FOPL-Homework3

昂伟 *PB11011058*

Problem 1

- $quicksort :: (Ord\ t) \Rightarrow [t] \rightarrow [t]$. 令 $p :: t$, 由 $(p : xs)$ 知,$quicksort$ 的参数类型为 $[t]$; 再由 $(quicksort\ lesser) + +[p]$ 知:$quicksort$ 的返回类型为 $[t]$. 由 $< p, \geq p$ 知: t 属于 Ord 类型类.
- $lesser :: (Ord\ t) \Rightarrow [t]$. 由 $quicksort\ lesser$, 且 $quicksort :: (Ord\ t) \Rightarrow [t] \rightarrow [t]$ 知: $lesser :: (Ord\ t) \Rightarrow [t]$.
- $greater :: (Ord\ t) \Rightarrow [t]$. 推导过程类似于 $lesser$.

Problem 2

(a)

$factRec :: (Int \rightarrow Int) \rightarrow Int \rightarrow Int$.
由

$$\begin{aligned} &case\ x\ of \\ &\quad 0\ \rightarrow\ 1 \\ &\quad \_\ \rightarrow\ x\ *\ (g\ (x\ -\ 1)) \end{aligned}$$

知:

$$x :: Int \tag{1}$$

由

$$* :: Int \rightarrow Int \rightarrow Int$$

$$x\ *\ g(x\ -\ 1)$$

知:

$$g :: Int \rightarrow Int \tag{2}$$

由 (1)(2) 得:

$$factRec :: (Int \rightarrow Int) \rightarrow Int \rightarrow Int$$

(b)

$y :: (a \to a) \to a$.
令

$$f :: a \to b,\ y :: (a \to b) \to b$$

由

$$y\ f\ =\ f\ (y\ f)$$

且

$$y\ f :: c, f(y\ f) :: b$$

知:

$$b\ =\ c$$

又在 $f(y\ f)$ 中 $y\ f$ 作为 f 的参数且 $y\ f :: c$
知:

$$a\ =\ c$$

即得

$$y :: (a \to a) \to a$$

(c)

```
fibRec g = \n -> case n of
                        0 -> 0
                        1 -> 1
                        n -> (g (n - 1)) + (g (n - 2))
```

(d)

i

$$\begin{aligned} y\ (f) &= \lambda g.f(g\ g)(\lambda g.f(g\ g)) \\ &= f\ (\lambda g.f(g\ g)(\lambda g.f(g\ g)) \\ &= f\ (y\ f) \end{aligned}$$

ii

Haskell 是静态类型语言, 不支持动态类型，所以 y 组合子定义无法通过 Haskell 静态类型推导系统, 才会产生 $T = T \to t_0$ 的类型错误.

(e)

```
reduceRec g =  \f  \l -> case l of
        []  -> undefined
        [x] -> x
        (x:xs) -> f x (g f xs)
```

Problem 3

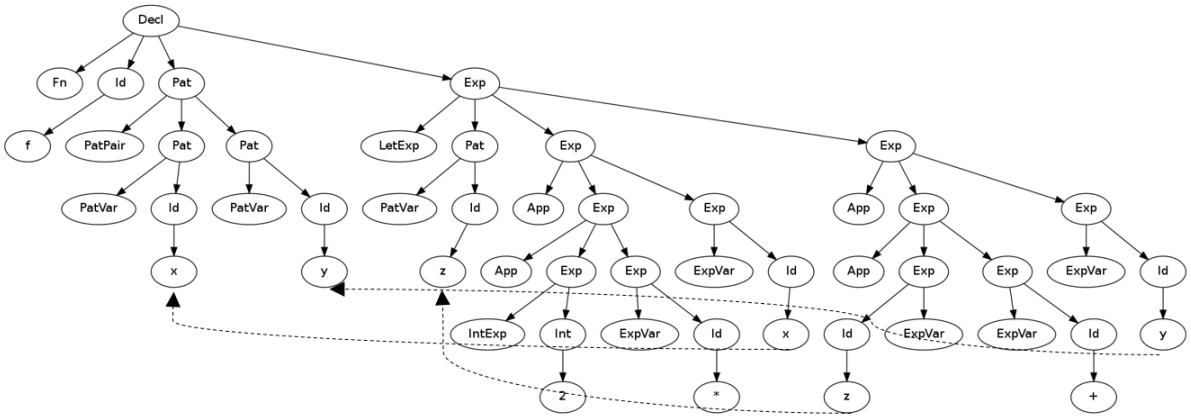
(a)

