CSE340 SPRING 2019 HOMEWORK 4

Due Thursday 14 March 2019 by 11:59 PM on Blackboard

PLEASE READ THE FOLLOWING CAREFULLY

1. Your answers can be handwritten or typed. If you write your answers, you need to have clear handwriting. If we cannot read it we cannot grade it!
2. You should answer the questions in the order they are listed
3. You should submit only one file containing all the answers not multiple files
4. Your submission should be in pdf format

Question 1 . Project 2 difficulties

List the three main bugs that you encountered while coding project 2. If you did not submit project 2, list the three main difficulties you faced.

The answers should be to the point.. For each bug you should say what the symptom was (wrong behavior) and what the bug was.

I plan on collecting the answers, organize them into categories and make them available on blackboard. You get full credit on this question for answering.

Problem 2 (Lambda calculus normal and applicative order). For each of the following identify the redex (if any) that will be reduced first under normal order reduction strategy.

a. x (λx. x x) x No redex

b. x (λx. (λx. x) x) λx. x

(λx. t) t'

c . (λx. (λx. x) x) λx. x

(λx. t) t'

d. (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

(λx. t) t'

e. (λx. (λx. (λx. x) x) x) (x (λx. x) x)

(λx. t) t'

Question 3 (Lambda calculus normal and applicative order). For each of the following expressions identify a redex (if any) that can be reduced first under call by value reduction strategy. Remember that call by value does not specify which redex should be reduced first. It specifies which redex cannot be reduced first (a redex whose right hand side is not a value) or cannot be reduced (under abstraction)

a. x (λx. x x) x No redex

b. x (λx. (λx. x) x) λx. x No reduction under abstraction

c . (λx. (λx. x) x) λx. x

(λx. t) t'

d. (λx. (λy. x) λx. x x) ((λx. (λy. x)) λx. x x)

(λx. t) t'

e. (λx. (λx. (λx. x) x) x) (x (λx. x) x) t' is a value because it has no redex

(λx. t) t'

Question 4 (Binary numbers with lambda calculus). We define a binary representation of numbers. I start by giving some examples.

0_b = pair tru 0
1_b = pair tru 1

$(10)_b$ = pair fls (pair 0 (pair tru 1))
$(11)_b$ = pair fls (pair 1 (pair tru 1))
$(01)_b$ = pair fls (pair 1 (pair tru 0))
$(00)_b$ = pair fls (pair 0 (pair tru 0))

$(110)_b$ = pair fls (pair 0 (pair fls (pair 1 (pair tru 1))))
$(111)_b$ = pair fls (pair 1 (pair fls (pair 1 (pair tru 1))))
$(101)_b$ = pair fls (pair 1 (pair fls (pair 0 (pair tru 1))))

1 representation of 10

$(100)_b$ = pair fls (pair 0 (pair fls (pair 0 (pair tru 1))))

0 representation of 10

In the representation, the bits are listed from left to right from least significant to most significant

In general, the representation of a number $(x_k x_{k-1} x_{k-2} \ldots x_2 x_1)_b$ where all the $x_1, x_2, \ldots, x_k$ values are 0 or 1 and **k > 1** is

pair fls (pair x_1 m)

where m is the representation of $(x_k x_{k-1} x_{k-2} \ldots x_2)_b$

Note that only the last pair has "tru" as the first element to indicate the end of the sequence of bits. The bits are listed from least significant to most significant in the lambda representation

1. Write a function to return the least significant bit of a number given its binary representation. Your function should distinguish between the case when the first element of the pair is tru, in which case it should return the second element of the pair, and the case, in which the first element is false in which case it should return the first of the second element of the pair.

   Answer. LSB = λn. test (fst n) (snd n) (fst (snd n))

2. Write a function that returns the most significant bit of a pair. This will be a recursive function using the approach I went over in class to describe how to implement recursion with lambda calculus. You can use any of the functions that we studied in the definition of your function.

