Programming Language

2018.7.29

王晨

Agenda

- Ruby
- Clojure
- Haskell

Ruby(1): 类型系统

- 强类型
- 弱类型
- 静态类型
- 动态类型
- Duck Typing & behave as a

Ruby(2) Mixin

- Mixin（实现继承）
- Interface（规格继承）

Ruby(3) 元编程

- Monkey patch
- 代码即数据

Ruby(4) 闭包

- 闭包是一个函数/方法
- 你可以向传递对象那样传递它, 以后调用
- 当这个函数创建的时候, 它记住了该环境下所有的变量的值. 被调用时, 它可以一直访问这些变量, 甚至这些变量已处于环境的外部.

Clojure(1) FP

- 闭包
- 高阶函数
- 惰性计算
- 递归
- no side effect
- immutable
- 引用透明性

Clojure(2) 尾调用

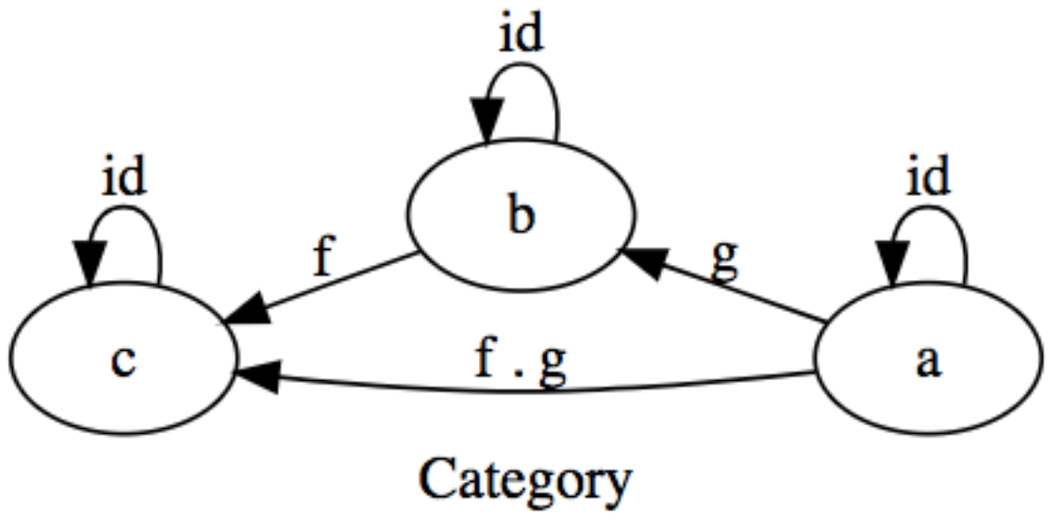
- function story() {
  - 从前有座山，山上有座庙，庙里有个老和尚，一天老和尚对小和尚讲故事：story()}
- function story() {
  - 从前有座山，山上有座庙，庙里有个老和尚，一天老和尚对小和尚讲故事：story()，小和尚听了，找了块豆腐撞死了}

Heskell

- 偏函数调用和Currying
- 类型推导初探

Category(1)

- 一个 范畴Category 包含两个玩意
  - 东西 O （Object）
  - 两个东西的关系，箭头 ~> （ 态射Morphism ）
- 一些属性
  - 一定有一个叫 id 的箭头，也叫做 1
  - 箭头可以 组合 compose



Category

```
class Category (c :: * -> * -> *) where
  id :: c a a
  (.) :: c y z -> c x y -> c x z
```

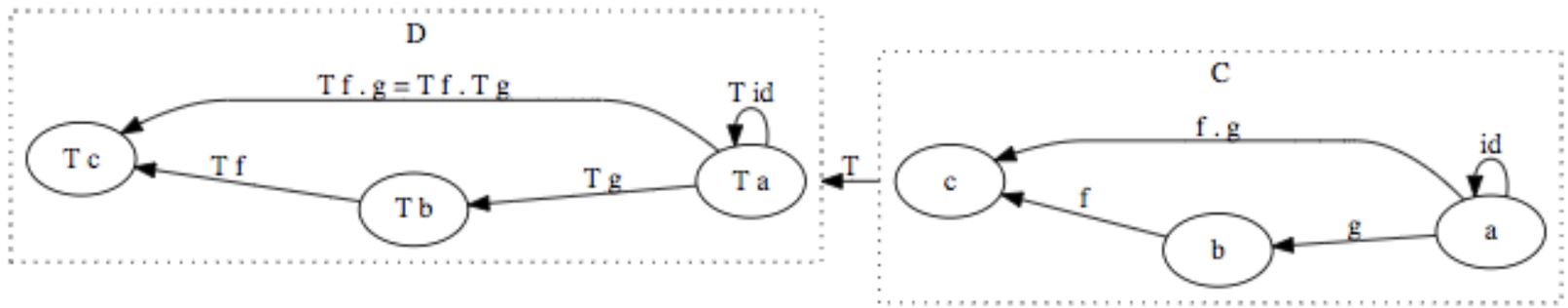
Category(2)

