)
defines *homf A* ≡ ⦇
om = (λ*B*∈*Ob*. *Hom A B*),
am = (λ*f*∈*Ar*. ⦇*set-dom*=*Hom A* (*Dom f*),*set-func*=(λ*g*∈*Hom A* (*Dom f*). *f* • *g*),*set-cod*=*Hom A* (*Cod f*)⦈)
⦈

lemma (**in** *into-set*) *homf-preserves-arrows*:
$Hom(A,\text{-})_{\mathrm{a}}$: *Ar* → *ar Set*
proof (*rule funcsetI*)
fix *f*
assume *f*: *f* ∈ *Ar*
thus $Hom(A,\text{-})_{\mathrm{a}}$ *f* ∈ *ar Set*
proof (*simp add*: *homf-def Set-def set-cat-def set-arrow-def U-def*)
have *1*: (•) : *Hom* (*Dom f*) (*Cod f*) → *Hom A* (*Dom f*) → *Hom A* (*Cod f*) **..**
have *2*: *f* ∈ *Hom* (*Dom f*) (*Cod f*) **using** *f* **by** (*simp add*: *hom-def*)
from *1* **and** *2* **have** *3*: (•) *f* : *Hom A* (*Dom f*) → *Hom A* (*Cod f*)
by (*rule funcset-mem*)
show (λ*g*∈*Hom A* (*Dom f*). *f* • *g*) : *Hom A* (*Dom f*) → *Hom A* (*Cod f*)
proof (*rule funcsetI*)
fix *g*′
assume *g*′ ∈ *Hom A* (*Dom f*)
from *3* **and** *this* **show** (λ*g*∈*Hom A* (*Dom f*). *f* • *g*) *g*′ ∈ *Hom A* (*Cod f*)
by *simp* (*rule funcset-mem*)
qed
qed
qed

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

lemma (**in** *into-set*) *homf-preserves-dom*:
assumes *f*: *f* ∈ *Ar*
shows $Hom(A,\text{-})_{\mathrm{o}}$ (*Dom f*) = *dom Set* ($Hom(A,\text{-})_{\mathrm{a}}$ *f*)
proof−

have *Dom f* $\in$ *Ob* **using** *f* **..**
hence *1*: *Hom*$(A,\text{-})_{\mathrm{o}}$ (*Dom f*) = *Hom A* (*Dom f*)
using *f* **by** (*simp add*: *homf-def*)
have *2*: *dom Set* (*Hom*$(A,\text{-})_{\mathrm{a}}$ *f*) = *Hom A* (*Dom f*)
using *f* **by** (*simp add*: *Set-def homf-def*)
from *1* **and** *2* **show** *?thesis* **by** *simp*
qed

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

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

lemma (**in** *into-set*) *homf-preserves-comp*:
assumes *f*: *f* $\in$ *Ar*
and *g*: *g* $\in$ *Ar*
and *fg*: *Cod f* = *Dom g*
shows *Hom*$(A,\text{-})_{\mathrm{a}}$ (*g* $\cdot$ *f*) = (*Hom*$(A,\text{-})_{\mathrm{a}}$ *g*) $\odot$ (*Hom*$(A,\text{-})_{\mathrm{a}}$ *f*)
proof−

