米田の反転工学
Yoneda's Reverse Engineering Machine

yukita@hosei.ac.jp

米田の補題 The Yoneda Lemma

- 圏論の基本定理
- でも今回は圏論の話を抜きにどれだけ米田の補題の核心に迫れるか頑張ってみたい
- 仕様を決めたら実装が決ってしまうという現象についての定理
- 仕様の情報から機械を reverse engineer するという話
- 具体的には Haskell の多相関数で signature を決めてしまうと実装が決まってしまうという現象(別にHaskellでなくてよいが)

米田信夫とは何者？

- 数学者でありコンピューター科学の先駆者(1930-1996)
  - 東大，プリンストン高等研究所，学習院大学，東京電機大学
- 米田の補題
  - 東大の数学教室で修士セミナーでの発表
  - 世界中の数学の論文では最も多く引用されている定理
- ALGOL Nの設計
- コンピューター科学の著書多数

補題(lemma)とは

- 補助定理という意味
- 定理を証明するための補助に使われる命題
- ところが，日々生産される数学者の論文で定理の証明に毎日のように使われている補題がいくつかある
  - 米田の補題
  - 中山の補題
  - Zornの補題
- これらははっきり言って定理より偉い

仕様が実装を決める

Milewski
および
A Neighborhood of Infinity
のブログから素材をもらっています

yoneda.hs

```
{-# LANGUAGE ExplicitForAll #-}
imager :: forall r . ((Bool -> r) -> [r])
imager iffie = fmap iffie [True, False, True, True]

data Color = Red | Green | Blue         deriving Show
data Note  = C | D | E | F | G | A | B deriving Show

colorMap x = if x then Blue else Red
heatMap  x = if x then 32   else 212
soundMap x = if x then C    else G

idBool :: Bool -> Bool
idBool x = x

{- suggested tests
imager colorMap
imager heatMap
imager soundMap
-}
```

Quiz 1: imager と同じシグネチャーをもつ imager2 を作成し同様の実験をせよ.

imager :: forall r . ((Bool -> r) -> [r])

- (Bool -> r)
  - Boolean の値を引数にとって r 型の値を返す関数の型
- [r]
  - r 型の要素からなるリストの型
- ((Bool -> r) -> [r])
  - (Bool -> r) 型の関数を引数にとって [r] 型のリストを返す関数
- forall r . _____
  - r にどんな型がやってきても，実装は特定の型 r に依存してはならない
  - 圏論の用語では自然変換

