Auxiliar Seis - CC41A
Lenguajes de Programación
Repasando

Oscar E. A. Callaú
oalvarez@dcc.uchile.cl

`Santiago - Chile, 27/Abr/2008`

1. Curry en Scheme

- Muestre, con un ejemplo simple, cómo puede ver si Scheme soporta currying de funciones o no.
- Defina en Scheme la función `curry` que acepta cómo argumento una función de dos parámetros (de tipo $(\alpha \times \beta) \rightarrow \gamma$) y retorna su versión *currificada* (de tipo $\alpha \rightarrow (\beta \rightarrow \gamma)$).
- Porqué su definición no sería válida en un lenguaje con scope dinámico? De un ejemplo (asumiendo un dialecto de Scheme con scope dinámico) en el cual su definición daría un resultado erroneo.

2. Lazy vs Eager

Implemente en Scheme la función `if0` que toma tres parámetros, `e1`, `e2` y `e3`, tal que evalua `e2` si `e1` vale 0, `e3` sino. Por qué no es posible, en Scheme, definir `if0` para que se use de la siguiente manera?:

```
(if0 (- 4 5)
     (write "zero")
     (write "not zero")))  --> "not zero"
```

1. Cambie el programa anterior para que funcione con su definición de `if0`.
2. Defina `if0` en Haskell, y explique por qué el programa anterior ahora sí funciona tal cual.

3. CFAE/L + IF

Dada la semántica del `if` en el ejercicio 2. Implemente lo en el lenguaje `CFAE/L`.

- Sin considerar *strictness point*. Además elabore un par de test que verifiquen si la implementación es apropiada.
- Dada su implementación anterior. Es necesario colocar algún punto de evaluación estricta? Redefina su implementación, según sea el caso.

4. Repaso de Haskell

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
```

1. Explique el significado de la primera linea de la definición.
2. Qué hace esta función? De un ejemplo de uso.
3. Cómo definiría esta función en una línea, reusando funciones estándares de Haskell?
4. Qué es el valor de la siguiente expresión?: `zzz 5 ones`, donde `ones` es la lista definida por `ones = 1 : ones`.

5. Árboles infinitos con Haskell

a) Elabore un tipo de datos (usando `data`) que represente un árbol binario, donde cada nodo interno (rama) tiene un valor `Int` y cada nodo hoja no tiene asociado un valor.

b) Defina el árbol binario infinito `ones`; considerando que no tiene nodos hojas, pero si una cantidad infinita de nodos internos. Cada nodo interno debe tener el valor de 1.

c) Defina la función `rDepth` que tome un árbol infito como su entrada y retorne un árbol donde cada valor de cada nodo interno debe ser igual al de su profundidad. *Nota: Considera que la raíz tiene profundidad 0*

d) Defina la función `take`, tal que toma dos entradas: la primera es un entero `d` y la segunda es un árbol `a`. Esta función debe retornar un árbol que es el mismo que `a` con la diferencia de que los nodos con profundidad mayor a `d` son removidos (solo incluye nodos con profundidad hasta `d`).

```
Main> take 2 (rDepth ones)
        0
    1       1
  2   2   2   2
.   .   .   .   .
```

Nota: no se preocupe de la representación en pantalla de un arból.