CSE 130 Final Solution, Spring 2018

Nadia Polikarpova

June 11, 2018

Q1: Lambda Calculus: Sets [20 pts]

1.1 Empty set [5 pts]

```
let EMPTY     = \x -> FALSE
```

1.2 Insert an element [5 pts]

```
let INSERT    = \n s x -> ITE (EQL n x) TRUE (s x)
```

Alternatively:

```
let INSERT    = \n s x -> OR (EQL n x) (s x)
```

1.3 Membership [5 pts]

```
let HAS       = \s x -> s x
```

1.4 Set intersection [5 pts]

```
let INTERSECT = \s1 s2 x -> AND (s1 x) (s2 x)
```

Q2: Datatypes and Recursion: Decision Trees [60 pts]

2.1 Evaluation [10 pts]

```
eval :: Env -> BDT -> Bool
eval _   (Leaf b) = b
eval env (Node x tt tf) =
  if lookup x env
    then eval env tt
    else eval env tf
```

2.2 Negation [15 pts]

```
tNot :: BDT -> BDT
tNot (Leaf b) = Leaf (not b)
tNot (Node x tt tf) = Node x (tNot tt) (tNot tf)
```

2.3 Conjunction [15 pts]

```
tAnd :: BDT -> BDT -> BDT
tAnd (Leaf False) _ = Leaf False
tAnd (Leaf True)  t = t
tAnd (Node x tt tf) t = Node x (tAnd tt t) (tAnd tf t)
```

2.4 Ordered BDTs* [20 pts]

```
tAndOrd :: BDT -> BDT -> BDT
tAndOrd (Leaf False)     _            = Leaf False
tAndOrd (Leaf True)      t            = t
tAndOrd _                (Leaf False) = Leaf False
tAndOrd t                (Leaf True)  = t
tAndOrd l@(Node x lt lf) r@(Node y rt rf)
  | x < y    = Node x (tAndOrd lt r)  (tAndOrd lf r)
  | x > y    = Node y (tAndOrd tr l)  (tAndOrd rf l)
  | x == y   = Node x (tAndOrd lt rt) (tAndOrd lf rf)
```

Q3: Higher-Order Functions [20 pts]

3.1 List reversal [5 pts]

```
reverse :: [a] -> [a]
reverse xs = foldl (\res x -> x : res) [] xs
```

3.2 Absolute values [10 pts]

```
absValues :: [Int] -> [Int]
absValues = map (\x -> if x < 0 then -x else x)
```

3.3 Remove duplicates [15 pts]

```
dedup :: [Int] -> [Int]
dedup = foldr insert []
  where
    insert x ys = x : (filter (/= x) ys)
```

3.4 Insertion Sort* [20 pts]

```
sort :: [Int] -> [Int]
sort xs = foldl insert [] xs
  where
    insert ys x = append (filter (< x) ys) (x : filter (>= x) ys)
    append xs ys = foldr (:) ys xs
```

Q4: Semantics and Type Systems [30 pts]

4.1 Reduction 1 [10 points]

```
E = [f -> <[], \x y -> x + y>]
```

```
[Var]   ------------------------
        E, f => E, <[], \x y -> x + y>
[App-L] --------------------------------
        E, f 1  =>  E, <[], \x y -> x + y> 1
[App-L] ------------------------------------
        E, f 1 2  =>  E, <[], \x y -> x + y> 1 2
```

4.2 Reduction 2 [10 points]

```
E = [f -> <[], \x y -> x + y>]
```

```
[App]   ---------------------------------------------------
        E, <[], \x y -> x + y> 1  =>  [x -> 1], \y -> x + y
[App-L] ---------------------------------------------------------
        E, <[], \x y -> x + y> 1 2  =>  [x -> 1], (\y -> x + y) 2
```

4.3 Typing 1 [10 points]

```
[T-Var] -------------------       -------------------[T-Num]
        [x:Int] |- x :: Int       [x:Int] |- 5 :: Int
[T-Add] ---------------------------------------------
        [x:Int] |- x + 5 :: Int
[T-Abs] -------------------------------
        [] |- \x -> x + 5 :: Int -> Int
```

4.4 Typing 2 [10 points]

```
G = [id -> forall a. a -> a, f -> Int -> Int]
```

```
[T-Var] ------------------------------------
        G |- id :: forall a . a -> a
[T-Inst] --------------------------------------     --------------------[T-Var]
        G |- id :: (Int -> Int) -> Int -> Int      G |- f :: Int -> Int
[T-App] ------------------------------------------------
        G |- id f :: Int -> Int
```

Q5: Prolog: Selection sort [30 pts]

5.1 Insert [10 points]

```
insert(X, Ys, [X|Ys]).
insert(X, [Y|Ys], [Y|Zs]) :- insert(X, Ys, Zs).
```

5.2 Minimum element [10 points]

```
list_min(A, [], A).
list_min(A, [X|Xs], Min) :-
    A1 is min(A, X),
    list_min(A1, Xs, Min).
```

5.3 Selection Sort [10 points]

```
selection_sort([],[]).
selection_sort([X|Xs],[Y|Ys]) :- list_min(X, Xs, Y)
                                , insert(Y,Zs,[X|Xs])
                                , selection_sort(Zs,Ys).
```