- Haskell 类型系统范畴叫做 Hask，在 Hask 范畴上：
  - 东西是类型
  - 箭头是类型的变换，即 ->
  - id 就是 id 函数的类型 a -> a
  - compose 当然就是函数组合的类型

```
type Hask = (->)
instance Category (Hask:: * -> * -> *) where
  (f . g) x = f (g x)
```

函子

- Functor
  - 是从一个范畴到另一个范畴的映射关系而已。
  - 范畴间 东西的 Functor 标记为 T(O)
  - 范畴间 箭头的 Functor 标记为 T(~>)
  - 任何范畴C上存在一个 T 把所有的 O 和 ~> 都映射到自己，标记为id functor 1C
    - 1C(O) = O
    - 1C(~>) = ~>



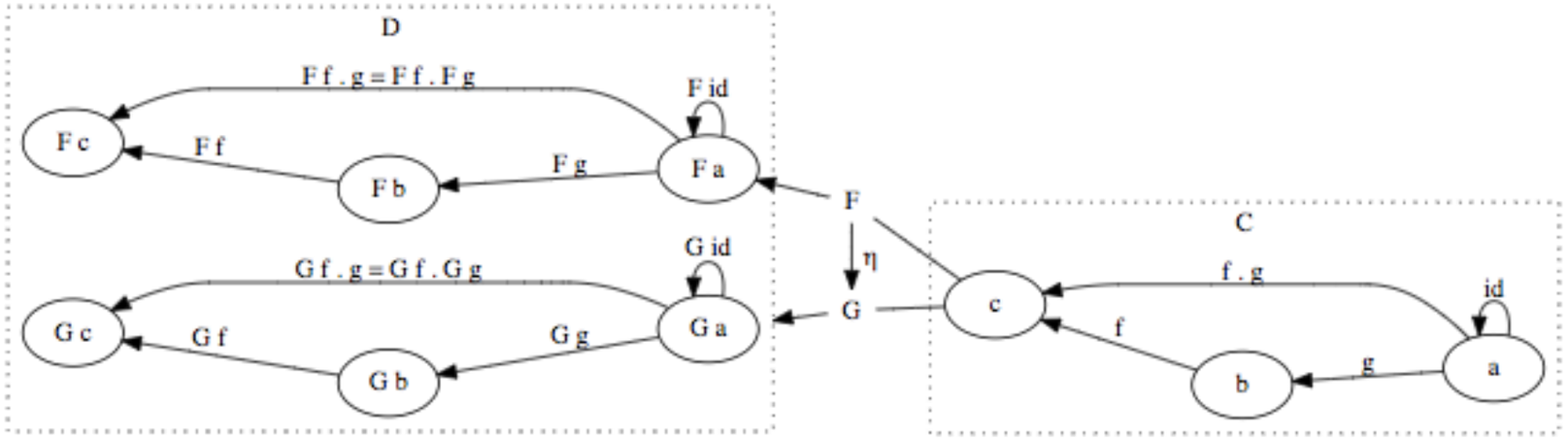

```
class Functor t where
  fmap :: (a -> b) -> (t a -> t b)
```

自函子

- 自函子endofunctor 就是这种连接相同范畴的 Functor，因为它从范畴 Hask 到达同样的范畴 Hask
- 这里的 fmap 就是 T(~>)，在 Hask 范畴上，所以是 T(->)，这个箭头是函数，所以也能表示成 T(f) 如果 f:: a -> b