米田の補題を適用すると

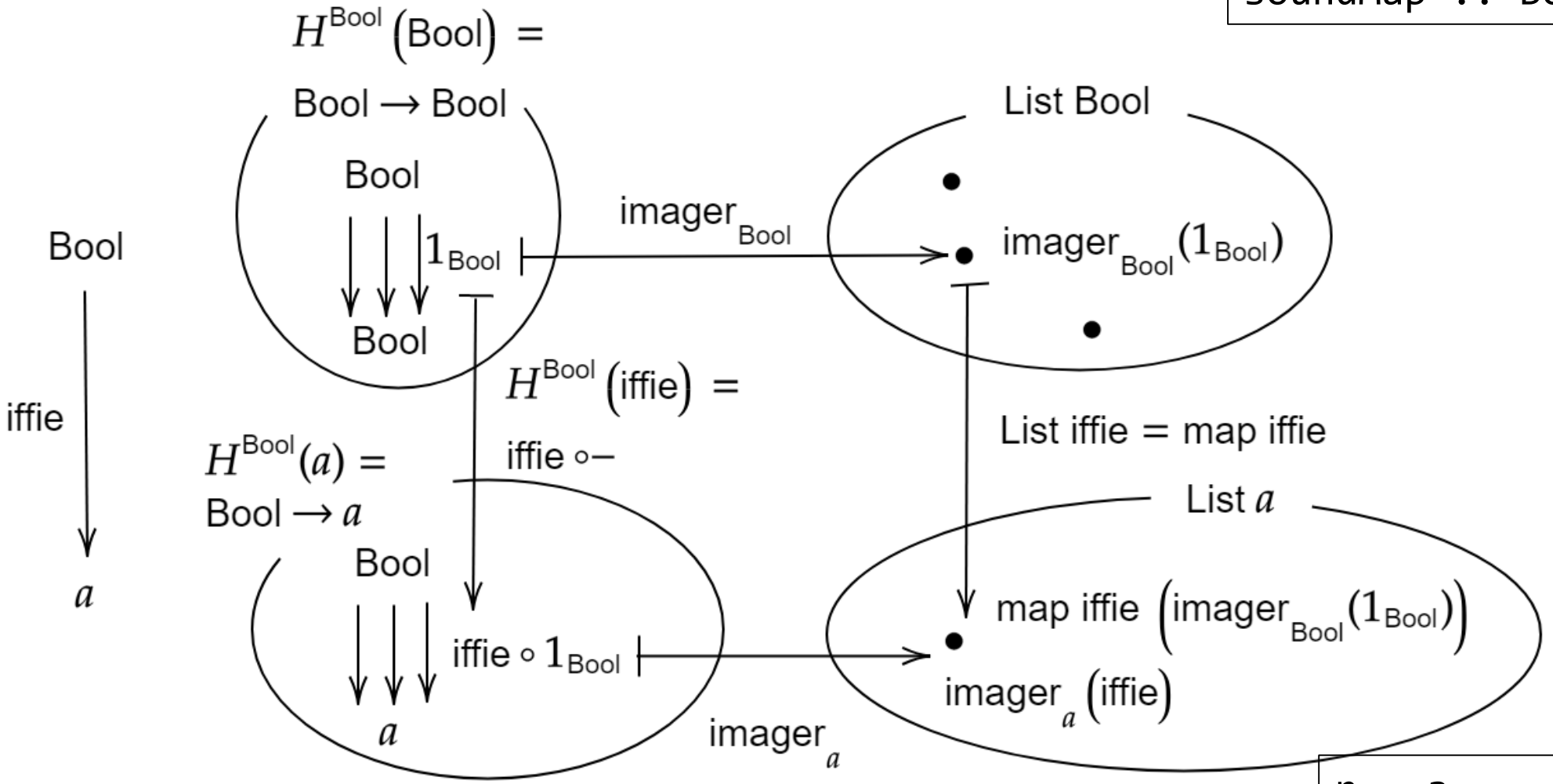
forall r . ((Bool -> r) -> [r])
のシグネチャーをもつ関数と
[Bool]
の型をもつリストは1対1に対応する

シグネチャー `forall r . ((Bool->r)->[r])`
をもつ関数は
Hom関手 `(Bool-> - )` から リスト関手 `[]` への自然変換
と呼ばれる.

imager :: forall r . ((Bool -> r) -> [r])

```
*Main> imager idBool
[True,False,True,True]
```

```
colorMap :: Bool -> Color
heatMap  :: Bool -> Int
soundMap :: Bool -> Note
```




```
r = Bool
```

