Lenguajes de Programación
Auxiliar 2

Auxiliares: Tomás Vallejos, Bryan Ortiz

11 de Septiembre 2020

- How to `test` y `test/exn`

```
(test (funcion-bacan args) resultado)
(test/exn (funcion-bacan args) "error: too nice for this
world")
```

- Podemos definir nuevos tipos con `deftype`:

```
(deftype tipo-a-crear
  (constructor1 arg1 ... argn)
  ...)
```

- Y analizar estructuras con `match` (Pattern Matching):

```
(match val
  [patron resultado]
  ...)
```

Patrones Importantes:	**Matchea con:**
`id`	Lo que sea
`'simbolo`	Exactamente ese simbolo
`(constructor arg1 .. argn)`	Constructor con sus argumentos
`(list arg1 .. argn)`	Lista con n elementos
`(? pred patrón)`	Los casos en que `(pred val)` es `#t`

- Otra forma de hacer pattern matching:

```
(def (cons a b) '(1 2))
```

Suficiente resumen, ahora vamos a los ejercicios! 

1 Ejercicios

Pueden usar el Help Desk en caso de dudas (menú Help en DrRacket).
Recuerden enunciar el contrato y escribir tests ;).

1. Defina las funciones:
   (a) `my-map`, una versión casera de `map`.
   (b) `my-filter`, lo mismo para `filter`.
   (c) `my-foldr`, un `foldr` echo a manito.
2. Usando `map`, `foldl`, `foldr`, o `filter` según corresponda, defina una función que cumpla lo pedido para cada caso:
   (a) Dada una lista de números, que retorne la misma lista con cada elemento en formato string.
   (b) Dada una lista de números, que retorne la sumatoria de todos su elementos.
   (c) Dada una lista de strings, que retorne la concatenación de todos sus elementos.
   (d) Dada una lista de lo que sea, que retorne la lista de todos los elementos numéricos.
   (e) Dada una lista de números y una función, retorne la sumatoria de los elementos de la lista tras haber aplicado la función
3. En clases se vió lo que es la currificación, que en simple es transformar la forma en que una función recibe sus argumentos, por ejemplo `(f arg1 arg2)` al ser currificada, se aplica como `((f arg1) arg2)`.
   (a) Defina la función `(curry a f)` que al pasarle una función de `a` argumentos, devuelve su versión currificada.
   (b) Defina la función `(uncurry-2 f)` que al pasarle una función currificada de 2 argumentos, devuelve su versión que recibe 2 argumentos.
4. Implemente la función `(sumatoria a f b)` que computa la suma
$$\sum_{i=a}^{b} f(i)$$
.
5. Un árbol binario se puede describir con la BNF sigiente:
   ⟨*AB*⟩ `::=` `(` `hoja` ⟨*val*⟩ `)`
   `|` `(` `nodo` ⟨*bt*⟩ ⟨*bt*⟩ `)`
   (Pueden pensarlo como una GLC).
   (a) Defina el tipo `AB` de los árboles binarios utilizando `deftype`.
   (b) Al crear un árbol con `deftype`, se define automágicamente la función `AB?`. Defina la función `map-ab` sobre árboles binarios, que lance un error si su argumento no es un árbol.

2 Propuestos

1. Usando `map`, `foldr`, o `filter` según corresponda, defina una función que cumpla lo pedido para cada caso:

   (a) Dada una lista de strings, retorne una lista con el largo de cada elemento.

   (b) Dada una lista de números, retorne la multiplicación de todos aquellos mayores a 5.

   (c) Dada una lista de strings que representan números, devuelva la suma de aquellos que son reales, por ejemplo:

   ```
   (f '( "1" "1+2i" "5.3" "-6" "8+3i"))
   ```

   debe entregar como resultado 0.3.

2. Implemente su propia versión `my-foldl`.

3. Implemente la función `map` usando `foldl`

4. Defina la función `filter-ab` para los árboles binarios vistos en la clase auxiliar