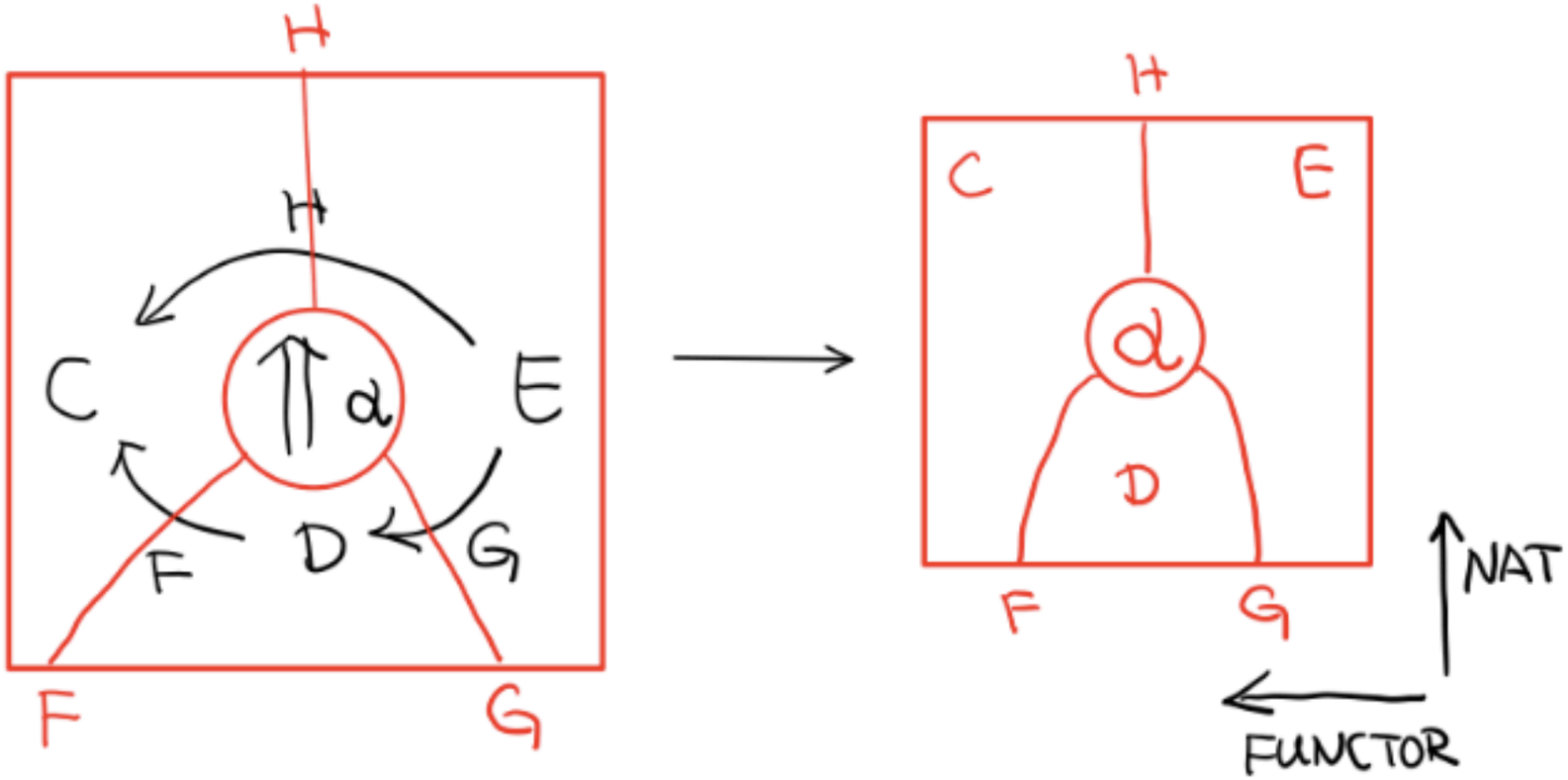
维度升高

- Cat 和 Nat



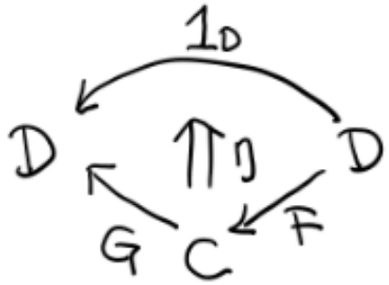
- 一维： Hask， 东西是类型，箭头是 ->
- 二维： Cat， 东西是 Hask， 箭头是 Functor
- 三维： Functor范畴， 东西是Functor， 箭头是自然变换

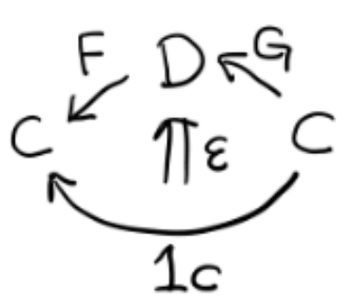
String Diagram

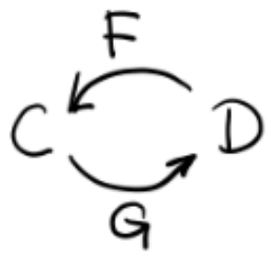


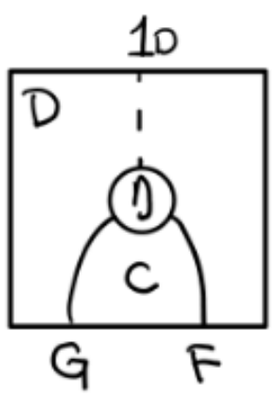
伴随函子（1）

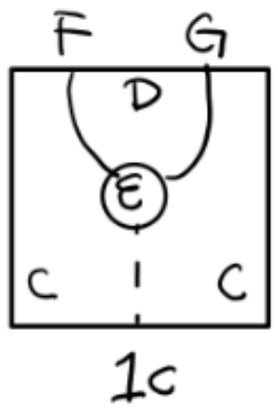
- 一个范畴 C 可以通过函子 G 到 D，再通过函子 F 回到 C，那么 F 和 G 就是伴随函子。
  - η 是 GF 到 1D 的自然变换
  - ε 是 1C 到 FG 的自然变换