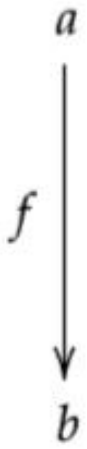
```
r = a
a = Color | Int | Note
```

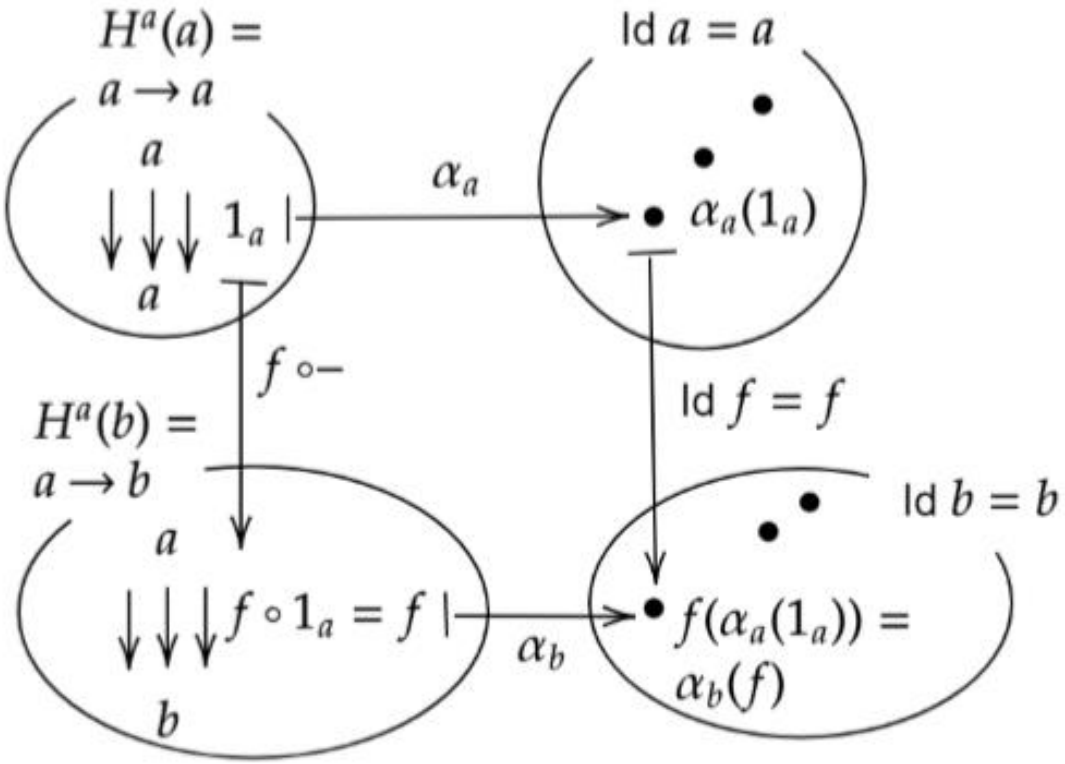
パラメータを隠し持つ機械の反転工学 unbox1.hs

```
{-# LANGUAGE RankNTypes #-}
factory1 :: a -> (forall b . (a -> b) -> b)
factory1 a f = f a

unbox1 :: (forall b . (a -> b) -> b) -> a
unbox1 t = t id

-- testdata
machine1 = factory1 10
{- suggested tests
machine1 (\x -> x*x)
unbox1 machine1
-}
```

米田の補題を適用すると

`forall b . (a -> b) -> b`
のシグネチャーをもつ関数と
`a`
の型をもつ値は**1**対**1**に対応する





シグネチャー
`forall b . (a -> b) -> b`
をもつ関数は
Hom関手 `(a -> - )` から
恒等関手 `Id` への自然変換
と呼ばれる.

