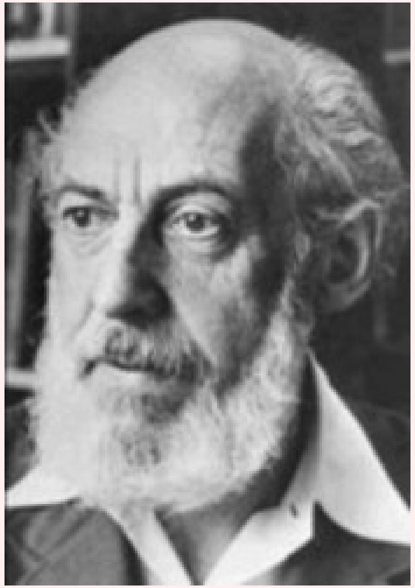
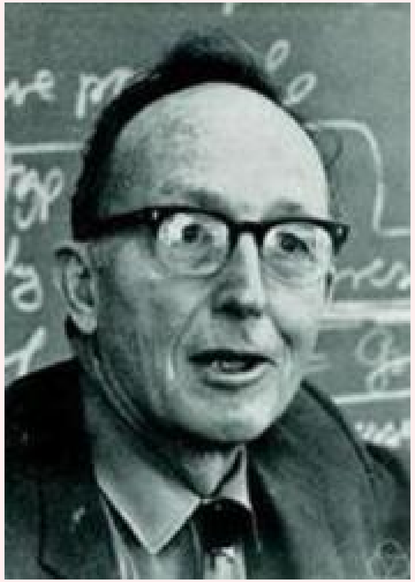
Naturality & The Yoneda Lemma

15-150 M21

Lecture 0809
09 August 2021





- The content of this lecture falls under the mathematical discipline of **category theory**
- Category theory was invented in the mid 20th century to study algebra, but has since revolutionized various fields of mathematics and computer science
- Take a course on category theory if you get the chance!

Note:

Only talking about total functions today

0 Type Isomorphisms

Bijections

Defn. Given sets X and Y, a function $f : X \to Y$ is said to be **bijective** if

- f is **injective**: for all $x, x' \in X$, if $f(x) = f(x')$ then $x = x'$
- f is **surjective**: for all $y \in Y$, there exists $x \in X$ such that $f(x) = y$

Defn. Given sets X and Y, a function $f : X \to Y$ is said to be **bijective** if there exists a function $f^{-1} : Y \to X$ such that

$$f \circ f^{-1} = \mathrm{id}_Y \qquad \text{and} \qquad f^{-1} \circ f = \mathrm{id}_X$$

Defn. An SML function `f : t1 -> t2` is called a **type isomorphism** if there exists some `g : t2 -> t1` such that

$$\texttt{f} \circ \texttt{g} \cong \texttt{id}_{\texttt{t2}} \qquad \text{and} \qquad \texttt{g} \circ \texttt{f} \cong \texttt{id}_{\texttt{t1}}$$

We'll write $\texttt{t1} \cong_{\mathsf{Ty}} \texttt{t2}$ if there exists such a type isomorphism `f : t1 -> t2`.

Example: Times 1

Claim For any type `t1`,

$$\texttt{t1} \cong_{\mathsf{Ty}} \texttt{t1 * unit}$$

0809.0 (iso.sml)

```
fun mulByOne x = (x,())

fun divByOne (x,()) = x
```

Demonstration: Sum Types

Claim For any type `t1`,

$$\texttt{t1 option} \cong_{\mathsf{Ty}} \texttt{(t1,unit) plus}$$

0809.1 (iso.sml)

```
fun encode NONE = inR()
  | encode (SOME x) = inL x

fun decode (inR()) = NONE
  | decode (inL x) = SOME x
```

Example: Distributivity

Claim For any types `t1`, `t2`, `t3`,

$$\texttt{(t1,t2) plus * t3} \cong_{\mathsf{Ty}} \texttt{(t1 * t3, t2 * t3) plus}$$

0809.2 (iso.sml)

```
fun distribute (inL x,z) = inL(x,z)
  | distribute (inR y,z) = inR(y,z)

fun factor (inL(x,z)) = (inL x,z)
  | factor (inR(y,z)) = (inR y,z)
```

1 Functors

Terminology Clash

- In SML, a **functor** is a `structure` that has been parametrized/lambda-abstracted with an argument (ascribing to some `signature`)
- In category theory (and in other parts of functional programming), it has a related but different meaning...