have *1*: $g \cdot f \in Ar$ **using** *assms* **..**
have *2*: $Dom\ (g \cdot f) = Dom\ f$ **using** *f g fg* **..**
have *3*: $Cod\ (g \cdot f) = Cod\ g$ **using** *f g fg* **..**
have *lhs*: $Hom(A,\text{-})_{\mathrm{a}}\ (g \cdot f) = (\!|$
set-dom$=Hom\ A\ (Dom\ f)$,
set-func$=(\lambda h{\in}Hom\ A\ (Dom\ f).\ (g \cdot f) \cdot h)$,
set-cod$=Hom\ A\ (Cod\ g)|\!)$
by (*simp add*: *homf-def 1 2 3*)
have *4*: *set-dom* $((Hom(A,\text{-})_{\mathrm{a}}\ g) \odot (Hom(A,\text{-})_{\mathrm{a}}\ f)) = Hom\ A\ (Dom\ f)$
using *f* **by** (*simp add*: *set-comp-def homf-def*)
have *5*: *set-cod* $((Hom(A,\text{-})_{\mathrm{a}}\ g) \odot (Hom(A,\text{-})_{\mathrm{a}}\ f)) = Hom\ A\ (Cod\ g)$
using *g* **by** (*simp add*: *set-comp-def homf-def*)
have *set-func* $((Hom(A,\text{-})_{\mathrm{a}}\ g) \odot (Hom(A,\text{-})_{\mathrm{a}}\ f))$
$= compose\ (Hom\ A\ (Dom\ f))\ (\lambda y{\in}Hom\ A\ (Dom\ g).\ g \cdot y)\ (\lambda x{\in}Hom\ A\ (Dom\ f).\ f \cdot x)$
using *f g* **by** (*simp add*: *set-comp-def homf-def*)
also have $\ldots = (\lambda h{\in}Hom\ A\ (Dom\ f).\ (g \cdot f) \cdot h)$
proof (
rule extensionalityI,
rule compose-extensional,
rule restrict-extensional,
simp)
fix *h*
assume *10*: $h \in Hom\ A\ (Dom\ f)$
hence *11*: $f \cdot h \in Hom\ A\ (Dom\ g)$
proof−
from *10* **have** $h \in Ar$ **by** (*simp add*: *hom-def*)
have *100*: $(\cdot) : Hom\ (Dom\ f)\ (Dom\ g) \to Hom\ A\ (Dom\ f) \to Hom\ A\ (Dom\ g)$
by (*rule AA.comp-types*)
have $f \in Hom\ (Dom\ f)\ (Cod\ f)$ **using** *f* **by** (*simp add*: *hom-def*)
hence *101*: $f \in Hom\ (Dom\ f)\ (Dom\ g)$ **using** *fg* **by** *simp*
from *100* **and** *101*
have $(\cdot)\ f : Hom\ A\ (Dom\ f) \to Hom\ A\ (Dom\ g)$
by (*rule funcset-mem*)
from *this* **and** *10*
show $f \cdot h \in Hom\ A\ (Dom\ g)$
by (*rule funcset-mem*)
qed
hence $Cod\ (f \cdot h) = Dom\ g$
and $Dom\ (f \cdot h) = A$
and $f \cdot h \in Ar$
by (*simp-all add*: *hom-def*)
thus $compose\ (Hom\ A\ (Dom\ f))\ (\lambda y{\in}Hom\ A\ (Dom\ g).\ g \cdot y)\ (\lambda x{\in}Hom\ A\ (Dom\ f).\ f \cdot x)\ h =$
$(g \cdot f) \cdot h$
using *f g fg 10* **by** (*simp add*: *compose-def 10 11 hom-def*)
qed
finally have *6*: *set-func* $((Hom(A,\text{-})_{\mathrm{a}}\ g) \odot (Hom(A,\text{-})_{\mathrm{a}}\ f))$

```
    = (λh∈Hom A (Dom f). (g · f) · h) .
  from 4 and 5 and 6
  have rhs: (Hom(A,-)a g) ⊙ (Hom(A,-)a f) = (|
    set-dom=Hom A (Dom f),
    set-func=(λh∈Hom A (Dom f). (g · f) · h),
    set-cod=Hom A (Cod g)|)
    by simp
  show ?thesis
    by (simp add: lhs rhs)
qed


theorem (in into-set) homf-into-set:
  Functor Hom(A,-) : AA ⟶ Set
proof (intro functor.intro functor-axioms.intro)
  show Hom(A,-)a : Ar → ar Set
    by (rule homf-preserves-arrows)
  show Hom(A,-)o : Ob → ob Set
    by (rule homf-preserves-objects)
  show ∀f∈Ar. Hom(A,-)o (Dom f) = dom Set (Hom(A,-)a f)
    by (intro ballI) (rule homf-preserves-dom)
  show ∀f∈Ar. Hom(A,-)o (Cod f) = cod Set (Hom(A,-)a f)
    by (intro ballI) (rule homf-preserves-cod)
  show ∀B∈Ob. Hom(A,-)a (Id B) = id Set (Hom(A,-)o B)
    by (intro ballI) (rule homf-preserves-id)
  show ∀f∈Ar. ∀g∈Ar.
    Cod f = Dom g ⟶
    Hom(A,-)a (g · f) = comp Set (Hom(A,-)a g) (Hom(A,-)a f)
    by (intro ballI impI, simp add: Set-def set-cat-def) (rule homf-preserves-comp)
  show two-cats AA Set
  proof intro-locales
    show category Set
      by (unfold Set-def, rule set-cat-cat)
  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 ⟶ BB
  and Functor G : AA ⟶ BB
```

