Control 2 – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter

14 de Mayo 2007

sin apuntes

1. **(1pt)** Considere la siguiente función Haskell:

```
zzz :: Int -> [a] -> [b] -> [(a, b)]
zzz _ [] _ = []
zzz _ _ [] = []
zzz 0 (a:as) (b:bs) -> []
zzz n (a:as) (b:bs) -> (a, b) : zzz (n-1) as bs
```

   a) Explique el significado de la primera linea de la definición.

   b) ¿Qué hace esta función? De un ejemplo de uso.

   c) ¿Cómo definiría esta función en una linea, reusando funciones estándares de Haskell?

   d) ¿Qué es el valor de la siguiente expresión?: `zzz 5 ones`, donde `ones` es la lista definida por `ones = 1 : ones`.

2. **(1pt)** En un interprete con funciones de *primer orden* (tal como el F1WAE) ¿se puede soportar definiciones recursivas? ¿sería necesario recurrir a un proceso de construcción como el visto en el caso de lenguajes con funciones de *primera clase* (tal como el FAE)? Explique.

3. **(3pt) Primeros pasos hacia objetos**

   a) **(0.75pt)** Defina la función `(point x y)` tal que:

```
> (define p (point 2 3))
> (p 'getX)
2
> (p 'getY)
3
```

   Hint: ¿qué tipo de valores retorna la función `point`*? ¿qué mecanismo estudiado en clase permite que una función tenga "estado"?*
   Hint: `'getX` *es un símbolo. Los símbolos pueden ser comparados con* `eq?`*.*

b) **(0.5pt)** Lo anterior se parece mucho a un programa con objetos básicos, con `point` siendo la función "constructor" de objetos puntos. ¿Qué es un objeto en este caso? ¿un método?

c) **(0.5pt)** Agregue el método `dist` que permita calcular la distancia al origen de un punto[1]:

```
> (define p (point 1 1))
> (p 'dist)
1.4142135623730951
```

d) **(0.75pt)** Cambie la definición de `point` de modo que el método `dist` use *los métodos* `getX` y `getY` (y *no* accede directamente a las variables `x` y `y`). ¿en qué mecanismo de Scheme se tiene que apoyar para que esto funcione?

e) **(0.5pt)** Si bien nuestros puntos son muy parecidos a objetos, no es posible con la definición actual que se pueda *cambiar* la posición de un punto. De una nueva definición de `point` que soporte lo siguiente:

```
> (define p (point 2 3))
> (p 'setX 10)
> (p 'getX '())
10
```

Hint: para simplificar, asuma que todos los métodos aceptan un parámetro, que se ignora cuando no es necesario.
Hint: acuérdese de como implementamos caching de evaluación perezosa.

4. **(1pt)** El siguiente interprete soporta funciones de primera clase. Se muestra la definición de la semántica de la aplicación de función:

```
(define (interp exp env)
  (type-case FAE exp
    ...
    (app (fun-expr arg-expr)
      (local ((define fun-val (interp fun-expr env))
              (define arg-val (interp arg-val env)))
        (interp (fun-body fun-val)
                (aSub (fun-param fun-val)
                 arg-val env))))))
```

¿Cómo caracterizaría las funciones de primera clase de este lenguaje? ¿Podría usar este interprete para implementar los objetos de la pregunta anterior? ¿Por qué?

[1] La función Scheme para calcular la raíz de un número es `sqrt`.

Pauta control 2

-- 1 ------------------

(a) Describe la firma de una función llamada zzz, que recibe como argumentos un entero "n" y 2 listas.

(b) Lo mismo que take n (zip a b). (Retorna una lista con los primeros "n" pares formados con los elementos de las listas, donde el primer elemento es de la

primera lista y el segundo de la segunda.)

Ej.

```
alumnos   = ["fulano", "sutano", "mengano"]
nots      = [7.0, 6.9, 4.0]
mejorNota = zzz 1 alumnos notas.
```

(c) zzz2 n a b = take n (zip a b)

(d) Una función que toma una lista "L" como parámetro y forma min(5,length(L)) pares donde el primer elemento del par es un 1 y el segundo se saca de la

lista L.

-- Pregunta 2 ------------------

Como la lista de funciones para el interprete se define a priori (es decir no podemos definir nuevas funciones dentro del código de F1WAE) entonces una

función puede llamarse a si misma (ie, recursivamente) sin problemas, ya que al hacer el lookup dentro de la lista de funciones encontrará la definición.

-- Pregunta 3 ------------------

```
; parte  (a) y (c)
```

```
(define (point x y)
   (lambda (fun)
     (cond
       ((eq? fun 'getX) x)
       ((eq? fun 'getY) y)
       ((eq? fun 'dist) (sqrt (+ (* x x) (* y y))))
       ))
   )

; parte (b)

;  Un objeto en este caso es una clausura a una función (lambda)
definida dentro
;  de la función point.
;  Un método es un parámetro para la lambda

; parte (d)

(define (point2 x y)
   (local ((define this
             (lambda (fun)
               (cond
                 ((eq? fun 'getX) x)
                 ((eq? fun 'getY) y)
                 ((eq? fun 'dist) (sqrt (+ (* (this 'getX) (this 'getX))
(* (this 'getY) (this 'getY)))))
                 ))
             )
           ) this)
   )

; parte (e)

(define (point3 x y)
   (local ((define xval (box x))
           (define yval (box y))
           (define this
             (lambda (fun param)
               (cond
                 ((eq? fun 'getX) (unbox xval))
                 ((eq? fun 'getY) (unbox yval))
                 ((eq? fun 'setX) (set-box! xval param))
```

```
                ((eq? fun 'setY) (set-box! yval param))
                ((eq? fun 'dist) (sqrt (+ (* (this 'getX '()) (this
'getX '())) (* (this 'getY '()) (this 'getY '()))))
                ))
            )
          ) this)
   )


-- P4 -------------------------
```

Son funciones de scope dinamico, por lo tanto no hay closures y debido a esto, no se puede hacer objetos como en la p3 (no hay el mecanismo para cerrar).
TOdo esto porque la interpretacion de un app se hace en el environment actual.