

Category Theory to Yoneda's Lemma

Greg O'Keefe

December 12, 2009

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

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

Contents

1	Categories	2
1.1	Definitions	2
1.2	Lemmas	2
2	Set is a Category	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 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 - -) 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>
```


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

)

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]: $A \subseteq U \Longrightarrow A \in ob (set-cat U)$

$\langle proof \rangle$

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

$\langle proof \rangle$

lemma $set-homI$ [intro]:

assumes $A \subseteq U$

and $B \subseteq U$

and $f : A \rightarrow B$

and $f \in extensional A$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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

$\langle proof \rangle$

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 (structure) and BB (structure) +*

assumes *AA = (AA :: ('o1, 'a1, 'm1)category-scheme)*

assumes *BB = (BB :: ('o2, 'a2, 'm2)category-scheme)*

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

$\forall f \in Ar_1. G_o (Dom_1 f) = Dom_2 (G_a f)$

and *preserves-cod G* \equiv

$\forall f \in Ar_1. G_o (Cod_1 f) = Cod_2 (G_a f)$

and *preserves-id G* \equiv

$\forall A \in Ob_1. G_a (Id_1 A) = Id_2 (G_o A)$

and *preserves-comp G* \equiv

$\forall f \in Ar_1. \forall g \in Ar_1. Cod_1 f = Dom_1 g \longrightarrow G_a (g \cdot_1 f) = (G_a g) \cdot_2 (G_a f)$

locale *functor* = *two-cats +*

fixes *F (structure)*

assumes *F-preserves-arrows: F_a : Ar₁ \rightarrow Ar₂*

and F -preserves-objects: $F_o : Ob_1 \rightarrow Ob_2$
and F -preserves-dom: preserves-dom F
and F -preserves-cod: preserves-cod F
and F -preserves-id: preserves-id F
and F -preserves-comp: preserves-comp F
notes F -axioms = F -preserves-arrows F -preserves-objects F -preserves-dom
 F -preserves-cod F -preserves-id F -preserves-comp
notes func-pred-defs = preserves-dom-def preserves-cod-def preserves-id-def preserves-comp-def

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

abbreviation

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

3.2 Simple Lemmas

For example:

lemma (**in** $functor$) $Functor$ $F : AA \longrightarrow BB$ $\langle proof \rangle$

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

assumes $Functor$ $F : AA \longrightarrow BB$
and $f \in ar$ AA
shows F_a $f \in ar$ BB
 $\langle proof \rangle$

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

assumes 1: $A \in Ob_1$
and 2: $B \in Ob_1$
and 3: $f \in Hom_1$ A B
shows F_a $f \in Hom_2$ (F_o A) (F_o B)
 $\langle proof \rangle$

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

assumes $Functor$ $F : AA \longrightarrow BB$
and $A \in ob$ AA
shows F_o $A \in ob$ BB
 $\langle proof \rangle$

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 $(id\text{-func } AA)_a : Ar \rightarrow Ar$
 ⟨*proof*⟩

lemma (in *one-cat*) *id-func-preserves-objects*:
 shows $(id\text{-func } AA)_o : Ob \rightarrow Ob$
 ⟨*proof*⟩

lemma (in *one-cat*) *id-func-preserves-dom*:
 shows *preserves-dom* (*id-func* *AA*)
 ⟨*proof*⟩

lemma (in *one-cat*) *id-func-preserves-cod*:
 preserves-cod (*id-func* *AA*)
 ⟨*proof*⟩

lemma (in *one-cat*) *id-func-preserves-id*:
 preserves-id (*id-func* *AA*)
 ⟨*proof*⟩

lemma (in *one-cat*) *id-func-preserves-comp*:
 preserves-comp (*id-func* *AA*)
 ⟨*proof*⟩

theorem (in *one-cat*) *id-func-functor*:
 Functor (*id-func* *AA*) : *AA* \longrightarrow *AA*
 ⟨*proof*⟩

end

4 HomFunctors

theory *HomFunctors*
imports *SetCat Functors*
begin

locale *into-set* = *two-cats* +
 assumes *AA* = (*AA*::('o,'a,'m)*category-scheme*)
 fixes *U* and *Set*
 defines *U* \equiv (*UNIV*::'a *set*)
 defines *Set* \equiv *set-cat* *U*
 assumes *BB-Set*: *BB* = *Set*
 fixes *homf* (*Hom*'(-,'-'))

defines $homf\ A \equiv \langle \langle$
 $om = (\lambda B \in Ob. Hom\ A\ B),$
 $am = (\lambda f \in Ar. \langle \langle set-dom = Hom\ A\ (Dom\ f), set-func = (\lambda g \in Hom\ A\ (Dom\ f). f \cdot$
 $g \rangle, set-cod = Hom\ A\ (Cod\ f) \rangle \rangle)$
 \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*:
 $Functor\ Hom(A, -) : AA \longrightarrow Set$
 $\langle proof \rangle$

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 ∈ Hom2 (Fo A) (Go A)
  and ∀ A ∈ Ob. ∀ B ∈ Ob. ∀ f ∈ Hom A B. (Ga f) •2 (u A) = (u B) •2 (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 AA = (AA :: ('o, 'a, 'm) category-scheme)
  fixes sandwich :: ['o, 'a, 'o] ⇒ 'a set-arrow (σ'(-, -'))
  defines sandwich A a ≡ (λB ∈ Ob. (
    set-dom = Hom A B,
    set-func = (λf ∈ Hom A B. set-func (Fa f) a),
    set-cod = Fo B
  ))
  fixes unsandwich :: ['o, 'o] ⇒ 'a set-arrow ⇒ 'a (σ←'(-, -'))
  defines unsandwich A u ≡ set-func (u A) (Id A)
```

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

Functor $F : AA \longrightarrow Set$

<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 $set\text{-}func\ (F_a\ (f \cdot g)) = compose\ (F_o\ A)\ (set\text{-}func\ (F_a\ f))\ (set\text{-}func\ (F_a\ g))$

<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 $\sigma(A, \sigma^{\leftarrow}(A, u)) = u$
 $\langle proof \rangle$

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**
 $equinumerous\ A\ B \longleftrightarrow (\exists f. bij-on\ f\ A\ B)$

theorem (in *Yoneda*) *Yoneda*:
assumes 1: $A \in Ob$
shows $F \circ A \cong \{u. u : Hom(A, -) \Rightarrow F\}$ in $Func(AA, Set)$
 $\langle proof \rangle$

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.