and u : *ob AA* $\rightarrow$ *ar BB*
and $u \in$ *extensional* (*ob AA*)
and $\forall A{\in}Ob.\ u\ A \in Hom_{BB}\ (F_o\ A)\ (G_o\ A)$
and $\forall A{\in}Ob.\ \forall B{\in}Ob.\ \forall f{\in}Hom\ A\ B.\ (G_a\ f) \bullet_{BB} (u\ A) = (u\ B) \bullet_{BB} (F_a\ f)$

abbreviation
nt-syn (‹- : - ⇒ - *in Func* ′(- , - ′)› [*81*]) **where**
u : $F \Rightarrow G$ *in Func*(*AA*, *BB*) ≡ *natural-transformation AA BB F G u*

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

theorem (**in** *endoNT*) *id-restrict-natural*:
($\lambda A{\in}Ob.\ Id\ A$) : (*id-func AA*) ⇒ (*id-func AA*) *in Func*(*AA*,*AA*)
proof (*intro natural-transformation.intro natural-transformation-axioms.intro two-cats.intro ballI*)
show ($\lambda A{\in}Ob.\ Id\ A$) : *Ob* → *Ar*
by (*rule funcsetI*) *auto*
show ($\lambda A{\in}Ob.\ Id\ A$) ∈ *extensional* (*Ob*)
by (*rule restrict-extensional*)
fix *A*
assume *A*: *A* ∈ *Ob*
hence *Id A* ∈ *Hom A A* **..**
thus ($\lambda X{\in}Ob.\ Id\ X$) $A \in Hom\ ((id\text{-}func\ AA)_o\ A)\ ((id\text{-}func\ AA)_o\ A)$
using *A* **by** (*simp add*: *id-func-def*)
fix *B* **and** *f*
assume *B*: *B* ∈ *Ob*
and *f* ∈ *Hom A B*
hence *f* ∈ *Ar* **and** *A* = *Dom f* **and** *B* = *Cod f* **and** *Dom f* ∈ *Ob* **and** *Cod f* ∈ *Ob*
using *A* **by** (*simp-all add*: *hom-def*)
thus $(id\text{-}func\ AA)_a\ f \cdot (\lambda A{\in}Ob.\ Id\ A)\ A$
$= (\lambda A{\in}Ob.\ Id\ A)\ B \cdot (id\text{-}func\ AA)_a\ f$
by (*simp add*: *id-func-def*)
qed (*auto intro*: *id-func-functor*, *unfold-locales*, *unfold-locales*)

end

6 Yonedas Lemma

theory *Yoneda*
imports *HomFunctors NatTrans*
begin

6.1 The Sandwich Natural Transformation

locale *Yoneda* = *functor* + *into-set* +
assumes *TERM* (*AA* :: (′*o*,′*a*,′*m*)*category-scheme*)
fixes *sandwich* :: [′*o*,′*a*,′*o*] ⇒ ′*a set-arrow* (‹σ′(-,-′)›)

defines *sandwich A a* ≡ (λ*B*∈*Ob*. (|
set-dom=*Hom A B*,
set-func=(λ*f*∈*Hom A B*. *set-func* (F_a *f*) *a*),
set-cod=F_o *B*
|))
fixes *unsandwich* :: ['*o*,'*o* ⇒ '*a set-arrow*] ⇒ '*a* (‹$\sigma^{\leftarrow}$'(-,-')›)
defines *unsandwich A u* ≡ *set-func* (*u A*) (*Id A*)

lemma (**in** *Yoneda*) *F-into-set*:
Functor F : *AA* ⟶ *Set*
proof−
from *F-axioms* **have** *Functor F* : *AA* ⟶ *BB* **by** *intro-locales*
thus *?thesis*
by (*simp only*: *BB-Set*)
qed

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

lemma (**in** *Yoneda*) *sandwich-funcset*:
assumes *A*: *A* ∈ *Ob*
and *a* ∈ F_o *A*
shows σ(*A*,*a*) : *Ob* → *ar Set*
proof (*rule funcsetI*)
fix *B*
assume *B*: *B* ∈ *Ob*