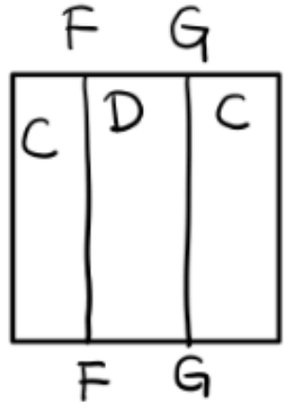






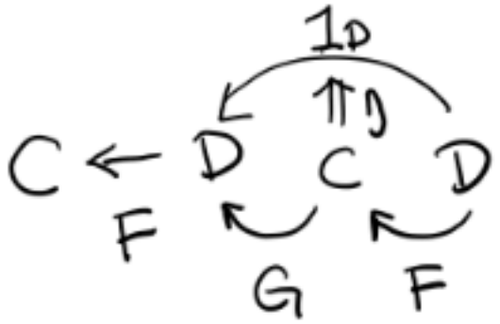


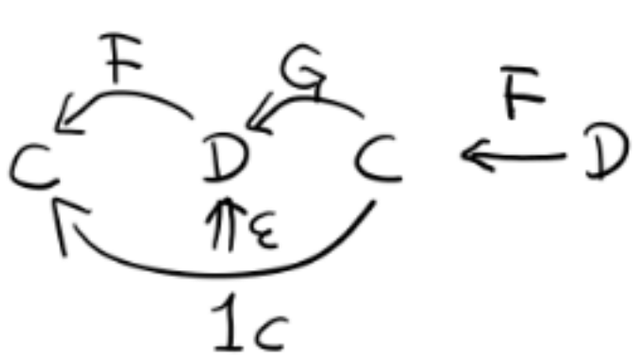


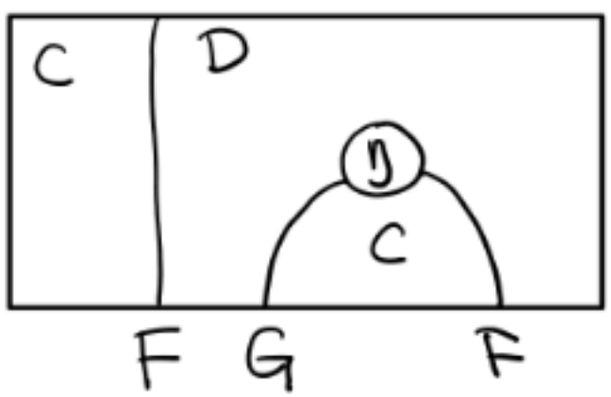


