

Category Theory to Yoneda's Lemma

Greg O'Keefe

June 16, 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	2
1.1	Definitions	2
1.2	Lemmas	2
2	Set is a Category	3
2.1	Definitions	3
2.2	Simple Rules and Lemmas	4
2.3	Set is a Category	5
3	Functors	6
3.1	Definitions	6
3.2	Simple Lemmas	7
3.3	Identity Functor	7
4	HomFunctors	8
5	Natural Transformations	10
6	Yoneda's Lemma	10
6.1	The Sandwich Natural Transformation	10
6.2	Sandwich Components are Bijective	11

1 Categories

```
theory Cat
imports HOL-Library.FuncSet
begin
```

1.1 Definitions

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

definition

```
hom :: [('o, 'a, 'm) category-scheme, 'o, 'o]  $\Rightarrow$  'a set
  (Hom1 - - [81,81] 80) where
  hom CC A B = { f. f  $\in$  ar CC & dom CC f = A & cod CC f = B }
```

locale category =

```
fixes CC (structure)
assumes dom-object [intro]:
  f  $\in$  Ar  $\implies$  Dom f  $\in$  Ob
and cod-object [intro]:
  f  $\in$  Ar  $\implies$  Cod f  $\in$  Ob
and id-left [simp]:
  f  $\in$  Ar  $\implies$  Id (Cod f)  $\cdot$  f = f
and id-right [simp]:
  f  $\in$  Ar  $\implies$  f  $\cdot$  Id (Dom f) = f
and id-hom [intro]:
  A  $\in$  Ob  $\implies$  Id A  $\in$  Hom A A
and comp-types [intro]:
   $\bigwedge$  A B C. (comp CC) : (Hom B C)  $\rightarrow$  (Hom A B)  $\rightarrow$  (Hom A C)
and comp-associative [simp]:
  f  $\in$  Ar  $\implies$  g  $\in$  Ar  $\implies$  h  $\in$  Ar
 $\implies$  Cod h = Dom g  $\implies$  Cod g = Dom f
 $\implies$  f  $\cdot$  (g  $\cdot$  h) = (f  $\cdot$  g)  $\cdot$  h
```

1.2 Lemmas

```
lemma (in category) homI:
  assumes f  $\in$  Ar and Dom f = A and Cod f = B
  shows f  $\in$  Hom A B
  <proof>
```

```
lemma (in category) homE:
  assumes A  $\in$  Ob and B  $\in$  Ob and f  $\in$  Hom A B
  shows Dom f = A and Cod f = B
```

<proof>

lemma (in *category*) *id-arrow* [*intro*]:

assumes $A \in Ob$

shows $Id\ A \in Ar$

<proof>

lemma (in *category*) *id-dom-cod*:

assumes $A \in Ob$

shows $Dom\ (Id\ A) = A$ and $Cod\ (Id\ A) = A$

<proof>

lemma (in *category*) *compI* [*intro*]:

assumes $f: f \in Ar$ and $g: g \in Ar$ and $Cod\ f = Dom\ g$

shows $g \cdot f \in Ar$

and $Dom\ (g \cdot f) = Dom\ f$

and $Cod\ (g \cdot f) = Cod\ g$

<proof>

end

2 Set is a Category

theory *SetCat*

imports *Cat*

begin

2.1 Definitions

record *'c set-arrow* =

set-dom :: *'c set*

set-func :: *'c* \Rightarrow *'c*

set-cod :: *'c set*

definition

set-arrow :: [*'c set*, *'c set-arrow*] \Rightarrow *bool* **where**

set-arrow $U\ f \longleftrightarrow set-dom\ f \subseteq U$ & $set-cod\ f \subseteq U$

& (*set-func* f): ($set-dom\ f$) \rightarrow ($set-cod\ f$)

