Haskell - P41

Applicative ((->) r)，在Haskell中，typeclass Applicative 的另一个instance 是 ((->) r)

```
instance Applicative ((->) r) where
    pure x  = (\_ -> x)
    f <*> g = \x -> f x (g x)
GHC.Base.liftA2 q f g x = q (f x) (g x)
```

当将一个value 包装进一个 applicative functor 时，
其结果应当始终是 value 本身 (result yielded always has to be the value)
于是 pure 传入 value 并构造3一个[redacted]一元函数
函数始终忽略传入的参数并返回 value
所以对于 (->) r 观察 pure 的类型
有 pure :: a -> [redacted] (r -> a)
由于 -> 是右结合的，实际上等价于 pure :: a -> r -> a
于是在 GHC.Base 中实现3一个二元函数 const
使得一元函数 (const x) 对于任意输入的计算结果都是 x
ⓐ const x ⓐ is a unary function which evaluates to ⓐ x ⓐ for all inputs
即 const :: a -> b -> a

```
const x _ = x
```

于是 GHC.Base 中[redacted]对于 instance Applicative ((->) r) 中 pure 也实现为

```
pure = const
```

对于 instance Applicative ((->) r) 中 <*> 的实现
传入两个 applicative functor 会生成一个 applicative functor
于是传入两个函数会返回一个函数
如 [redacted]:t (+) <$> (+3) <*> (*100) :: (Num a) => a -> a
由于 (+) <$> (+3) <*> [redacted] (*100) 等价于 pure (+) <*> (+3) <*> (*100)
分别分[redacted]析每一个步骤则有
pure (+) :: (Applicative f, Num a) => f (a -> a -> a)
可见此时由于没有指定 context，所以 pure 只是假定3一个 Applicative f
pure (+) <*> (+3) :: (Num a) => a -> a -> a
于是此时生成3一个二元函数
由于 pure (+) <*> (*3) 实质上等价于 [redacted]fmap (+) (*3)
即等价于 (+).(*3)
如 (pure (+) <*> (*3)) 4 5 → 17 , (3x4)+5

Haskell - P42

对于 instance Applicative ((->) r) 中 <*> 的实现

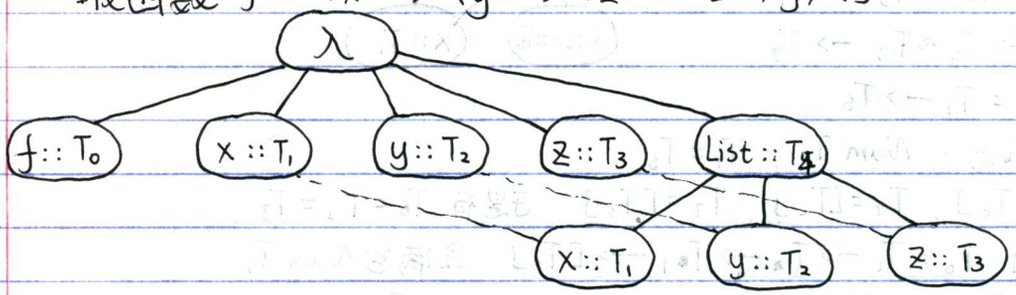
pure (+) <*> (*3) <*> (*4) :: (Num b) => b -> b

于是此时生成 3 个一元函数 f(x) = (x*3) + (x*4)

如 pure (+) <*> (*3) <*> (*4) $ 5 ⟶ 35 , (3*5) + (4*5)

Applicative ((->) r) ，进一步通过 type inference 来分析 <*> 的实现

取函数 f = \x -> \y -> \z -> [x, y, z]



于是有：

$T_0 = T_1 \to T_2 \to T_3 \to T_4$

$T_4 = [T_1]$ (1)

$T_4 = [T_2]$ (2)

$T_4 = [T_3]$ (3)

(1)(2)(3) : $T_1 = T_2 = T_3$

即有 f :: a -> a -> a -> [a]