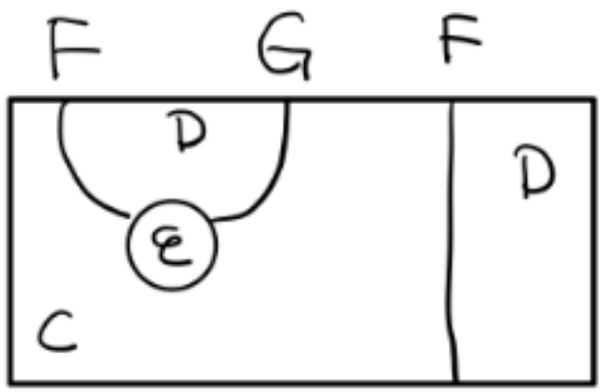
伴随函子（2）

- Fη和εF



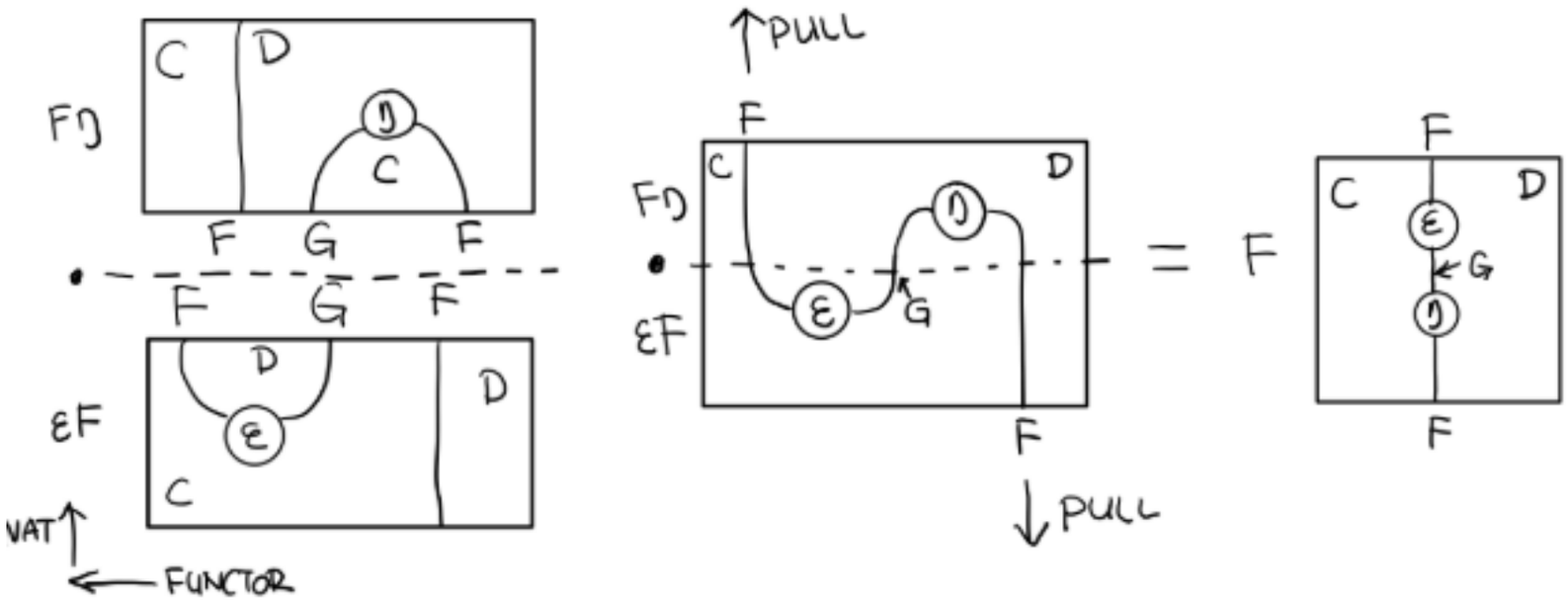






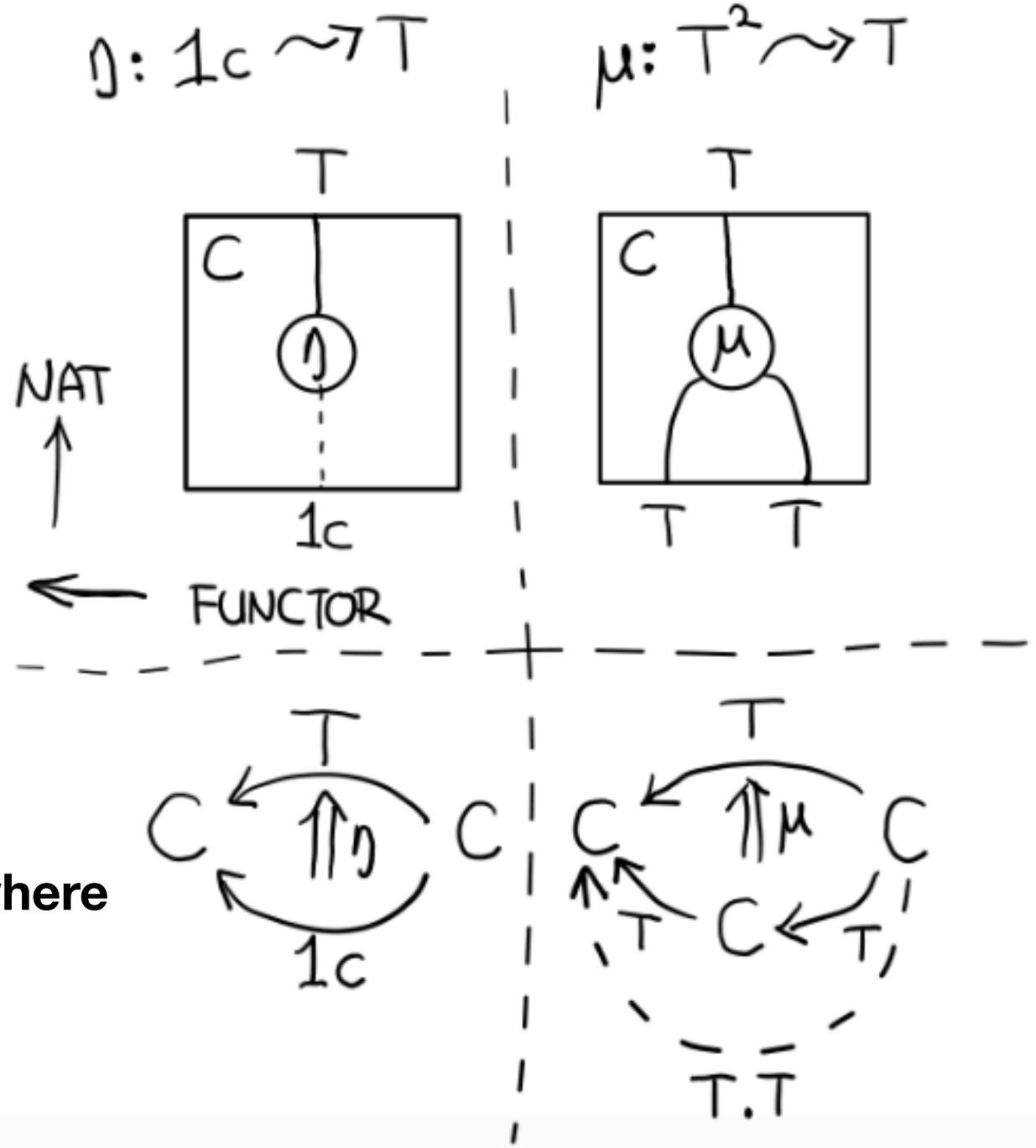
伴随函子（2）