thus $\sigma(A,a)\ B \in ar\ Set$
proof (*simp add*: *Set-def sandwich-def set-cat-def*)
show *set-arrow* U ⦇
set-dom $=$ *Hom* $A\ B$,
set-func $= \lambda f{\in}Hom\ A\ B.$ *set-func* $(F_{\mathrm{a}}\ f)\ a$,
set-cod $= F_{\mathrm{o}}\ B$⦈
proof (*simp add*: *set-arrow-def*, *intro conjI*)
show *Hom* $A\ B \subseteq U$ **and** $F_{\mathrm{o}}\ B \subseteq U$
by (*simp-all add*: *U-def*)
show $(\lambda f{\in}Hom\ A\ B.$ *set-func* $(F_{\mathrm{a}}\ f)\ a) \in Hom\ A\ B \rightarrow F_{\mathrm{o}}\ B$
proof (*rule funcsetI*, *simp*)
fix f
assume f: $f \in Hom\ A\ B$
with $A\ B$ **have** $F_{\mathrm{a}}\ f \in Hom_{BB}\ (F_{\mathrm{o}}\ A)\ (F_{\mathrm{o}}\ B)$
by (*rule functors-preserve-homsets*)
hence $F_{\mathrm{a}}\ f \in ar\ Set$
and *set-dom* $(F_{\mathrm{a}}\ f) = (F_{\mathrm{o}}\ A)$
and *set-cod* $(F_{\mathrm{a}}\ f) = (F_{\mathrm{o}}\ B)$
by (*simp-all add*: *hom-def BB-Set Set-def*)
hence *set-func* $(F_{\mathrm{a}}\ f) : (F_{\mathrm{o}}\ A) \rightarrow (F_{\mathrm{o}}\ B)$
by (*simp add*: *Set-def set-cat-def set-arrow-def*)
thus *set-func* $(F_{\mathrm{a}}\ f)\ a \in F_{\mathrm{o}}\ B$
using ‹$a \in F_{\mathrm{o}}\ A$›
by (*rule funcset-mem*)
qed
qed
qed
qed

lemma (**in** *Yoneda*) *sandwich-type*:
assumes A: $A \in Ob$ **and** B: $B \in Ob$
and $a \in F_{\mathrm{o}}\ A$
shows $\sigma(A,a)\ B \in hom\ Set\ (Hom\ A\ B)\ (F_{\mathrm{o}}\ B)$
proof−
have $\sigma(A,a) \in Ob \rightarrow Ar_{Set}$
using A **and** ‹$a \in F_{\mathrm{o}}\ A$› **by** (*rule sandwich-funcset*)
hence $\sigma(A,a)\ B \in ar\ Set$
using B **by** (*rule funcset-mem*)
thus *?thesis*
using B **by** (*simp add*: *sandwich-def hom-def Set-def*)
qed

lemma (**in** *Yoneda*) *sandwich-commutes*:
assumes *AOb*: $A \in Ob$ **and** *BOb*: $B \in Ob$ **and** *COb*: $C \in Ob$
and *aFa*: $a \in F_{\mathrm{o}}\ A$
and *fBC*: $f \in Hom\ B\ C$
shows $(F_{\mathrm{a}}\ f) \odot (\sigma(A,a)\ B) = (\sigma(A,a)\ C) \odot (Hom(A,\text{-})_{\mathrm{a}}\ f)$

