CSE 130 Final Solution, Fall 2019

Nadia Polikarpova

December 12, 2019

Q1: Lambda Calculus: Fibonacci [10 pts]

Solution without `FIX`:

```
let STEP = \p -> PAIR (SND p) (ADD (FST p) (SND p))

let FIB  = \n -> FST (n STEP (PAIR ZERO ONE))
```

Solution with `FIX`:

```
let STEP = \rec n -> ITE (ISZ n) ZERO (ITE (EQL ONE n) ONE
                     (ADD (rec (DEC n)) (rec (DEC (DEC n)))))

let FIB  = FIX STEP
```

Q2: Datatypes and Recursion [30 pts]

2.1 Tail-Recursive Delete [10 pts]

```
delete :: Id -> [Id] -> [Id]
delete x xs = loop [] xs
  where
    loop :: [Id] -> [Id] -> [Id]
    loop acc []     = acc
    loop acc (y:ys) = if x == y
                        then loop      acc  ys
                        else loop (y : acc) ys
```

2.2 Free Variables [10 pts]

```
freeVars :: Expr -> [Id]
freeVars (Num n)       = []
freeVars (Var x)       = [x]
freeVars (Add e1 e2)   = freeVars e1 ++ freeVars e2
freeVars (Let x e1 e2) = freeVars e1 ++ (delete x (freeVars e2))
```

2.3 Optimize [10 pts]

```
optimize :: Expr -> Expr
optimize (Num n)       = Num n
optimize (Var x)       = Var x
optimize (Add e1 e2)   = Add (optimize e1) (optimize e2)
optimize (Let x e1 e2) = let
                           e1' = optimize e1
                           e2' = optimize e2
                         in if x `elem` freeVars e2'
                              then Let x e1' e2'
                              else e2'
```

Q3: Higher-Order Functions [30 pts]

3.1 List minimum [10 pts]

```haskell
-- | Minimum element of a non-empty list
-- | minimum [4, 1, 1, 2]  ==> 1
minimum :: [Int] -> Int
minimum (x:xs) = foldr min x xs
```

3.2 Bucket [10 pts]

```haskell
-- | bucket xs bs distributes elements from `xs` into `bs`:
-- | bucket [4, 1, 1, 2] [1, 2, 3, 4]  ==>  [[1, 1], [2], [], [4]]
bucket :: [Int] -> [Int] -> [[Int]]
bucket xs bs = map (\b -> filter (== b) xs) bs
```

3.3 Concatenation [10 pts]

```haskell
-- | Concatenate a list of lists
-- | concat [[1, 1], [2], [], [4]]  ==>  [1, 1, 2, 4]
concat :: [[Int]] -> [Int]
concat xss = foldr (++) [] xss
```

Q4: Semantics and Type Systems [20 pts]

4.1 Evaluation 1 [5 points]

Which of these evaluation relations are valid according to the operational semantics of Nano?

(A) `[] ; 1 + x ==> 1` [ ]

(B) `[] ; (\x -> 1) ==> 1` [ ]

(C) `[] ; (\x -> 1) (2 + 3) ==> 1` [X]

(D) `[] ; (\x -> x x) (\x -> x x) ==> <[], x, x x>` [ ]

(E) `[f := <[x:=5],y,x + y>] ; f 1 ==> 6` [X]

4.2 Evaluation 2 [5 points]

Which of the following rules are used in the derivation of the reduction

`[] ; (\x y -> x + y) 5 ==> <[x:=5], y, x+y>`

(A) `E-Num` [X]

(B) `E-Var` [ ]

(C) `E-Add` [ ]

(D) `E-Lam` [X]

(E) `E-App` [X]

4.3 Typing 1 [5 points]

Which of the following typing judgments are valid according to the type system of Nano?

(A) `[x:Int,y:Int] |- x :: Int` [X]

(B) `[x:Int] |- x + y :: Int` [ ]

(C) `[] |- \x y -> x :: Int -> Int -> Int` [X]

(D) `[] |- \x y -> x :: Int -> (Int->Int) -> Int` [X]

(E) `[] |- \x y -> x :: Int -> Int -> Int -> Int` [ ]

4.4 Typing 2 [5 points]

Which of the following rules are used in the derivation of the typing judgment

```
[] |- (\x y -> x + y) 5 :: Int -> Int
```

(A) `T-Num` [X]

(B) `T-Var` [X]

(C) `T-Add` [X]

(D) `T-Lam` [X]

(E) `T-App` [X]