- $F\eta . \varepsilon F = F$



Monad(1)

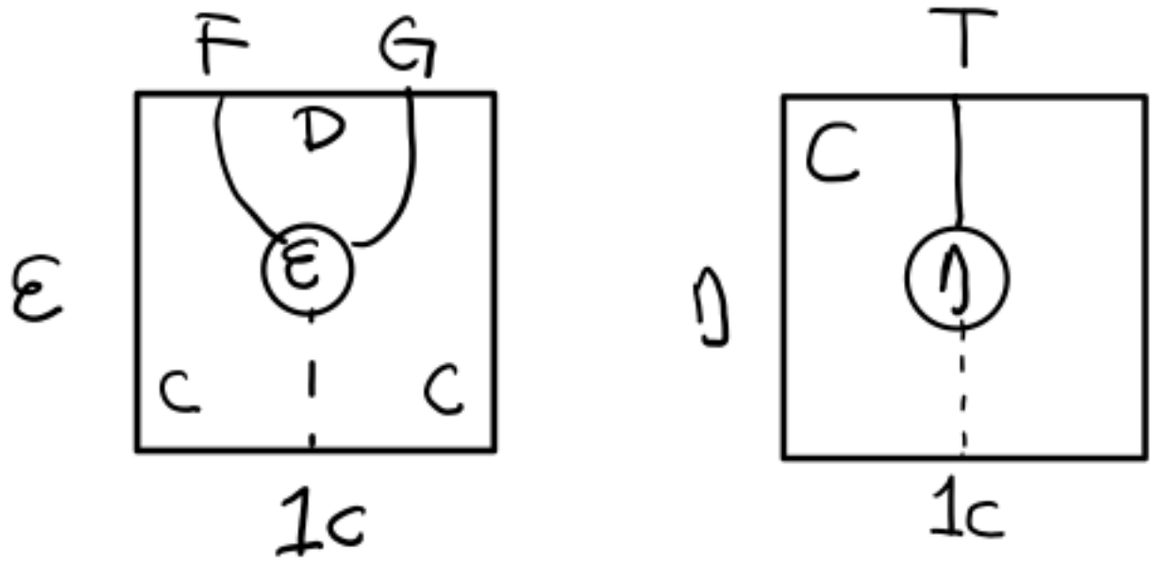
- 单子
  - 首先，它是一个 endofunctor T
  - 有一个从 ic 到 T 的自然变化 η (eta)
  - 有一个从 T2 到 T 的自然变化 μ (mu)



