CSE 130 Midterm, Spring 20

Nadia Polikarpova

May 4, 2020

Q1: Lambda Calculus: Reductions [5 pts]

Check the box next to *each* term that is *in normal form*.

(A) `\y -> y (\y -> y)` `[X]`

(B) `(\y -> y) (\y -> y)` `[ ]`

(C) `g (\y -> y) (\y -> y)` `[X]`

(D) `(\y -> y) g (\y -> y)` `[ ]`

(E) `(\y -> y y) (\y -> y y)` `[ ]`

Q2: Lambda Calculus: Functions [15 pts]

```
-- (1) = (C) equality on booleans
let F1     = \x y -> x y (NOT y)

-- (2) = (E) fibonacci
let H2     = \rec n -> ITE (ISZ (DEC n))
                           n
                           (ADD (rec (DEC n)) (rec (DEC (DEC n))))
let F2    = FIX H2

-- (3) = (B) integer division
let H3     = \m i j -> ITE (EQL (INC j) m)
                           (PAIR (INC i) ZERO)
                           (PAIR i (INC j))
let F3     = \n m -> FST (n (\p -> H3 m (FST p) (SND p))
                            (PAIR ZERO ZERO))
```

Q3: Binary Heaps [20 pts]

```haskell
-- | Binary heap datatype:
data BHeap = Leaf | Node Int BHeap BHeap

-- | Height of the heap
height :: BHeap -> Int
height Leaf         = 0
height (Node _ l r) = 1 + max (height l) (height r)

{- Task 2.1: Is full? -}

-- | Is this binary tree full?
-- | We say that a tree is full if has no partially filled levels.
-- | For example:
-- | isFull (Node 5 (Node 3 Leaf Leaf) (Node 4 Leaf Leaf)) ==> True
-- | isFull (Node 5 (Node 3 Leaf Leaf) Leaf)               ==> False
isFull :: BHeap -> Bool
isFull Leaf         = {- (1)(D) -} True
isFull (Node _ l r) = {- (2)(W) -} isFull l && isFull r && (height l == height r)

{- Task 2.2: Insert -}

-- | Insert a new key into the binary heap.
-- | For example:
-- | insert 5 (Node 11 (Node 3 Leaf Leaf) (Node 8 Leaf Leaf)) ==>
-- |     Node 11 (Node 5 (Node 3 Leaf Leaf) Leaf) (Node 4 Leaf Leaf)
-- | (see one more example in fig 1)
-- |
-- | Recall that the pattern h@(Node y l r) matches a value against Node,
-- | but also binds the whole value to h
insert :: Int -> BHeap -> BHeap
insert x Leaf          = {- (3)(I) -} Node x Leaf Leaf
insert x h@(Node y l r)
    | isFull h               = {- (4)(S) -} Node hi (insert lo l) r
    | {- (5)(F) -} isFull l = {- (6)(T) -} Node hi l (insert lo r)
    | {- (7)(D) -} True     = {- (8)(S) -} Node hi (insert lo l) r
```

```haskell
  where
      lo = min x y
      hi = max x y


{- Task 2.3: Tail-recursive insert all -}

-- | Insert all values from a list into a binary heap.
-- | For example:
-- | insertAll [5,3,4] Leaf ==> Node 5 (Node 3 Leaf Leaf) (Node 4 Leaf Leaf)
-- |
-- | Your function must be *tail-recursive*!
insertAll :: [Int] -> BHeap -> BHeap
insertAll []     h = {- (9)(C) -} h
insertAll (x:xs) h = {- (10)(U) -} insertAll xs (insert x h)
```

```
{- A -} 0
{- B -} 1
{- C -} h
{- D -} True
{- E -} False
{- F -} isFull l
{- G -} isFull r
{- H -} Node x l r
{- I -} Node x Leaf Leaf
{- J -} isFull l || isFull r
{- K -} isFull l && isFull r
{- L -} Node y (insert x l) r
{- M -} Node x (insert y l) r
{- N -} Node y l (insert x r)
{- O -} Node x l (insert y r)
{- P -} height l == height r
{- Q -} height l >= height r
{- R -} insert x (insert xs h)
{- S -} Node hi (insert lo l) r
{- T -} Node hi l (insert lo r)
{- U -} insertAll xs (insert x h)
{- V -} insert x (insertAll xs h)
{- W -} isFull l && isFull r && (height l == height r)
```