   Answer. **g =** λmsb . λn. test (fst n) (snd n) (msb (snd (snd n)))
   MSB = fix g.

3. (Bonus) Write a function to add two numbers.

Function to add two binary numbers (1/2)

This question is best broken into many pieces to keep the functions manageable. I will start with small helper functions and then define the addition function.

We start with a simple function that takes three bits , representing two bits and a carry bit, and returns the sum. Another function will calculate the carry for three bits. Note that the three bits are either 0 or 1 and are not binary numbers

addbit = λa. λb. λc. (iszero a) ((iszero b) c^1 ((iszero c) 1^2 0^3)) ((iszero b) ((iszero c) 1 0) c)

I explain part of the function

1. The answer is c when a = 0 and b = 0
2. The answer is 1 when a = 0, b != 0 and c = 0
3. The answer is 0 is a = 0 , b !=0 and c !=0

The second function calculates the carry for three bits

carry = λa. λb. λc. (iszero a) ((iszero b) 0 ((iszero c) 0 1)) ((iszero b) ((iszero c) 0 1) 1)

The next recursive function calculates the number of bits in a binary number

g1 = λlen. λn. (fst n) 1 (succ (len (snd (snd n))))

len = fix g1

The next recursive function creates a binary number consisting of n 0's. n should be greater than 0 and is assumed to be represented as a Church number.

g2 = λnz. λn. (= n 1) (pair tru 0) (pair fls (pair 0 (nz (prd n)))

NZ = fix g2

The next recursive function extends a binary number b with n 0's

g3 = λext. λn. λb. (fst b) (pair fls (pair (snd b) (NZ n))) (pair fls (pair (fst (snd b)) (ext (snd (snd b)))))

extend = fix g3

The next function takes two binary numbers as parameters and returns a pair consisting of one of the two numbers as is and the other extended so that the two numbers are equal in length

```
mk_eq_len = λa. λb. test (= (len a) (len b))
                         (pair a b)
                         test ((> (len a) (len b))
                                      (pair          a                    (extend (- (len a) (len b)) b) )
                                      (pair (extend (- (len b) (len a)) a)          b                    )
```

Function to add two binary numbers (2/2)

The next recursive function adds two numbers that have the same number of bits, givem an initial carry value

```
g4 = λaddeq. λa. λb. λc.
            (fst a)                                          // 1 bit numbers
                (iszero (carry (snd a) (snd b) c))           // if there is no carry by adding the two 1-bit
                                                             // numbers with the initial carry value
                      (pair tru (addbit (snd a) (snd b) (snd )c) )  // make a 1-bit number by adding bits
                                                             // otherwise make a two bit number:
                                                             //  - one bit for addition
                                                             //  - one bit for carry bit

                      (pair fls (pair (addbit (snd a) (snd b) c) ( pair (tru  (carry (snd a) (snd b) c )

                // if the two numbers are not 1-bit numbers, then the LSB of the sum is obtained
                // by adding the two LSBs with the initial carry
                // and the remaining bits of the sum is obtained by recursively adding the remaining bits
                // of the two numbers with the new initial carry being the carry obtained of the two
                // LSBs with the initial carry
                ( pair fls ( pair (addbit (LSB a) (LSB b) c) ( addeq (snd (snd a)) (snd (snd b)) (carry (LSB a) (LSB b) c) ) ) )


add_eq_len = fix g4


add = λa. λb.  add_eq_len (fst (mk_eq_len a b)) (snd (mk_eq_len a b)) 0
```

Question 5 (scoping). Give the output of the following program under

1. static scoping
2. dynamic scoping

```
int a;
int b;

void f()
{
   int a;
   int b;

   a = 2;
   b = 20;

   { int a;
     a = a + b;
     g();
   }
   a = a+b;
   g();

}

void g()
{
   print a;
   print b;
}

int main ()
{
    a = 1;
    b = 10;
    f();
    g();
}
```

Show your work!