所以有 $T_0 = T_1 \to T_1 \to T_1 \to [T_1]$

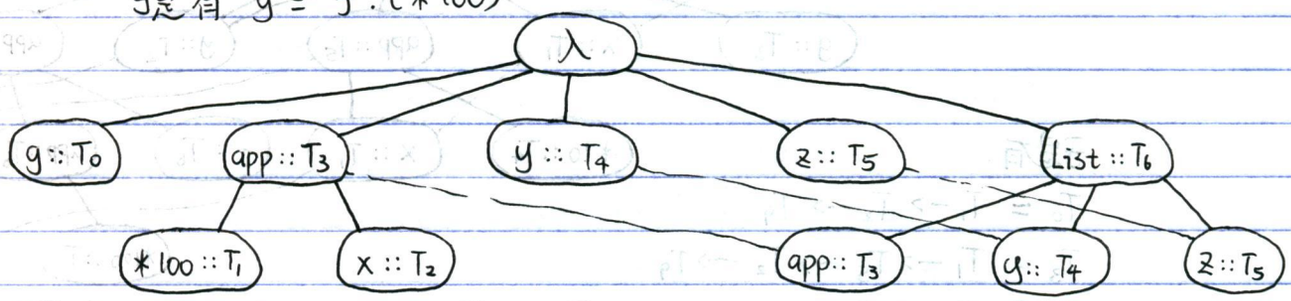
而 pure f :: (Applicative f) => f (a -> a -> a -> [a])

其中 f 是一个依据 context 决定的 Applicative Functor，与函数 f 不同

取函数 g = pure f <*> (*100)

等价地有 g = f <$> (*100) , g = fmap f (*100)

于是有 g = f . (*100)



于是有：$T_0 = T_3 \to T_4 \to T_5 \to T_6$

$T_1 = T_2 \to T_3$

constraint : Num T_2

$T_2 = T_3$

对于传入函数 (*100)

(*100) :: (Num a) => a -> a

$T_6 = [T_3]$

$T_6 = [T_4]$

$T_6 = [T_5]$

对于 List 的 constructor

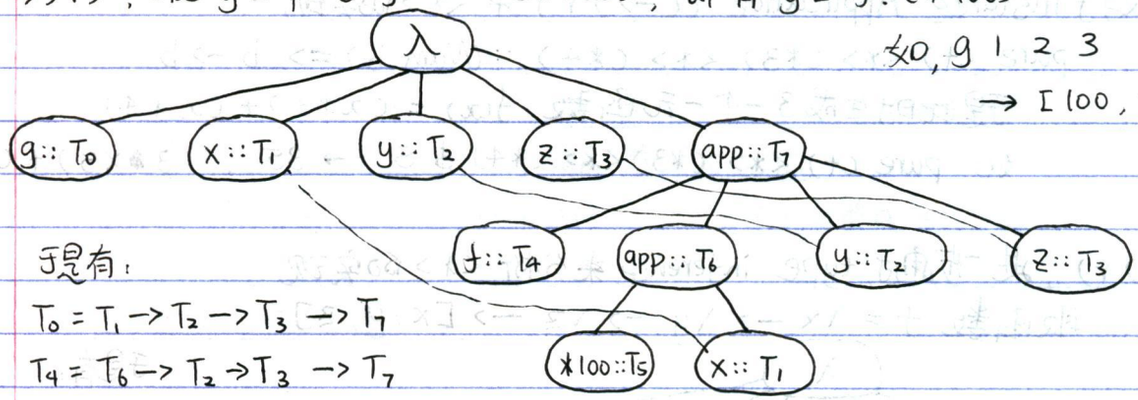
于是有 $T_3 = T_4 = T_5$

所以有 $T_0 = T_2 \to T_2 \to T_2 \to [T_2]$，且满足 Num T_2

即有 g :: (Num a) => a -> a -> a -> [a]

Haskell - P43

Applicative ((->) r)，取 g = pure f <*> (* 100)，即有 g = f . (* 100)



如，g 1 2 3
→ [100, 2, 3]