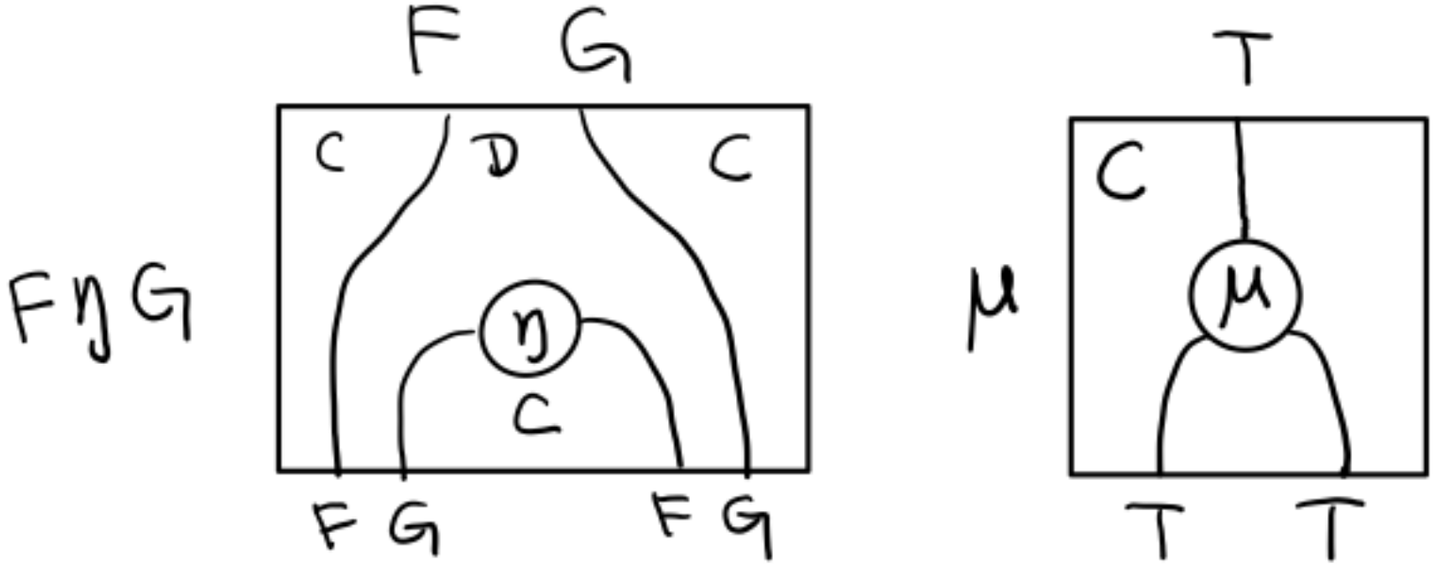

```
class Endofunctor m => Monad m where
  eta :: a -> (m a)
  mu :: m m a -> m a
```

Monad(2)