& *set-func* $f \in extensional\ (set-dom\ f)$

definition

set-id :: [*'c set*, *'c set*] \Rightarrow *'c set-arrow* **where**

set-id $U = (\lambda s \in Pow\ U. (\setminus set-dom=s, set-func=\lambda x \in s. x, set-cod=s))$

definition

set-comp :: [*'c set-arrow*, *'c set-arrow*] \Rightarrow *'c set-arrow* (**infix** \odot 70) **where**

set-comp $g\ f =$

(

$set-dom = set-dom f,$
 $set-func = compose (set-dom f) (set-func g) (set-func f),$
 $set-cod = set-cod g$

)

definition

$set-cat :: 'c set \Rightarrow ('c set, 'c set-arrow)$ category **where**

$set-cat U =$

(

$ob = Pow U,$
 $ar = \{f. set-arrow U f\},$
 $dom = set-dom,$
 $cod = set-cod,$
 $id = set-id U,$
 $comp = set-comp$

)

2.2 Simple Rules and Lemmas

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

<proof>

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

<proof>

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

assumes $A \subseteq U$

and $B \subseteq U$

and $f : A \rightarrow B$

and $f \in extensional A$

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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

<proof>

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 $\text{cod } (\text{set-cat } U) f \in \text{ob } (\text{set-cat } U)$
and $\text{set-cod } f \subseteq U$
and $\text{set-dom } f \subseteq U$
 <proof>

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>

2.3 Set is a Category

lemma *set-id-left*:
assumes $f: f \in \text{ar } (\text{set-cat } U)$
shows $\text{set-id } U (\text{set-cod } f) \odot f = f$
 <proof>

lemma *set-id-right*:
assumes $f: f \in \text{ar } (\text{set-cat } U)$
shows $f \odot (\text{set-id } U (\text{set-dom } f)) = f$
 <proof>

lemma *set-id-hom*:
assumes $A \in \text{ob } (\text{set-cat } U)$
shows $\text{id } (\text{set-cat } U) A \in \text{hom } (\text{set-cat } U) A A$
 <proof>

lemma *set-comp-types*:
 $\text{comp } (\text{set-cat } U) \in \text{hom } (\text{set-cat } U) B C \rightarrow \text{hom } (\text{set-cat } U) A B \rightarrow \text{hom } (\text{set-cat } U) A C$
 <proof>

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 $f: f \in \text{ar } (\text{set-cat } U)$
and $g: g \in \text{ar } (\text{set-cat } U)$
and $h: h \in \text{ar } (\text{set-cat } U)$
and $hg: \text{cod } (\text{set-cat } U) h = \text{dom } (\text{set-cat } U) g$
and $gf: \text{cod } (\text{set-cat } U) g = \text{dom } (\text{set-cat } U) f$
shows $\text{comp } (\text{set-cat } U) f (\text{comp } (\text{set-cat } U) g h) = \text{comp } (\text{set-cat } U) (\text{comp } (\text{set-cat } U) f g) h$

<proof>

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

<proof>

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 (*- o* [81]) **where**

$F_o \equiv om\ F$

abbreviation

am-syn (*- a* [81]) **where**

$F_a \equiv am\ F$

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* \Rightarrow *bool*

and *preserves-cod* :: (*'o1, 'a1, 'o2, 'a2*)*functor* \Rightarrow *bool*

and *preserves-id* :: (*'o1, 'a1, 'o2, 'a2*)*functor* \Rightarrow *bool*

and *preserves-comp* :: (*'o1, 'a1, 'o2, 'a2*)*functor* \Rightarrow *bool*

defines *preserves-dom* $G \equiv \forall f \in Ar_{AA}. G_o (Dom_{AA}\ f) = Dom_{BB}\ (G_a\ f)$

and *preserves-cod* $G \equiv \forall f \in Ar_{AA}. G_o (Cod_{AA}\ f) = Cod_{BB}\ (G_a\ f)$

and *preserves-id* $G \equiv \forall A \in Ob_{AA}. G_a (Id_{AA}\ A) = Id_{BB}\ (G_o\ A)$

and *preserves-comp* $G \equiv$

$\forall f \in Ar_{AA}. \forall g \in Ar_{AA}. Cod_{AA}\ f = Dom_{AA}\ g \longrightarrow G_a (g \cdot_{AA}\ f) = (G_a\ g) \cdot_{BB}\ (G_a\ f)$

locale *functor* = *two-cats* +

fixes *F (structure)*

assumes *F-preserves-arrows*: $F_a : Ar_{AA} \rightarrow Ar_{BB}$

and *F-preserves-objects*: $F_o : Ob_{AA} \rightarrow Ob_{BB}$

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* *<proof>*

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

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

and *f* \in *ar AA*

shows *F*_a *f* \in *ar BB*

<proof>

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*_a *f* \in *Hom*_{BB} (*F*_o *A*) (*F*_o *B*)

