6.827 Mid-Term Quiz

Professor Arvind

Name ______________________________

Email ______________________________

Remember to write your name on every page!

This is a closed book quiz.
Access to lecture notes is permitted.
1hr 30min
11 Pages

Problem 1	________	30 Points
Problem 2	________	20 Points
Problem 3	________	32 Points
Problem 4	________	18 Points
Total	________	100 Points

Problem 1 Reduction

To make the expressions in this problem easier to view, we have used braces to underline terms which are contained between matching parentheses.

Part 1-1: (10 points)

Reduce the following λ-calculus expression, in any order you choose, until there are no more redexes in the expression. Give all the intermediate steps.

$$\lambda z.\ \underbrace{(\underbrace{(\lambda x.\,\lambda z.\,z\ \underbrace{(x\ z)})}\ \underbrace{(\lambda f.\,z\ f)})}\ \underbrace{(\lambda x.\,x)}$$

Part 1-2: (10 points)

Reduce the following expression to **normal form** using the **normal order** reduction strategy. Give all the intermediate steps in the reduction. If you discover that the reduction will not terminate, stop and indicate as much.

$$\underbrace{(\lambda x.\ \lambda y.\ x\ \underbrace{(\lambda g.\ \underbrace{(g\ x)}\ \underbrace{(g\ y)})}\ \underbrace{(\lambda x.\ x)})}\ \underbrace{(\lambda x.\ \lambda y.\ x\ y)}$$

Part 1-3: (10 points)

Reduce the following expression to **normal form** using the **applicative order** reduction strategy. Give all the intermediate steps in the reduction. If you discover that the reduction will not terminate, stop and indicate as much.

$$\underbrace{(\lambda x.\, \lambda y.\, x)}\ \underbrace{(\lambda z.\ \underbrace{(\lambda x.\, \lambda y.\, x)}\ z\ \underbrace{(\underbrace{(\lambda x.\, z\, x)}\ \underbrace{(\lambda x.\, z\, x)})})}$$

Problem 2 — Recursion

In class, we defined a fixed-point operator Y which produces the least fixed-point of a recursive definition. That is, it solves the equation:

$$\mathrm{Y}\,\mathrm{F} = \mathrm{F}\,(\mathrm{Y}\,\mathrm{F}) \tag{1}$$

This equation simply says that if we apply the reduction rules to the expression (Y F), we can reduce it to the expression F (Y F).

Part 2-1: (10 points)

We said that the combinator which satisfies this condition is:

$$\mathrm{Y} = (\lambda f.\,(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x)))$$

However, there are many other equally valid definitions for this combinator. Consider the following definition:

$$\mathrm{Y}' = (\lambda x.\,\lambda y.\,y\,(x\,x\,y))\,(\lambda x.\,\lambda y.\,y\,(x\,x\,y))$$

Show that Y' is a least fixed-point operator by showing that it satisfies equation (1).

Part 2-2: (10 points)

Given the following recursive definition:

```
length l = case l of
             [] -> 0
             (x:xs) -> 1 + length xs
```

construct a non-recursive expression for `length` using the fixed-point operator Y.

Problem 3 — Hindley Milner Types

Give the Hindley-Milner types for the following functions. Be sure to give the most general type for any functions which are polymorphic. Assume that there is no overloading, that all arithmetic functions operate on values of type *Int*, and that all comparisons return results of type *Bool*. Some functions may not type, but instead produce a type error. In those cases, indicate that the expression is not type correct and explain why.

3-1: (3 points)

```
curry f x y       =  f (x,y)
```

3-2: (3 points)

```
repeat x          =  let
                         xs = (x:xs)
                     in
                         xs
```

3-3: (6 points)

```
f x y             =  f x y

g x y             =  g y x

h x y             =  if (x == 0) then f x y
                                 else g y x
```

3-4: (4 points)

```
max1 f n m       =  let
                        a = f n
                        b = f m
                        d = a > b
                    in
                        if (f d) then a
                                 else b
```

3-5: (4 points)

```
max2 n m         =  let
                        f x = x
                        a   = f n
                        b   = f m
                        d   = (a > b)
                    in
                        if (f d) then a
                                 else b
```

3-6: (8 points)

Given the following types for `map`, `(&&)` (the boolean AND operator), and `(.)` (an infix operator which composes functions), determine the types of the other three functions:

```
map  :: (a -> b) -> [a] -> [b]

(&&) :: Bool -> Bool -> Bool

(.)  :: (b -> a) -> (c -> b) -> c -> a
```

```
foldr f z l      =  case l of
                      [] -> z
                      (x:xs) -> f x (foldr f z xs)

and              =  foldr (&&) True

all p            =  and . map p
```

3-7: (4 points)

```
unzip            =  let
                        f (a,b) (as,bs) = ((a:as),(b:bs))
                    in
                        foldr f ([],[])
```

Problem 4 Typechecking Using the Class System

This problem is the same as the previous problem, except that we have added overloading to our language with the following type classes:

```
class  Eq a  where
    (==), (/=)       :: a -> a -> Bool

class  (Eq a) => Num a  where
    (+), (-), (*)    :: a -> a -> a
    negate           :: a -> a
    fromInteger      :: Integer -> a

class  (Num a) => Fractional a  where
    (/)              :: a -> a -> a
    recip            :: a -> a

class  (Num a) => Integral a  where
    div, mod         :: a -> a -> a
```

In addition to the types `Bool` and `Int` from the previous problem, we also have the types `Float` and `Char`. There is an instance of the `Eq` class for all four types. The `Num` class is instanced for the numeric types `Int` and `Float`. The only instance of the `Fractional` class is for type `Float` and the only instance of the `Integral` class is for type `Int`.

Remember that the function `fromInteger` in the `Num` class allows us to overload whole-number constants (so `5 :: (Num a) => a`). However, floating-point constants have the type `Float`.

Identify the type of each of the following expressions or indictate that the expression does not type check (and explain why):

4-1: (4 points)

```
y_intercept a b  =  (negate b) / a
```

4-2: (4 points)

```
quadratic a b c  =  ((b * b) - 4 * a * c) / (2 * a)
```

4-3: (5 points)

```
ones_digit n     =  mod n 10.0
```

4-4: (5 points)

```
gcd x y          =  if (y == 0) then x
                                else gcd y (mod x y)
```