POPL 2024 Monsoon Final Exam

Instructions:

1. Total Time: **180min** Total Points: **180**
2. Write your roll no. at the top right of the title page.

---

1 Abstract Reduction Systems (ARS) [15 points]

1. [5 points]: Define what is a normal form in an ARS.
2. [5 points]: Define what it means for an ARS to be confluent. Give a non-trivial example of a confluent ARS.
3. [5 points]: Given a non-trivial example of an ARS that is confluent but has no normal forms. A picture of the ARS is sufficient.

2 Continuation Passing Style (CPS) [15 points]

4. [15 points]: Convert the following program into CPS. (Hint: Assume both g and f are potentially recursive.)

```
(define (f g n)
  (if (= (g 0) n)
      5
      (+ 4 (f g (- n 1)))))
```

3 Mapcode [45 points]

5. [30 points]: Consider the 'iterative' version of mergesort, which takes an array, looks for the largest two adjacent sequences of sorted sub-arrays and merges them. It starts from the left end (i=0) and stops

when the end of the array is reached (i=N, where N is the length of the array). Design a mapcode system (mathematical definition is adequate) in which merge is treated as a primitive operation. Make sure your definition handles the boundary cases (N=0 and N=1) correctly. Assume merge is a primitive (You don't need to write the definition of merge, you only need to write its specificication, i.e., what is does.)

6. [15 points]: Draw the trace for the iterative mergesort on the array [9 3 5 8 2 1 7 6].

4 Rewrite systems and Computation Diagrams [15 points]

7. [15 points]: Consider the classical recursive definition of mergesort given in pseudocode below:

```
mergesort(a) = a      if  |a| <= 1
             = merge(mergesort(a1), mergesort(a2)),
   where split(a)=[a1,a2]
```

`split` splits the array into two equal halves (or nearly equal with the first half smaller by one if the array is odd sized).

Draw the computation diagram starting from the array [9 3 5 8 2 1] and ending in [1 2 3 5 8 9]. Use the primitives `split`, `merge` and appropriate projection and tupling functions $(\_, b)$ and $(a, \_)$: $a \xrightarrow{(\_,b)} (a, b)$ and $b \xrightarrow{(a,\_)} (a, b)$.

5 Evaluation by AST Annotation [15 points]

8. [15 points]: Evaluate the program below by drawing an AST and annotating it with environments and values.

```
(recfuns ([even? (n) (if (= 0 n) #t
                         (if (= n 1) #f
                             (odd? (- n 1))))]
          [odd? (n) (if (= 0 n) #f
                         (if (= n 1) #t
                             (even? (- n 1))))])
    (even? 3))
```

Assume that = and - are keywords.

6 SimpliPy [45 points]

9. [45 points]: Consider the program below:

```
a = 5
def f(a, b):
    x = 5
    def g(a):
        nonlocal x
        y = x + a*b
        x = x-1
        return y
    return g
g = f(2, 3)
v = g(4)
# done
```

Construct the compact trace of SimpliPy notional machine (without stores). Clearly indicate the following:

- a. Lexical blocks and decvars of each block. [5 points]
- b. Control transfer functions next and err, call and return. [5 points]
- c. Contro Flow graph. [5 point]
- d. The partial order between environments. [5 points]
- e. The environment state variable with assignment events along with their timestamps (index of the state transition in the trace). [10 points]
- f. The trace of the contexts with transitions annotated by the control transfer function and the update of the appropriate variables if any at that transition. Include the name of each variable prefixed by the environment where it resides. [15 points]

7 Lambda calculus [15 points]

9. Consider the lambda calculus expression

```
(lambda (z)
  ((lambda (x)
     (lambda (y) ((x z) y)))
   (lambda (x) x)))
```

- a. Draw the above expression as a tree. [3 points]
- b. Identify all the beta redexes, if any. [3 points]
- c. Identify all the eta redexes, if any. [3 points]
- d. Reduce the expression, to a normal form, if it exists (i.e., without any redexes). [6 points]

8 Church Numerals [15 points]

15. Here is a representation of naturals in Church numeral form: The `zero`, `succ` and the `add` functions are given below: (Note that `:` denotes curried application and `define:` is used to define curried functions.)

```
(define: (zero f x) x)
; equivalent to
; (define zero (lambda (f) (lambda (x) x)))
(define: (succ n f x) (f (: n f x)))
; equivalent to
; (define succ
;   (lambda (n)
;     (lambda (f) (lambda (x) (f ((n f) x))))))

(define one (succ zero))
(define two (succ one))
(define three (succ two))

(define (show-nat n)
  (: n 0 add1))

(define: (add m n) (: n succ m))
(show-nat (: add two three)) ; => 5
```

Complete the following definitions:

```
;;; m*n
(define: (mul m n)
_____________________________________)

(show-nat (: mul two three)) ; => 6
;;; m^n
(define: (exp m n)
_____________________________________)

(show-nat (: exp two three)) ; => 8

;;; tetration: m^(m^(m^ (... ^m))) [n times]
(define: (tetr m n)
_____________________________________)

(show-nat (: tetr two three)) ; => (2 ^ (2 ^ 2)) = 2^4
= 16
```

[Hint: define `mul` in terms of `add` and `exp` in terms of `mul`, etc.]