于是有：

$T_0 = T_1 \to T_2 \to T_3 \to T_7$

$T_4 = T_6 \to T_2 \to T_3 \to T_7$

$T_5 = T_1 \to T_6$

constraint : Num T_1, $T_1 = T_6$

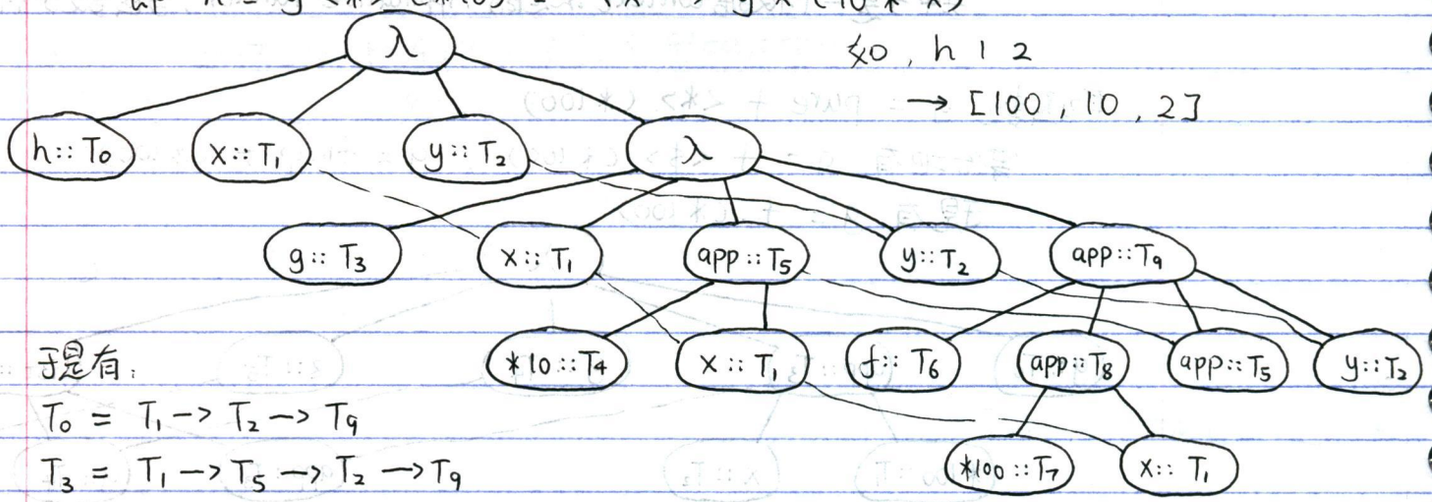
$T_7 = [T_6]$, $T_7 = [T_2]$, $T_7 = [T_3]$, 于是有 $T_6 = T_2 = T_3$

所以有 $T_0 = T_1 \to T_2 \to T_3 \to [T_1]$，且满足 Num T_1

即有 g :: (Num a) => a -> a -> a -> [a]

取函数 h = g <*> (* 10)，此时真正应用了 Applicative 中 <*> 的应用

即 h = g <*> (* 10) = \x -> g x (10 * x)



如，h 1 2
→ [100, 10, 2]

于是有：

$T_0 = T_1 \to T_2 \to T_9$

$T_3 = T_1 \to T_5 \to T_2 \to T_9$

$T_4 = T_1 \to T_5$

constrant : Num T_1, $T_1 = T_5$

$T_6 = T_8 \to T_5 \to T_2 \to T_9$

$T_7 = T_1 \to T_8$

constrant : Num T_1, $T_1 = T_8$

$T_9 = [T_8]$, $T_9 = [T_5]$, $T_9 = [T_2]$

$T_8 = T_5 = T_2$

所以有：$T_0 = T_1 \to T_1 \to [T_1]$，且满足 Num T_1

于是 h 的类型可以由此确定

即有 h :: (Num a) => a -> a -> [a]