リストを隠し持つ機械の反転工学 unbox2.hs

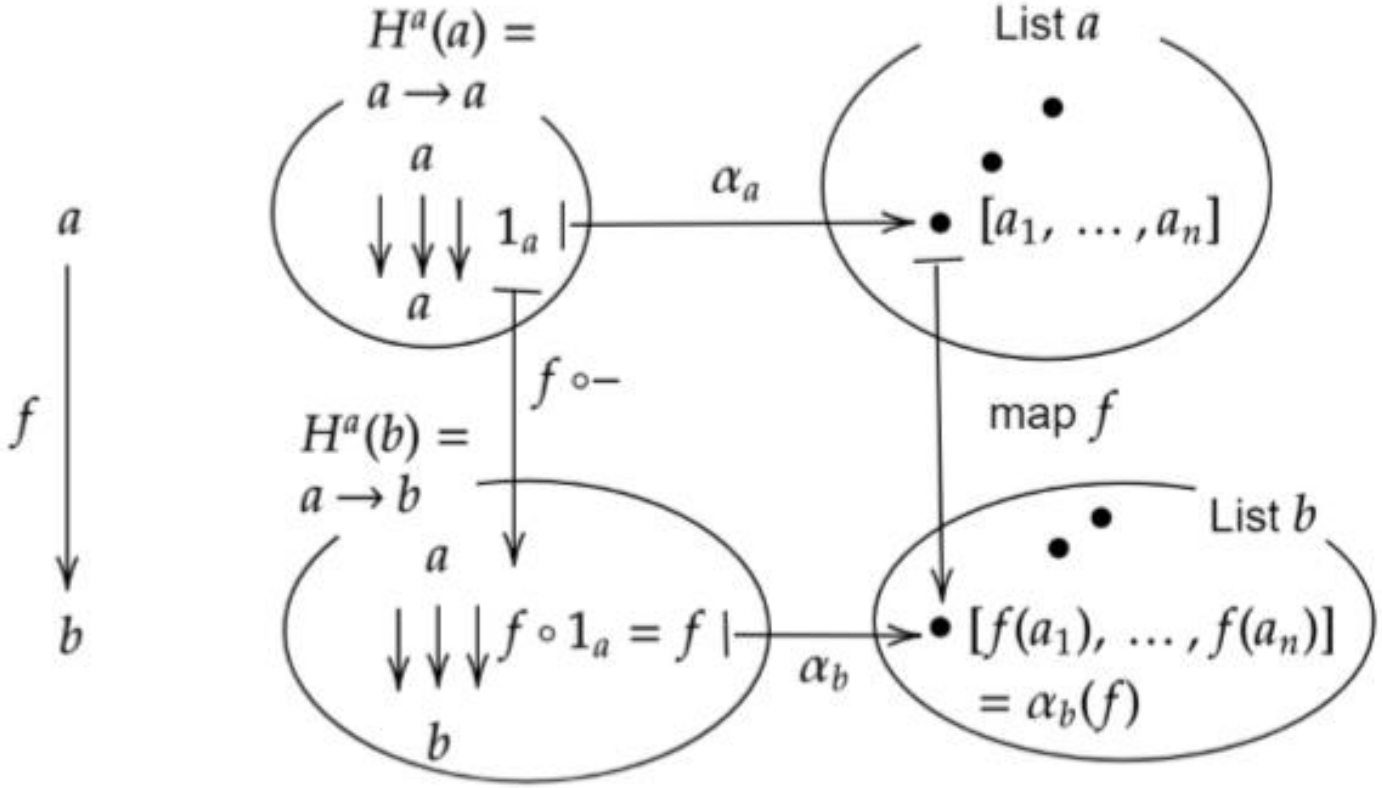
```
{-# LANGUAGE RankNTypes #-}

factory2 :: [a] -> (forall b . (a -> b) -> [b])
factory2 a f = map f a

unbox2 :: (forall b . (a -> b) -> [b]) -> [a]
unbox2 t = t id

-- test data
machine2 = factory2 [1,2,3,4]

{- tests
machine2 (\x -> x*x)
unbox2 machine2
-}
```

Quiz 2:
machine2'
machine2''
のような機械を追加し実験せよ.

米田の補題を適用すると

`forall b . (a->b) -> [b]`
のシグネチャーをもつ関数と
`[a]`
の型をもつ値は1対1に対応する

シグネチャー
`forall b . (a->b)->[b]`
をもつ関数は
Hom関手 `(a -> - )` から
List関手 `[]` への自然変換
と呼ばれる.