- 伴随函子的 ε 就是单子的 η

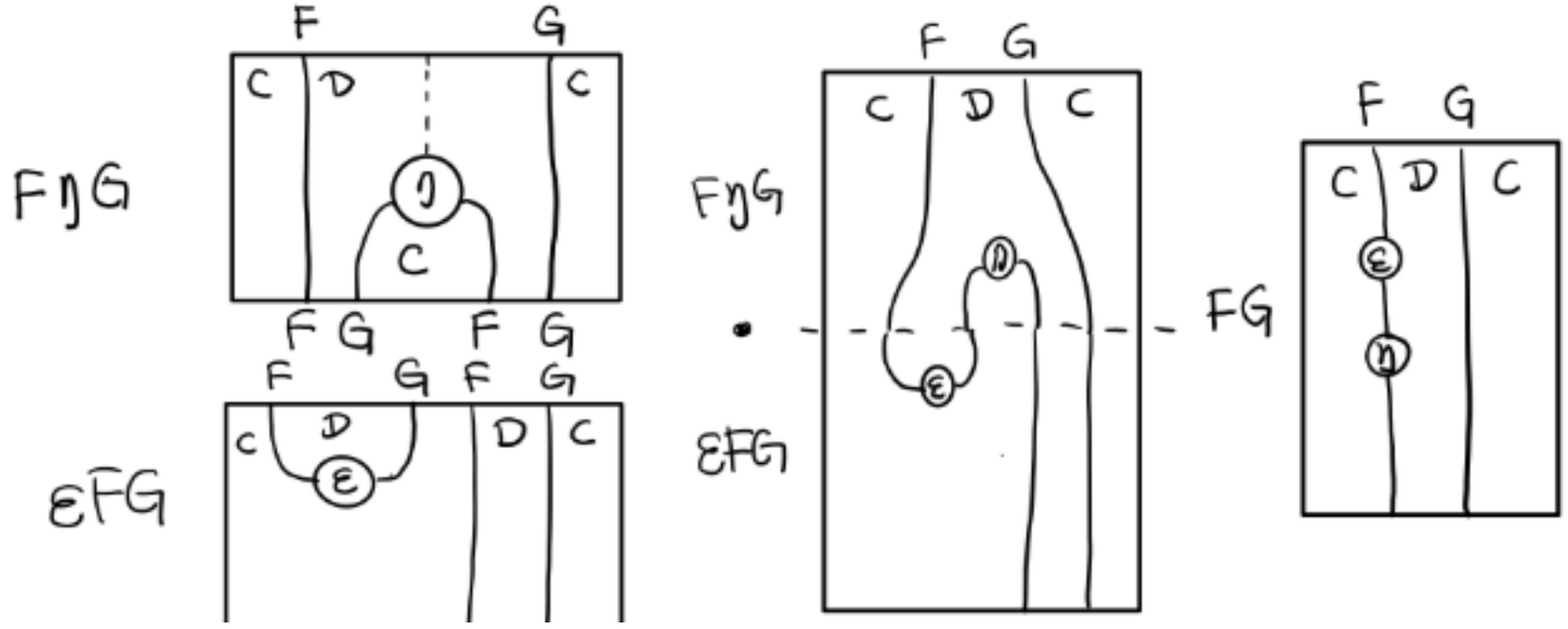


- 伴随函子的 F η G 是函子的 μ



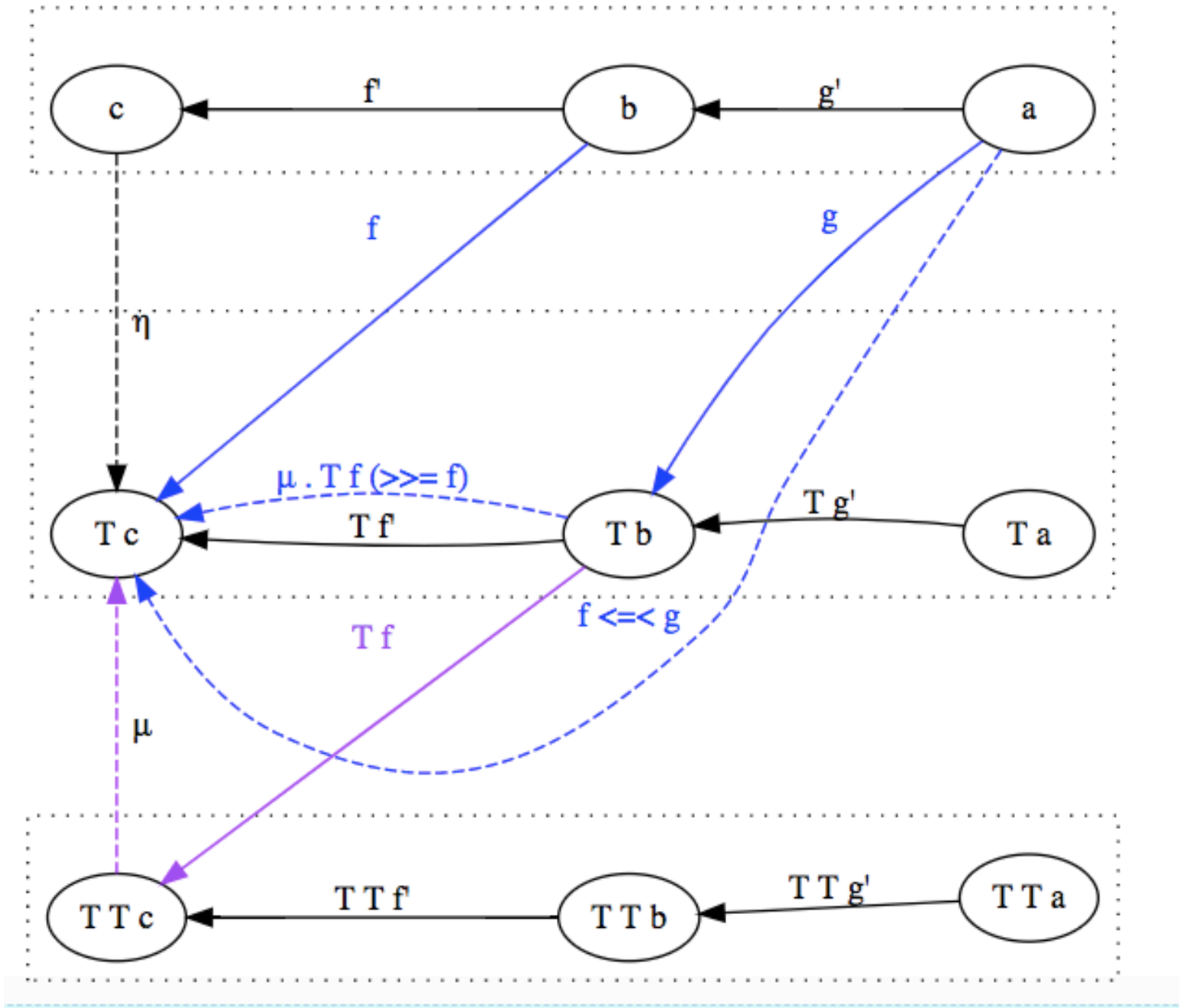
三角等式

- $\mu . T\eta = T = \mu . \eta T$
  - $F\eta\, G . \varepsilon\, F\, G = F\, G$



Kleisli Catergory

- Kleisli Catergory
  - 箭头是 Kleisli 箭头 a -> T b
  - 东西就是c范畴中的东西. 因为 a 和 b 都是 c 范畴上的， 由于T是自函子，所以 T b 也是 c 范畴的



Maybe Factor & Monad