Key Idea:

What do `options`, `lists`, and `trees` have in common?

They all "contain" data

Part of what it means to "contain" is to *map*

```
map : ('a -> 'b) -> 'a option -> 'b option

map : ('a -> 'b) -> 'a list -> 'b list

map : ('a -> 'b) -> 'a tree -> 'b tree
```

0809.3 (functors.sml)

```
signature FUNCTOR =
sig
  type 'a t
  val fmap : ('a -> 'b) -> 'a t -> 'b t
end
```

Invariant

- For all `f : t1 -> t2`, `g : t2 -> t3`,

  $$\texttt{(fmap g) o (fmap f)} \cong \texttt{fmap(g o f)}$$

- For all types `t1`,

  $$\texttt{fmap id}_{\texttt{t1}} \cong \texttt{id}_{\texttt{t1 t}}$$

0809.4 (functors.sml)

```
structure L : FUNCTOR =
struct
  type 'a t = 'a list
  val fmap = List.map
end
```

0809.5 (functors.sml)

```
structure O : FUNCTOR = struct
  type 'a t = 'a option
  fun fmap f NONE = NONE
    | fmap f (SOME x) = SOME(f x)
end
```

0809.6 (functors.sml)

```
structure T : FUNCTOR = struct
  type 'a t = 'a tree
  fun fmap f Empty = Empty
    | fmap f (Node(L,x,R)) =
       Node(fmap f L,f x, fmap f R)
end
```

0809.7 (functors.sml)

```
functor P1(type t0) : FUNCTOR =
struct
   type 'a t = t0 * 'a

   fun fmap f (t,x) = (t,f x)
end
functor P2(type t0) : FUNCTOR =
struct
   type 'a t = 'a * t0

   fun fmap f (x,t) = (f x, t)
```

0809.8 (functors.sml)

```
structure I : FUNCTOR =
struct
  type 'a t = 'a

  fun fmap f x = f x
end
```

0809.9 (functors.sml)

```
functor Z(type t0) : FUNCTOR =
struct
  type 'a t = t0 -> 'a

  fun fmap f g = f o g
end
```

2 Naturality

0809.10 (natural.sml)

```
structure S : FUNCTOR =
struct
  type 'a t = ('a,unit) plus

  fun fmap f (inR ()) = inR ()
    | fmap f (inL x) = inL(f x)
end
```

0809.5 (functors.sml)

```
structure O : FUNCTOR = struct
  type 'a t = 'a option
  fun fmap f NONE = NONE
    | fmap f (SOME x) = SOME(f x)
```

0809.1 (iso.sml)

```
fun encode NONE = inR()
  | encode (SOME x) = inL x

fun decode (inR()) = NONE
  | decode (inL x) = SOME x
```

