Final Exam: COSE212 Programming Languages, Fall 2025

Student Id: Student Name:

Problem 1. (10pts) Define the following function using

```
fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a.
```

The function `concat_map` applies a function `f` to every element of a list and concatenates all resulting lists.

```
let concat_map f lst =
fold_left (fun acc x -> (①)) (②) lst
```

For example,

```
concat_map (fun x -> [x; x]) [1;2;3] = [1;1;2;2;3;3].
```

Problem 2. (10pts) Consider the two functions for list reversal:

```
let rec reverse1 l =
  match l with
  | [] -> []
  | hd::tl -> (reverse1 tl)@[hd]

let reverse2 l =
  let rec aux acc l =
    match l with
    | [] -> acc
    | hd :: tl -> aux (hd :: acc) tl
  in aux [] l
```

1. Does `reverse1` produce the correct result for all inputs?
2. Does `reverse2` produce the correct result for all inputs?
3. What is the time complexity of `reverse1`?
4. What is the time complexity of `reverse2`?

Problem 3. (10pts) Consider the following program written in the language with implicit references:

```
let create =
  proc (init)
    (proc (step) (init := init + step; init)) in
let inc1 = (create 5) in
let inc2 = inc1 in
let a = (inc1 1) in
let b = (inc2 1)
```

What are the values of `a` and `b` at the end of the program?

Problem 4. (10pts) Let g be the function defined by

$$fix\lambda f.(\lambda x.\text{if } x = 0\text{? } 0 : x + f(x-1))$$

where fix denotes a fixpoint combinator (e.g., the Z-combinator). Compute $g(10)$.

Problem 5. (10pts) Consider the program:

```
let fix =
    proc (f) ((proc (x) f (proc (y) ((x x) y)))
              (proc (x) f (proc (y) ((x x) y))))
```

1. After rewriting this program in valid Python syntax, would it be executed by Python? Explain why.

2. After rewriting this program in valid OCaml syntax, would it be executed by OCaml? Explain why.

Problem 6. (10pts) Consider the following program:

```
let id = proc (x) x in
  let f = proc (y) (id (y + 1)) in
  if (id true) then (f 10) else (f 20)
```

1. Is the program accepted by the simple type system? Explain why.

2. Is the program accepted by the let-polymorphic type system? Explain why.

Problem 7. (10pts) What is the value of the program?

```
let a = 5 in
  let f = proc (x) (proc (y) (x + y + a)) in
    let a = 20 in
      let g = f a in
        let x = 1 in
          (g 3) + ((f 2) 4)
```

1. With static scoping:
2. With dynamic scoping:

Problem 8. (10pts) We can represent and manipulate *pairs* in the lambda calculus using suitable encodings:

pair $v\ w$	:	pair of v and w
fst p	:	first component of p
snd p	:	second component of p

Suppose we define pair as follows:

$$\mathsf{pair} = \lambda f.\, \lambda s.\, \lambda b.\, b\ f\ s$$

We would like to define fst and snd so that they satisfy:

$$\begin{aligned} \mathsf{fst}\ (\mathsf{pair}\ v\ w) &= v, \\ \mathsf{snd}\ (\mathsf{pair}\ v\ w) &= w. \end{aligned}$$

Complete the definitions:

1. $\mathsf{fst} = \lambda p.\ \boxed{???}$
2. $\mathsf{snd} = \lambda p.\ \boxed{???}$

Problem 9. (10pts) Recall that the following two expressions are semantically equivalent:

`let x =` e_1 `in` e_2 $\equiv$ `(fun x ->` e_2`)` e_1

1. Does OCaml, which supports let-polymorphism, accept the following code? Explain why.

```
let f = fun x -> x in
  if f true then 1 else ((f f) 2)
```

2. Does OCaml accept the following code? Explain why.

```
(fun f ->
  if f true then 1 else ((f f) 2)) (fun x -> x)
```

Problem 10. (20pts) O/X questions. Leave a blank when uncertain. Each correct answer gets you 1 point but you lose 1 point for each wrong answer.

1. Consider the set R of points (x, y) on the 2D integer grid:

   $$R = \{(x, y) \in \mathbb{Z}^2 \mid x \geq 0, y \geq 0, x + y \equiv 0 \pmod 3\}$$

   R can be defined inductively as follows:

   $$\frac{}{(0,0) \in R} \qquad \frac{(x, y) \in R}{(x+1,\ y+2) \in R} \qquad \frac{(x, y) \in R}{(x+2,\ y+1) \in R}$$

2. With static scoping, the following program

   ```
   let b = 3 in
   let p = proc (x) proc (y) (x := 4; b) in
     ((p <b>) <b>)
   ```

   evaluates to 3.
3. The following program evaluates to -1.

   ```
   let f = proc (x) (
     let counter = ref 0
     in (counter := !counter + 1; !counter))
   in let a = (f 0)
   in let b = (f 0)
   in (a - b)
   ```

4. The following program evaluates to 103.

   ```
   let r = { x := 0, y := 0 } in
   let p = &r.x in
   let q = p in
   let r = { x := 100, y := 0 } in
   (*p := *p + 1; *q := *q + 1; r.x + *p + *q)
   ```

5. Recall the Church encoding of booleans: true and false are represented by $\underline{true} \equiv \lambda t.\lambda f.t$ and $\underline{false} \equiv \lambda t.\lambda f.f$. We can then define the logical-and operator as follows: $\lambda b.\lambda c.b\ c\ \underline{true}$.
6. The type inference algorithm is sound and complete with the simple type system.
7. $\mathsf{unify}(\alpha \to \beta, \beta \to \mathsf{int}, \emptyset) = \{\alpha \mapsto \beta, \beta \mapsto \mathsf{int}\}$.
8. Consider the domain of a programming language:

x, y	$\in$	Id		identifier (variable)
l	$\in$	$Addr$		address (memory location)
n	$\in$	$\mathbb{Z}$		integer
b	$\in$	$\mathbb{B}$	$=$	$\{true, false\}$
r	$\in$	$Record$	$=$	$Id \to Addr$
v	$\in$	Val	$=$	$\mathbb{Z} + \mathbb{B} + \{\cdot\} + Record$
σ	$\in$	Env	$=$	$Id \to Addr + Procedure$
M	$\in$	Mem	$=$	$Addr \to Val$
		$Procedure$	$=$	$(Id \times \cdots) \times Expression \times Env$

   This language supports high-order functions.
9. The nameless representation of the program

   ```
   (let a = 5 in proc (x) (let y = x-a in x-y)) 7
   ```

   is

   ```
   (let 5 in proc (let (#0-#1) in (#1-#0))) 7
   ```

10. Quantum programming languages can, in principle, solve NP-complete problems in polynomial time.