注意此处实现了对于一个多参数函数

进行分参参数的复合

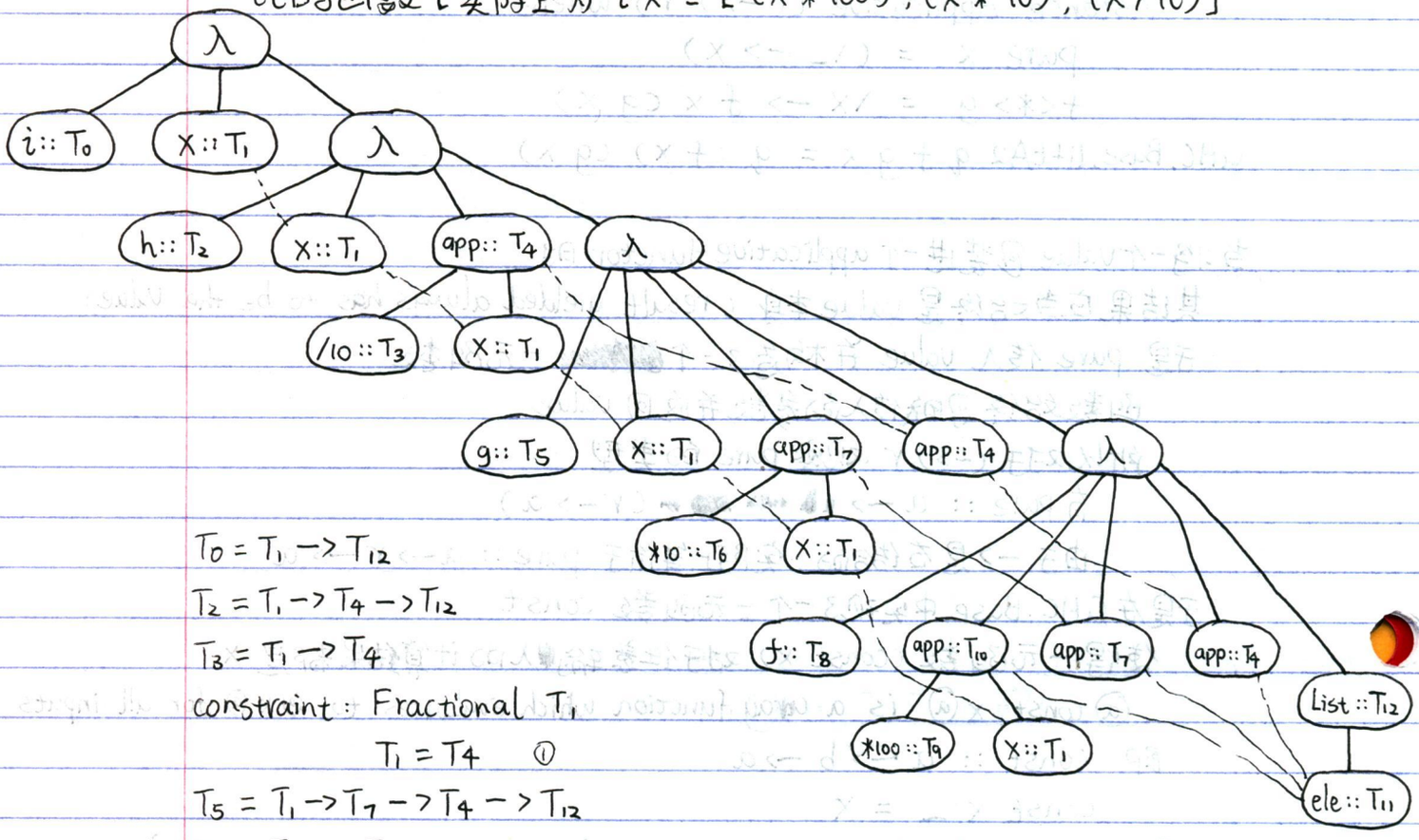
即有 h(x, y) = g(x, (*10) x, y)

= f((*100) x, (*10) x, y)

Haskell - P44

Applicative ((->) r)，取函数 i = h <*> (/10)，即 i = \x -> h x (x / 10)

此时函数 i 实际上为 i x = [(x * 100), (x * 10), (x / 10)]



$T_0 = T_1 \to T_{12}$

$T_2 = T_1 \to T_4 \to T_{12}$

$T_3 = T_1 \to T_4$

constraint : Fractional T_1

$T_1 = T_4$ ①

$T_5 = T_1 \to T_7 \to T_4 \to T_{12}$

$T_6 = T_1 \to T_7$

constraint : Num T_1, $T_1 = T_7$ ②

$T_8 = T_{10} \to T_7 \to T_4 \to T_{12}$

$T_9 = T_1 \to T_{10}$

constraint : Num T_1, $T_1 = T_{10}$ ③

$T_{10} = T_{11}$, $T_7 = T_{11}$, $T_4 = T_{11}$ ④

$T_{12} = [T_{11}]$

由①②③可知，T_1 应当满足 constraint (Fractional T_1)

由①②③④可知，$T_1 = T_4 = T_7 = T_{10} = T_{11}$

于是有 $T_{12} = [T_1]$

则有 $T_0 = T_1 \to [T_1]$

于是有 i :: (Fractional a) => a -> [a]