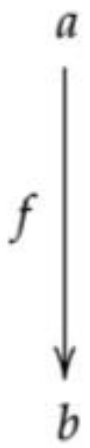
秘密の射をもつ機械の反転工学 unbox3.hs

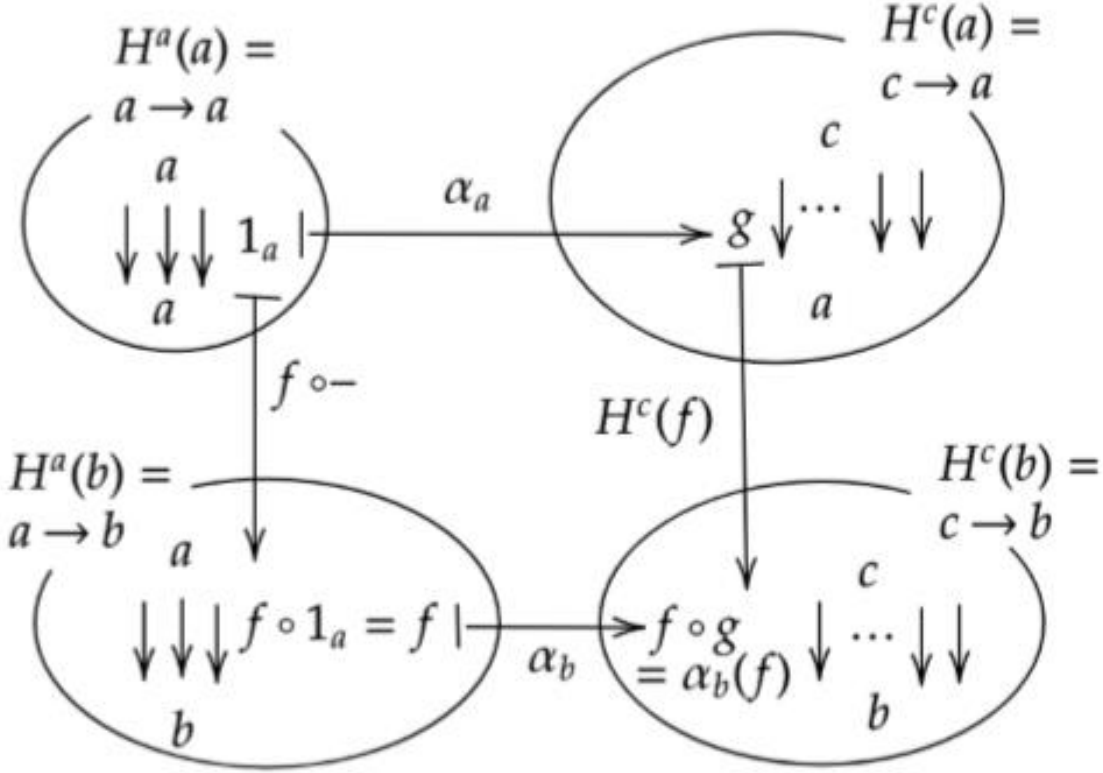
```
{-# LANGUAGE RankNTypes #-}

factory3 :: (c -> a) -> (forall b . (a -> b) -> (c -> b))
factory3 a f = f . a

unbox3 :: (forall b . (a -> b) -> (c -> b)) -> (c -> a)
unbox3 t = t id

-- test data
machine3 = factory3 (\x -> x + 1)

{- tests
map (machine3 (\x -> x*x)) [1..10]

map (unbox3 machine3) [1..10]
-}
```

米田の補題を適用すると

`forall b . (a->b)->(c->b)`
のシグネチャーをもつ関数と
`c->a`
の型を関数は1対1に対応する

シグネチャー
`forall b . (a->b)->(c->b)`
をもつ関数は
Hom関手 `(a-> -)` から
Hom関手 `(c-> -)` への自然変換
と呼ばれる.





参考文献

- 圏論入門 – Haskell で計算する実例から, 雪田修一, 日本評論社, 2020年
- On the Homology Theory of Modules, Nobuo Yoneda, Journal of the Faculty of Science, Imperial University of Tokyo, 1954.
- Types, Abstraction and Parametric Polymorphism, J.C.Reynolds, Information Processing 83, 1983.

課題 AかBを選択せよ.

- A: Quiz 1,2 に取り組め
- B: ALGOL N の簡単なプログラムを作成し，その他の得意なプログラミング言語での実装と比較せよ.
  - The description and the structure of ALGOL N, Nobuo Yoneda, ACM SIGPLAN Notices, Sept. 1971.