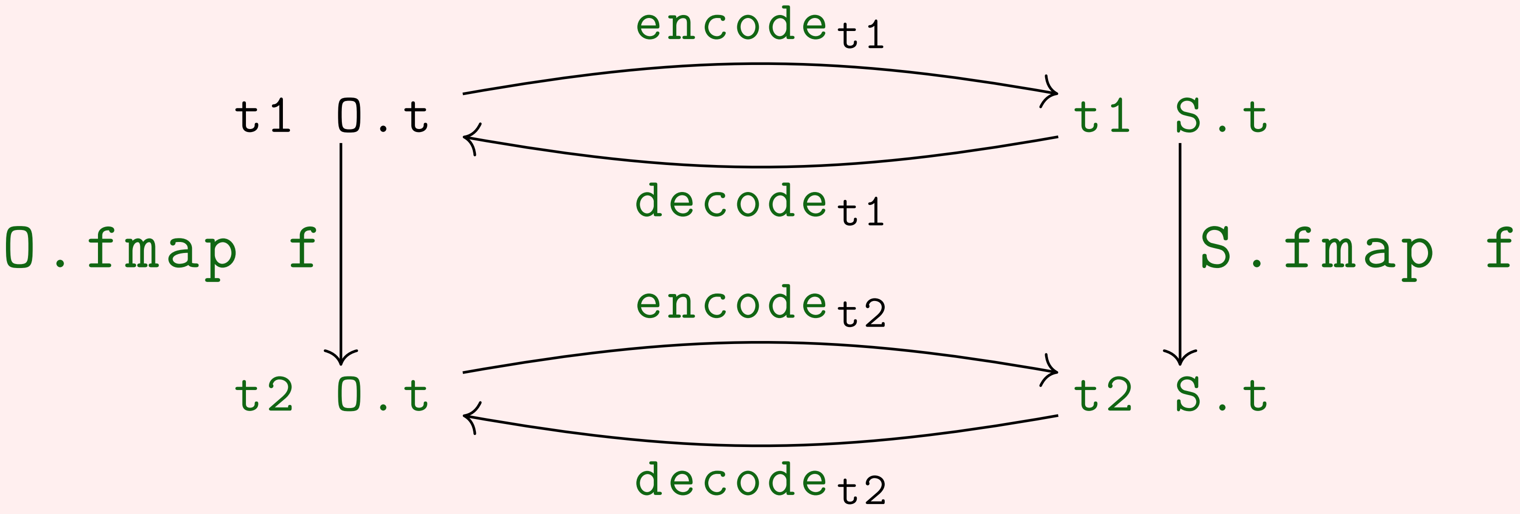
For each type `t1`, write

$$\texttt{encode}_{\texttt{t1}}\texttt{: t1 option -> (t1,unit) plus}$$
$$\texttt{decode}_{\texttt{t1}}\texttt{: (t1,unit) plus -> t1 option}$$

For all types t1, t2 and all total f : t1 -> t2,

$$\texttt{encode}_{\texttt{t2}} \circ (\texttt{O.fmap f}) \cong (\texttt{S.fmap f}) \circ \texttt{encode}_{\texttt{t1}}$$

$$(\texttt{O.fmap f}) \circ \texttt{decode}_{\texttt{t1}} \cong \texttt{decode}_{\texttt{t2}} \circ (\texttt{S.fmap f})$$



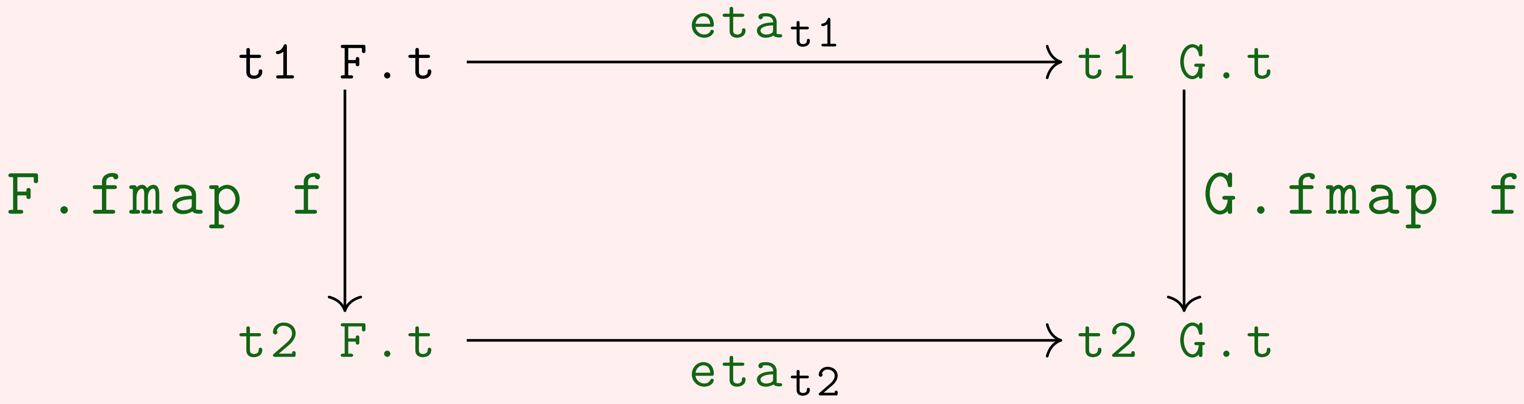
Defn. Given functors `F`, `G`, a total polymorphic function

```
eta : 'a F.t -> 'a G.t
```

is said to be a **natural transformation from `F` to `G`** if

$$\texttt{eta}_{\texttt{t2}} \circ (\texttt{F.fmap f}) \cong (\texttt{G.fmap f}) \circ \texttt{eta}_{\texttt{t1}}$$

for all `f : t1 -> t2`.