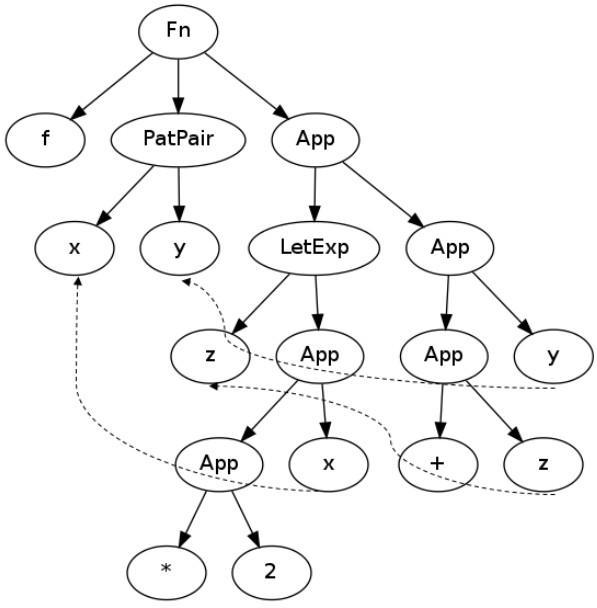
```
myDecl = Fn "f"
            (PatPair (PatVar "x")
                     (PatVar "y"))
            (LetExp
                (PatVar "z")
                (App
                    (App
                        (IntExp 2)
                        (ExpVar "*"))
                    (ExpVar "x"))
                (App
                    (App
                        (ExpVar "z")
                        (ExpVar "+"))
                    (ExpVar "y")))
```

(b)

见Inference.hs

(c)



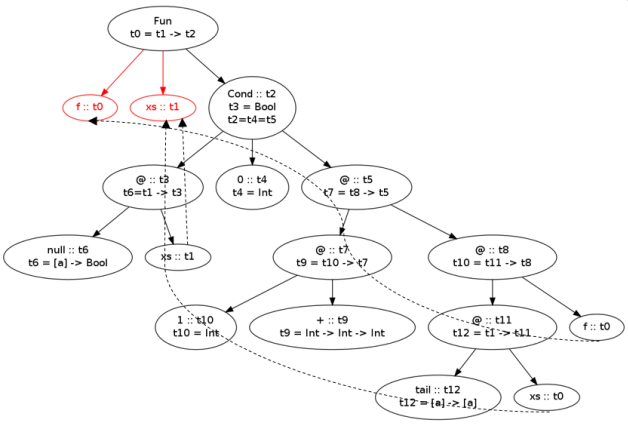


(d)

i

由 *Cond Exp Exp Exp* 知: 第一个 Exp 的类型必为 *Bool*, 第二个, 第三个 Exp 的类型必须相同且为 *Cond Exp Exp Exp* 的类型.

ii



iii

$$t0 = t1 \to t2 \tag{1}$$
$$t2 = t4 = t5 \tag{2}$$
$$t3 = Bool \tag{3}$$
$$t6 = t1 \to t3 \tag{4}$$
$$t6 = [a] \to Bool \tag{5}$$
$$t7 = t8 \to t5 \tag{6}$$
$$t10 = t11 \to t8 \tag{7}$$
$$t12 = t1 \to t11 \tag{8}$$
$$t12 = [a] \to [a] \tag{9}$$
$$t9 = t10 \to t7 \tag{10}$$
$$t2 = Int \tag{11}$$

由 (2)(11) 知:

$$t2 = t4 = t5 = Int \tag{12}$$

由 (4)(5) 知:

$$t1 = [a] \tag{13}$$

由 (1)(12)(13) 知:

$$t0 = [a] \to Int$$

Problem 4

(a)

$$t\_0 = t\_3 \to t\_10 \tag{1}$$
$$t\_3 = (t\_1, t\_2) \tag{2}$$
$$t\_4 = t\_9 \to t\_10 \tag{3}$$
$$t\_9 = (t\_7, t\_1) \tag{4}$$
$$t\_5 = t\_2 \to t\_7 \tag{5}$$
$$t\_5 = Int \to String \tag{6}$$
$$t\_4 = (String, String) \to String \tag{7}$$

