Final Exam

COSE212 Programming Languages, Fall 2015

Instructor: Hakjoo Oh

Problem 1 (10pts) Natural numbers are inductively defined as follows:

$$n \quad \to \quad \circ \mid S\, n$$

where $\circ$ denotes 0, $S\ \circ$ denotes 1, $S\ (S\ \circ)$ denotes 2, and so on.

1. Define a function

   $$\mathsf{add} : n \times n \to n$$

   that adds two natural numbers.
2. Define a function

   $$\mathsf{mul} : n \times n \to n$$

   that multiplies two natural numbers.

Problem 2 (10pts) The common pattern of the functions that accumulate something over a list can be captured by the higher-order function `fold`:

```
let rec fold f l a =
  match l with
  | [] -> a
  | hd::tl -> f hd (fold f tl a)
```

Re-write the following functions using `fold`:

1. ```
   let rec length l =
     match l with
     | [] -> 0
     | hd::tl -> 1 + length tl
   ```
2. ```
   let rec append x y =
     match x with
     | [] -> y
     | hd::tl -> hd::(append tl y)
   ```

Problem 3 (10pts) Consider the minimal yet Turing-complete programming language:

$$E \quad \to \quad x \mid \texttt{proc}\ x\ E \mid E\ E$$

1. Define its semantics with static scoping. The domain is given below.

   $$\begin{array}{rcl} \mathit{Val} & = & \mathit{Procedure} \\ \mathit{Procedure} & = & \mathit{Var} \times E \times \mathit{Env} \\ \mathit{Env} & = & \mathit{Var} \to \mathit{Val} \end{array}$$

2. Define its semantics with dynamic scoping. The domain is given below.

   $$\begin{array}{rcl} \mathit{Val} & = & \mathit{Procedure} \\ \mathit{Procedure} & = & \mathit{Var} \times E \\ \mathit{Env} & = & \mathit{Var} \to \mathit{Val} \end{array}$$

Problem 4 (10pts) Convert the following programs into the lexical-address-based nameless representation:

1. `let a = 1 in let b = 2 in a + b`
2. ```
   let x = 3
     in proc (y)
         let z = (y - x)
         in (x - z + y)
   ```

Problem 5 (10pts) Assuming static scoping for procedures, compare the behaviors and final values of the following two programs.

1. ```
   let f = let counter = ref 0
           in proc (x) (counter := !counter + 1;
                        !counter)
   in let a = (f 0)
      in let b = (f 0)
   in (a - b)
   ```
2. ```
   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)
   ```
   ```

Problem 6 (10pts) Infer the type of $(\lambda x.x)\ 1$:

$$\underbrace{\underbrace{(\lambda x.\ \underbrace{x}_{1})}_{2}\ \underbrace{1}_{3}}_{4}$$

1. (5pts) Generate type equations.
2. (10pts) Solve the equations using the unification algorithm. Explain each step clearly.

Problem 7 (20pts) Consider the following language:

$$E \to \mathsf{true} \mid \mathsf{false} \mid n \mid E_1 + E_2 \mid \text{if } E_1\ E_2\ E_3$$

and the lambda calculus:

$$L \to x \mid \lambda x.L \mid L_1\ L_2$$

We write $\underline{E}$ for the equivalent lambda term in L: that is, if E goes to a value v and $\underline{E}$ goes to a value l in lambda term, then $\underline{v} = l$. Define $\underline{E}$:

Problem 8 (20pts) O/X questions:

1. $\{3n \mid n \in N\}$ ($N = \{0, 1, 2, 3, \ldots\}$) is the only set S that satisfies the following two properties:
   (a) $0 \in S$, and
   (b) if $n \in S$, then $n + 3 \in S$
2. Determining the values of program variables is a static property.
3. C supports call-by-reference for procedure calls.
4. Computers came first than programming languages.
5. C's pointers, structs, set-jumps/long-jumps, gotos, local blocks, and loops are all syntactic sugars of eager-evaluating λ-calculus.
6. All syntactically correct programs run OK in this language:
$$\begin{array}{lcl} C & \to & x := E \mid C; C \\ E & \to & Z \mid B \\ Z & \to & n \mid Z + Z \mid x \\ B & \to & \mathit{true} \mid \mathit{false} \mid Z < Z \end{array}$$
7. There is only one redex in $((\lambda x.\lambda y.x)\ 1)\ 2$.
8. The factorial function can be defined by
$$\mathit{fact} = Y(\lambda f.\lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n * f(n-1))$$
   where Y is the Y-combinator.
9. We can design a sound and complete type system for Java.
10. It is possible for the lambda calculus to simulate all language constructs of Java.