proof −
  from *fBC* **have** *1*: $f \in Ar$ **and** *2*: $Dom\ f = B$ **and** *3*: $Cod\ f = C$
    by (*simp-all add*: *hom-def*)
  from *BOb* **have** *set-dom* $((F_a\ f) \odot (\sigma(A,a)\ B)) = Hom\ A\ B$
    by (*simp add*: *set-comp-def sandwich-def*)
  also have $\ldots =$ *set-dom* $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f))$
    by (*simp add*: *set-comp-def homf-def 1 2*)
  finally have *set-dom-eq*:
    set-dom $((F_a\ f) \odot (\sigma(A,a)\ B))$
    $=$ *set-dom* $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f))$ **.**
  from *BOb COb fBC* **have** $(F_a\ f) \in Hom_{BB}\ (F_o\ B)\ (F_o\ C)$
    by (*rule functors-preserve-homsets*)
  hence *set-cod* $((F_a\ f) \odot (\sigma(A,a)\ B)) = F_o\ C$
    by (*simp add*: *set-comp-def BB-Set Set-def set-cat-def hom-def*)
  also from *COb*
  have $\ldots =$ *set-cod* $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f))$
    by (*simp add*: *set-comp-def sandwich-def*)
  finally have *set-cod-eq*:
    set-cod $((F_a\ f) \odot (\sigma(A,a)\ B))$
    $=$ *set-cod* $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f))$ **.**
  from *AOb* **and** *BOb* **and** *COb* **and** *fBC* **and** *aFa*
  have *set-func-lhs*:
    set-func $((F_a\ f) \odot (\sigma(A,a)\ B)) =$
    $(\lambda g{\in}Hom\ A\ B.$ *set-func* $(F_a\ (f \cdot g))\ a)$
    apply (*simp add*: *set-comp-def sandwich-def compose-def*)
    apply (*rule extensionalityI*, *rule restrict-extensional*, *rule restrict-extensional*)
    by (*simp add*: *F-comp-func compose-def*)
  have $(\cdot) : Hom\ B\ C \to Hom\ A\ B \to Hom\ A\ C$ **..**
  from *this* **and** *fBC*
  have *opfType*: $(\cdot)\ f : Hom\ A\ B \to Hom\ A\ C$
    by (*rule funcset-mem*)
  from *1* **and** *2*
  have *set-func* $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f)) =$
    $(\lambda g{\in}Hom\ A\ B.$ *set-func* $(\sigma(A,a)\ C)\ (f \cdot g))$
    apply (*simp add*: *set-comp-def homf-def*)
    apply (*simp add*: *compose-def*)
    apply (*rule extensionalityI*, *rule restrict-extensional*, *rule restrict-extensional*)
    by *auto*
  also from *COb* **and** *opfType*
  have $\ldots = (\lambda g{\in}Hom\ A\ B.$ *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 *set-func-rhs*:
    set-func $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f)) =$
    $(\lambda g{\in}Hom\ A\ B.$ *set-func* $(F_a\ (f \cdot g))\ a)$ **.**
  from *set-func-lhs* **and** *set-func-rhs* **have**
    set-func $((F_a\ f) \odot (\sigma(A,a)\ B))$
    $=$ *set-func* $((\sigma(A,a)\ C) \odot (Hom(A,\text{-})_a\ f))$

by *simp*
with *set-dom-eq* **and** *set-cod-eq* **show** *?thesis*
by *simp*
qed

lemma (**in** *Yoneda*) *sandwich-natural*:
assumes $A \in Ob$
and $a \in F_o\ A$
shows $\sigma(A,a) : Hom(A,\text{-}) \Rightarrow F\ in\ Func(AA,Set)$
proof (*intro natural-transformation.intro natural-transformation-axioms.intro two-cats.intro*)
show *category AA* **..**
show *category Set*
by (*simp only*: *Set-def*)(*rule set-cat-cat*)
show $Functor\ Hom(A,\text{-}) : AA \longrightarrow Set$
by (*rule homf-into-set*)
show $Functor\ F : AA \longrightarrow Set$
by (*rule F-into-set*)
show $\forall B{\in}Ob.\ \sigma(A,a)\ B \in hom\ Set\ (Hom(A,\text{-})_o\ B)\ (F_o\ B)$
using *assms* **by** (*auto simp add*: *homf-def intro*: *sandwich-type*)
show $\sigma(A,a) : Ob \to ar\ Set$
using *assms* **by** (*rule sandwich-funcset*)
show $\sigma(A,a) \in extensional\ (Ob)$
unfolding *sandwich-def* **by** (*rule restrict-extensional*)
show $\forall B{\in}Ob.\ \forall C{\in}Ob.\ \forall f{\in}Hom\ B\ C.$
$comp\ Set\ (F\ _a\ f)\ (\sigma(A,a)\ B) = comp\ Set\ (\sigma(A,a)\ C)\ (Hom(A,\text{-})_a\ f)$
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*: $A \in Ob$
and *2*: $a \in F_o\ A$
shows $\sigma^{\leftarrow}(A,\sigma(A,a)) = a$
proof−
from *1* **have** $Id\ A \in Hom\ A\ A$ **..**
with *1*
have *3*: $\sigma^{\leftarrow}(A,\sigma(A,a)) = set\text{-}func\ (F_a\ (Id\ A))\ a$
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 $F_o\ A \in Ob_{BB}$
by (*rule funcset-mem*)
hence $F_o\ A \subseteq U$
by (*simp add*: *BB-Set Set-def set-cat-def*)
with *2*

have *5*: *set-func* (*id Set* (F_o *A*)) *a* = *a*
  by (*simp add*: *Set-def set-id-def*)
show *?thesis*
  by (*simp add*: *3 4 5*)
qed