由 (5)(6) 知:

$$t\_2 = Int \tag{8}$$

$$t\_7 = String \tag{9}$$

由 (3)(7) 知:

$$t\_9 = (String, String) \tag{10}$$
$$t\_10 = String \tag{11}$$

由 (4)(10) 知:

$$t\_1 = String \tag{12}$$

由 (12)(8) 知:

$$t\_3 = (String, Int) \tag{13}$$

由 (11)(13) 知:

$$t\_0 = (String, Int) \to String \tag{14}$$

所以,

$$g :: (String, Int) \to String$$

(b)

```
h :: (t_1, t_2) -> t_2
y :: t_2
xs :: [t_1]
foldright h y xs :: t_2
```

(c)

我们想要

$$f :: [Int] \to String$$

但实际上由

$$h = g :: (String, Int) \to String$$

得,

$$t_1 :: String,\ t_2 :: Int$$

即 $f :: [Int] \to [Int],\ y = \text{""} :: String = t_2$ 矛盾.

(d)

```
g(n, s) = Concats (show n, s)
```

Problem 5

(a)

```
-- Integer Comparison
dCompInt :: CompD Int
dCompInt = MakeCompD CompareInt

-- List Comparison
dCompList :: CompD a -> CompD [a]
dcompList d = MakeCompD compList where
    compList [] [] = EQ
    compList (x:xs) [] = GT
    compList [] (y:ys) = LT
    compList (x:xs) (y:ys) = if (((?=) d x y) /= EQ)
                             then ((?=) d x y)
                             else (compList xs ys)
```

(b)

```
(?=) (dCompTuple (dCompInt, (dCompList dCompChar)))
                 (length "Hello", "Hello")
                 (length "World", "World"))
(?=) dCompInt (length "Hello) (length "World")
(?=) dCompList dCompChar "Hello" "World"
(?=) dCompChar 'H' 'W'
```

(c)

$f :: (Comp\ t) \Rightarrow [t] \rightarrow [t] \rightarrow Ordering$. 由 $lengthx$ 知: $x :: [t]$; 又 $xx\ ?=\ yy$ 知: $y :: [t]$; 再由 $f\ x\ y\ =\ xx\ ?=\ yy :: Ordering$，即得:$f :: (Comp\ t) \Rightarrow [t] \rightarrow [t] -> Ordering$.

Problem 6

(a)

$MkEqD$ 是一个字典操作, 用于重载时查询对应数据类型的相关函数调用操作;=== 从其第一个参数（即操作字典）中提取判别相等的函数.

(b)

```
instance MyEq a => MyEq (Tree a) where
    (===) (Leaf v1) (Leaf v2) = v1 === v2
    (===) (Node v1 tl1 tr1)
          (Node v2 tl2 tr2) = (v1 === v2)
                            & (tl1 === tl2)
                            & (tr1 === tr2)
    (===) _ _ = False
```

(c)

```
dMyEqTree :: MyEq a -> MyEq (Tree a)
dMyEqTree d = MkMyEqD myEqTree where
    myEqTree (Leaf v1) (Leaf v2) = (===) d v1 v2
    myEqTree (Node v1 tl1 tr1)
             (Node v2 tl2 tr2) =
                            (===) d v1 v2
                             & (myEqTree tl1 tl2)
                             & (myEqTree tr1 tr2)
```

(d)

因为函数 cmp 声明使用了 qualified type $MyEq\ a$. 为了能正确的调用类型 a 所对应的 $(===)$ 函数, 编译器必须对 cmp 进行改写, 即加入 a 类型数据所对应的操作字典.

(e)

```
cmp' :: MyEq a -> a -> a -> String
cmp' d t1 t2 = if (===) d t1 t2
               then "Equal"
               else "Not Equal"
result' = cmp' (dMyEqTree dMyEqInt) test1 test2
```