CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Control

Ejercicio 1 **10 Pt**

Considere el siguiente lenguaje:

```
;; abstract syntax
;; <expr> ::= (num <num>) | (id <sym>) | (add <expr> <expr>)
;;          | (fun <sym> <expr>) | (app <expr> <expr>)
(deftype Expr
  (num n)
  (id x)
  (add l r)
  (fun arg body)
  (app f-name f-arg))


;; values
;; <value> ::= (numV <num>) | (closureV <sym> <expr> <env>)
(deftype Value
  (numV n)
  (closureV id body env))

;; eval :: Expr Env -> Value
(define (eval expr env)
  (match expr
    [(num n) (numV n)]
    [(id x) (env-lookup x env)]
    [(add l r) (num+ (eval l env) (eval r env))]
    [(fun id body) (closureV id body env)]
    [(app f e) (def (closureV the-arg the-body the-clos-env) (eval f env))
               (def the-ext-env (extend-env the-arg (eval e env) the-clos-env))
               (eval the-body the-ext-env)]))
```

donde las funciones `num+`, `env-lookup` y `extend-lookup` están definidas como en clase.

Responda las siguientes preguntas con respecto al mismo:

(a) [2.5 Pt] ¿`eval` representa un interprete puramente sintáctico, un intérprete puramente meta, o una combinación de ambos? Justifique su respuesta.

(b) [2.5 Pt] ¿El lenguaje es eager o lazy? ¿A partir de qué parte del intérprete infiere su respuesta?

(c) [2.5 Pt] ¿El lenguaje presenta funciones de primer orden o de primera clase? Justifique su respuesta.

(d) [2.5 Pt] ¿El lenguaje tiene scoping estático o scoping dinámico? ¿A partir de qué parte del intérprete infiere su respuesta?

Ejercicio 2 **14 Pt**

(a) [8 Pt] En términos de espacio requerido, ¿cuál es el problema (más grave) que le encuentra a la implementación de clausuras dada en el intérprete del Ejercicio 1?

(b) [6 Pt] ¿Cómo podría corregir dicho problema y qué parte del intérprete debería modificar para ello?

Observación: No se pide que reimplemente nada, sino que explique coloquialmente cómo podría corregir el problema (y en caso de que se implementase, qué parte del intérprete sería la afectada).

Ejercicio 3 **20 Pt**

(a) [3 Pt] Jaqueline sostiene que todos los lenguajes que definimos en clase son súper limitados porque puede escribir sólo funciones que toman un único argumento. Javiera, por el contrario, sostiene que eso no es una limitación para nada y que no haría falta modificar ninguno de los lenguajes. ¿Qué opina Ud.?

(b) [3 Pt] Pedro insiste en que los lenguajes con una estrategia de evaluación *call-by-name* necesariamente van a tener un scope dinámico. Pablo, en cambio, insiste en que es imposible tener scope dinámico en un lenguaje con *call-by-name*. ¿Qué opina Ud.?

(c) [4 Pt] A Francisco se le acaba de ocurrir una idea para mejorar la eficiencia de los lenguajes que hemos visto en clase: Cada vez que invocamos a una función con un argumento dado, memorizamos el resultado (creando una tabla asociativa), de tal manera que si más adelante volvemos a invocar a la función con el mismo argumento, no hace falta recalcular el resultado, sino que podemos recuperarlo de la tabla. ¿Esa optimización sería válida para todos los lenguajes que vimos en clase? Justifique su respuesta.

(d) [3 Pt] Felipe cree que la optimización de Francisco no tiene ningún sentido porque eso es justamente lo que hace la *lazy evaluation*, que ya fue implementada en clase. ¿Qué opina Ud.? Justifique su respuesta.

(e) [3 Pt] En clase dijimos que el hecho de si una invocación a función es por la cola o no, es una propiedad *estática*. ¿Qué significa concretamente eso? ¿Qué implicancias tiene si uno quiere implementar una herramienta que determine las invocaciones por la cola de un programa?

(f) [4 Pt] El proceso de ejecución de los lenguajes que vimos en clase tiene una estructura bien clara: está compuesto por dos fases, que se ejecutan secuencialmente; la primera es el *parsing*, y la segunda es la evaluación. ¿Cómo se vería afectada esta estructura si extendemos los lenguajes para incorporar macros? (¿habría que quitar o agregar alguna fase? ¿habría que modificar alguna de las fases ya existentes?, ¿habría que reordenar las fases?, etc.)

Ejercicio 4 **16 Pt**

Considere el siguiente fragmento de intérprete de un lenguaje con estructuras de datos mutables (cajas), escrito usando el *store-passing style*:

```
(define (interp expr env sto)
  ...
  [(seqnn exprs)
       (foldr (λ (e vs) (interp e env sto))
              (v*s #f sto)
              exprs)]
  ... )
```

El operador `seqnn` debe evaluar una sequencia *arbitraria* de expresiones de izquierda a derecha, generalizando el `seqn` visto en clase que estaba restringido a sólo a *dos* expresiones; `exprs` es una lista arbitrariamente larga de expresiones.

(a) [6 Pt] ¿Qué problema(s) tiene este intérprete?

(b) [5 Pt] Escriba un programa que exponga dicho problema. ¿A qué valor debería reducir dicho programa? ¿A qué valor efectivamente reduce siguiendo el intérprete de arriba?

(c) [5 Pt] Corrija el intérprete.