<proof>

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

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

and *A* \in *ob* *AA*

shows *F*_o *A* \in *ob* *BB*

<proof>

3.3 Identity Functor

definition

id-func :: (*'o','a','m*) *category-scheme* \Rightarrow (*'o','a','o','a*) *functor* **where**

id-func *CC* = (*om*=($\lambda A \in ob$ *CC*. *A*), *am*=($\lambda f \in ar$ *CC*. *f*))

locale *one-cat* = *two-cats* +

```

assumes endo:  $BB = AA$ 

lemma (in one-cat) id-func-preserves-arrows:
  shows  $(id-func\ AA)_a : Ar \rightarrow Ar$ 
   $\langle proof \rangle$ 

lemma (in one-cat) id-func-preserves-objects:
  shows  $(id-func\ AA)_o : Ob \rightarrow Ob$ 
   $\langle proof \rangle$ 

lemma (in one-cat) id-func-preserves-dom:
  shows preserves-dom (id-func AA)
   $\langle proof \rangle$ 

lemma (in one-cat) id-func-preserves-cod:
  preserves-cod (id-func AA)
   $\langle proof \rangle$ 

lemma (in one-cat) id-func-preserves-id:
  preserves-id (id-func AA)
   $\langle proof \rangle$ 

lemma (in one-cat) id-func-preserves-comp:
  preserves-comp (id-func AA)
   $\langle proof \rangle$ 

theorem (in one-cat) id-func-functor:
  Functor (id-func AA) :  $AA \longrightarrow AA$ 
   $\langle proof \rangle$ 

end

```

4 HomFunctors

```

theory HomFunctors
imports SetCat Functors
begin

locale into-set = two-cats AA BB
  for AA ::  $(\prime o, \prime a, \prime m)category-scheme$  (structure)
  and BB (structure) +
  fixes U and Set
  defines  $U \equiv (UNIV :: \prime a\ set)$ 
  defines  $Set \equiv set-cat\ U$ 
  assumes BB-Set:  $BB = Set$ 

```


fixes $homf$ ($Hom'(-, '-')$)
defines $homf$ $A \equiv \langle$
 $om = (\lambda B \in Ob. Hom\ A\ B),$
 $am = (\lambda f \in Ar. \langle set-dom = Hom\ A\ (Dom\ f), set-func = (\lambda g \in Hom\ A\ (Dom\ f). f \cdot$
 $g), set-cod = Hom\ A\ (Cod\ f) \rangle)$
 \rangle

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

$Hom(A, -)_a : Ar \rightarrow ar\ Set$
 $\langle proof \rangle$

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

$Hom(A, -)_o : Ob \rightarrow ob\ Set$
 $\langle proof \rangle$

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

assumes $f: f \in Ar$
shows $Hom(A, -)_o (Dom\ f) = dom\ Set\ (Hom(A, -)_a\ f)$
 $\langle proof \rangle$

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

assumes $f: f \in Ar$
shows $Hom(A, -)_o (Cod\ f) = cod\ Set\ (Hom(A, -)_a\ f)$
 $\langle proof \rangle$

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

assumes $B: B \in Ob$
shows $Hom(A, -)_a (Id\ B) = id\ Set\ (Hom(A, -)_o\ B)$
 $\langle proof \rangle$

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

assumes $f: f \in Ar$
and $g: g \in Ar$
and $fg: Cod\ f = Dom\ g$
shows $Hom(A, -)_a (g \cdot f) = (Hom(A, -)_a\ g) \odot (Hom(A, -)_a\ f)$
 $\langle proof \rangle$