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

by *simp*
have $\forall f \in Hom\ A\ B.\ set\text{-}dom\ (Hom(A,\text{-})_{\mathrm{a}}\ f) = Hom\ A\ A$
by (*intro ballI*, *simp add*: *homf-def hom-def*)
have *roolz*: $\bigwedge f.\ f \in Hom\ A\ B \Longrightarrow set\text{-}dom\ (Hom(A,\text{-})_{\mathrm{a}}\ f) = Hom\ A\ A$
by (*simp add*: *homf-def hom-def*)
from *1* **have** *rooly*: $Id\ A \in Hom\ A\ A$ **..**
have *roolx*: $\bigwedge f.\ f \in Hom\ A\ B \Longrightarrow f \in Ar$
by (*simp add*: *hom-def*)
have *roolw*: $\bigwedge f.\ f \in Hom\ A\ B \Longrightarrow Id\ A \in Hom\ A\ (Dom\ f)$
proof−
fix *f*
assume $f \in Hom\ A\ B$
hence $Dom\ f = A$ **by** (*simp add*: *hom-def*)
thus $Id\ A \in Hom\ A\ (Dom\ f)$
by (*simp add*: *rooly*)
qed
have *annoying*: $\bigwedge f.\ f \in Hom\ A\ B \Longrightarrow Id\ A = Id\ (Dom\ f)$
by (*simp add*: *hom-def*)
have $(\lambda f{\in}Hom\ A\ B.\ (set\text{-}func\ ((u\ B) \odot (Hom(A,\text{-}))_{\mathrm{a}}\ f))\ (Id\ A))$
$= (\lambda f{\in}Hom\ A\ B.\ (compose\ (Hom\ A\ A)\ (set\text{-}func\ (u\ B))\ (set\text{-}func\ (Hom(A,\text{-})_{\mathrm{a}}\ f)))\ (Id\ A))$
apply (*rule extensionalityI*)
apply (*rule restrict-extensional*, *rule restrict-extensional*)
by (*simp add*: *compose-def set-comp-def roolz rooly*)
also have $\ldots = (\lambda f{\in}Hom\ A\ B.\ (set\text{-}func\ (u\ B)\ f))$
apply (*rule extensionalityI*)
apply (*rule restrict-extensional*, *rule restrict-extensional*)
apply (*simp add*: *compose-def homf-def rooly roolx roolw*)
apply (*simp only*: *annoying*)
apply (*simp add*: *roolx id-right*)
done
finally have *four*:
$(\lambda f{\in}Hom\ A\ B.\ (set\text{-}func\ ((u\ B) \odot (Hom(A,\text{-}))_{\mathrm{a}}\ f))\ (Id\ A))$
$= (\lambda f{\in}Hom\ A\ B.\ (set\text{-}func\ (u\ B)\ f))$ **.**
from *2* **and** *3*
have *uBhom*: $u\ B \in hom\ Set\ (Hom(A,\text{-})_{\mathrm{o}}\ B)\ (F_{\mathrm{o}}\ B)$
by (*simp add*: *natural-transformation-def natural-transformation-axioms-def*)
with *3*
have *five*: $set\text{-}dom\ (u\ B) = Hom\ A\ B$
by (*simp add*: *hom-def homf-def Set-def set-cat-def*)
from *uBhom*
have *six*: $set\text{-}cod\ (u\ B) = F_{\mathrm{o}}\ B$
by (*simp add*: *hom-def homf-def Set-def set-cat-def*)
have *seven*: $restrict\ (set\text{-}func\ (u\ B))\ (Hom\ A\ B) = set\text{-}func\ (u\ B)$
apply (*rule extensionalityI*)
apply (*rule restrict-extensional*)
proof−
from *uBhom* **have** $u\ B \in ar\ Set$
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 $\sigma(A,\sigma^{\leftarrow}(A,u))\ B = u\ B$
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_{o} *A* ≅ {*u*. *u* : *Hom*(*A*,-) ⇒ *F in Func*(*AA*,*Set*)}
unfolding *equinumerous-def bij-betw-eq inj-on-def*
proof (*intro exI conjI bexI ballI impI*)
— Sandwich is injective
fix *x* **and** *y*
assume *2*: *x* ∈ F_{o} *A* **and** *3*: *y* ∈ F_{o} *A*
and *4*: $\sigma(A,x) = \sigma(A,y)$
hence $\sigma^{\leftarrow}(A,\sigma(A,x)) = \sigma^{\leftarrow}(A,\sigma(A,y))$
by *simp*
with *unsandwich-left-inverse*
show *x* = *y*
by (*simp add*: *1 2 3*)

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

end

References

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