Defn. A natural transformation `eta` from `F` to `G` is said to be a **natural isomorphism** if there exists `eta'` from `G` to `F` such that, for each type `t1`, $\texttt{eta}_{\texttt{t1}}$ is a type isomorphism (with inverse $\texttt{eta'}_{\texttt{t1}}$) witnessing

$$\texttt{t1 F.t} \cong_{\mathsf{Ty}} \texttt{t1 G.t}$$

We'll write $\texttt{F} \cong_{\mathsf{N}} \texttt{G}$ if there exists such a natural iso.

Good News: Polymorphic functions are always natural

Prop. If `eta : 'a F.t -> 'a G.t` is a pure, total, polymorphic SML function, then `eta` is natural with respect to the functors `F` and `G`.

Proving this requires use of an important theoretical result known as *parametricity*.

Example: `preord`

The function `preord : 'a tree -> 'a list` constitutes a natural transformation from `T` (the tree functor) to `L` (the list functor).

Check Your Understanding Is it a natural iso?

Check Your Understanding

Verify that the function

0809.11 (natural.sml)

```
fun swap (x,y) = (y,x)
```

is a natural isomorphism between the functors `P1(t0)` and `P2(t0)` for any type `t0`.

Check Your Understanding

Define functors `F` and `G` such that the functions

```
Fn.curry : ('a * 'b -> 'c) -> ('a -> 'b -> 'c)
Fn.uncurry : ('a -> 'b -> 'c) -> ('a * 'b -> 'c)
```

constitute a natural isomorphism `F` $\cong_{\mathsf{N}}$ `G`.

3 The Yoneda Lemma

Notation: Write `Nat(F,G)` for the type of natural transformations `eta:'a F.t -> 'a G.t`. Also recall that for any type `t1`, `Z(t1)` is the functor `F` where `'a F.t = t1 -> 'a`.

Lemma For any functor `G` and any type `t1`, there is a type isomorphism

$$\texttt{Nat(Z(t1), G)} \quad \cong_{\mathsf{Ty}} \quad \texttt{t1 G.t}$$

Note: `Nat(Z(t1),G)` is the same as `(t1 -> 'a) -> 'a G.t`.

Proof (sketch)

0809.12 (natural.sml)

```
functor YonedaLemma (type t1
                     structure G : FUNCTOR) =
struct
  structure Zt1 : FUNCTOR = Z(type t0 = t1)
  (*
   fun forward(eta:'a Zt1.t -> 'a G.t):t1 G.t =
      eta(Fn.id)
  *)
  fun backward(x : t1 G.t):'a Zt1.t -> 'a G.t =
       fn k => G.fmap k x
end
```

Example: Identity Functor

Take `G=I`, the identity functor. Then the lemma says that

$$\texttt{(t1 -> 'a) -> 'a} \quad \cong_{\mathsf{Ty}} \quad \texttt{t1}$$

Take `G=O`, the option functor. Then the lemma says that

$$\texttt{(t1 -> 'a) -> 'a option} \quad \cong_{\mathsf{Ty}} \quad \texttt{t1 option}$$

Example: List Functor

Take `G=L`, the list functor. Then the lemma says that

$$\texttt{(t1 -> 'a) -> 'a list} \quad \cong_{\mathsf{Ty}} \quad \texttt{t1 list}$$

Take `G=P1(t0)`, the product-with-`t0` functor. Then the lemma says that

$$\texttt{(t1 -> 'a) -> t0 * 'a} \quad \cong_{\mathsf{Ty}} \quad \texttt{t0 * t1}$$

Take `G=Z(t0)`. Then the lemma says that

$$\texttt{(t1 -> 'a) -> t0 -> 'a} \quad \cong_{\mathsf{Ty}} \quad \texttt{t0 -> t1}$$

$$\texttt{t0 -> (t1 -> 'a) -> 'a} \quad \cong_{\mathsf{Ty}} \quad \texttt{t0 -> t1}$$

```
fact : int -> int
```
REQUIRES: `n` $\geq$ `0`
ENSURES: `fact n` $\cong$ `n!`

```
factCPS : (int -> 'a) -> int -> 'a
```
REQUIRES: `n` $\geq$ `0`
ENSURES:

`factCPS k` $\cong$ `k o fact`

Check Your Understanding

Verify that the type isomorphism

```
t0 -> t1   ≅Ty   (t1 -> 'a) -> t0 -> 'a
```

takes

```
f : t0 -> t1
```

to

```
(fn k => k o f) : (t1 -> 'a) -> t0 -> 'a
```

Thank you!