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

$Functor\ Hom(A, -) : AA \rightarrow Set$
 $\langle proof \rangle$

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 → ar BB
  and u ∈ extensional (ob AA)
  and ∀ A ∈ Ob. u A ∈ HomBB (Fo A) (Go A)
  and ∀ A ∈ Ob. ∀ B ∈ Ob. ∀ f ∈ Hom A B. (Ga f) ·BB (u A) = (u B) ·BB (Fa f)
```

abbreviation

```
nt-syn (- : - ⇒ - in Func '(-, -) [81]) where
u : F ⇒ G in Func(AA, BB) ≡ natural-transformation AA BB F G u
```

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

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

```
(λA ∈ Ob. Id A) : (id-func AA) ⇒ (id-func AA) in Func(AA, AA)
⟨proof⟩
```

end

6 Yoneda 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 ≡ (λB ∈ Ob. (
    set-dom = Hom A B,
    set-func = (λf ∈ Hom A B. set-func (Fa f) a),
    set-cod = Fo B
  ))
  fixes unsandwich :: ['o, 'o] ⇒ 'a set-arrow ⇒ 'a (σ←'(-, -))
  defines unsandwich A u ≡ set-func (u A) (Id A)
```

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

Functor $F : AA \longrightarrow Set$
<proof>

lemma (in *Yoneda*) *F-comp-func*:
assumes 1: $A \in Ob$ and 2: $B \in Ob$ and 3: $C \in Ob$
and 4: $g \in Hom\ A\ B$ and 5: $f \in Hom\ B\ C$
shows *set-func* $(F_a (f \cdot g)) = compose (F_o A) (set-func (F_a f)) (set-func (F_a g))$
<proof>

lemma (in *Yoneda*) *sandwich-funcset*:
assumes $A: A \in Ob$
and $a \in F_o A$
shows $\sigma(A,a) : Ob \rightarrow ar\ Set$
<proof>

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>

lemma (in *Yoneda*) *sandwich-commutes*:
assumes $AOb: A \in Ob$ and $BOb: B \in Ob$ and $COb: C \in Ob$
and $aFa: a \in F_o A$
and $fBC: f \in Hom\ B\ C$
shows $(F_a f) \odot (\sigma(A,a) B) = (\sigma(A,a) C) \odot (Hom(A,-)_a f)$
<proof>

lemma (in *Yoneda*) *sandwich-natural*:
assumes $A \in Ob$
and $a \in F_o A$
shows $\sigma(A,a) : Hom(A,-) \Rightarrow F\ in\ Func(AA,Set)$
<proof>

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>

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

assumes 1: $A \in Ob$
and 2: $u : Hom(A, -) \Rightarrow F$ in $Func(AA, Set)$
shows $\sigma(A, \sigma^{\leftarrow}(A, u)) = u$
 <proof>

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

$surj-on :: ['a \Rightarrow 'b, 'a\ set, 'b\ set] \Rightarrow bool$ **where**
 $surj-on\ f\ A\ B \longleftrightarrow (\forall y \in B. \exists x \in A. f(x)=y)$

definition

$bij-on :: ['a \Rightarrow 'b, 'a\ set, 'b\ set] \Rightarrow bool$ **where**
 $bij-on\ f\ A\ B \longleftrightarrow inj-on\ f\ A \ \&\ surj-on\ f\ A\ B$

definition

$equinumerous :: ['a\ set, 'b\ set] \Rightarrow bool$ (**infix** \cong 40) **where**
 $equinumerous\ A\ B \longleftrightarrow (\exists f. bij-betw\ f\ A\ B)$

lemma *bij-betw-eq*:

$bij-betw\ f\ A\ B \longleftrightarrow$
 $inj-on\ f\ A \wedge (\forall y \in B. \exists x \in A. f(x)=y) \wedge (\forall x \in A. f\ x \in B)$
 <proof>

theorem (**in** *Yoneda*) *Yoneda*:

assumes 1: $A \in Ob$
shows $F_{\circ} A \cong \{u. u : Hom(A, -) \Rightarrow F\}$ in $Func(AA, Set)$
 <proof>

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.