

Aux 1 – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter
Auxiliar: Oscar E. A. Callaú

4 de Agosto del 2008

1. Polinomios

Se presenta un problema sencillo sobre la manipulación de polinomios y se plantea una etapa de pasos (en base a la técnica *Como Diseñar Programas* presentado en clases por el profesor Tanter) para solucionar el problema de forma óptima en base a su estructuración y su semántica.

1.1. Problema

Dada la representación de un polinomio de grado n

$$p(x) = \sum_{i=0}^n c_i \times x^i \quad \wedge c_n \neq 0$$

se quiere encontrar su derivada.

Ej. $p_1(x) = 2 + 3x - 4x^3$ se tiene que $p'_1(x) = 3 - 12x^2$ y $p''_1(x) = -24x$

1.2. Diseño de Datos

Se propone la siguiente representación `<poly>` en BNF de polinomios en base a su definición.

```
<poly> ::= <list>
<list> ::= <coef> <list>
          | <empty>
<coef> ::= <number>
```

Que indica que un polinomio esta representado por una lista de coeficientes del tipo entero y la posición del coeficiente representa la potencia de la base por la cual se multiplica, la primera posición de la lista corresponde al 0 y la longitud de la lista menos uno es el grado del polinomio (por simplicidad suponemos que nunca vamos a manejar una lista vacía como representación de un polinomio).

Ej. $p_1(x) = (2\ 3\ 0\ -4)$, $p'_1(x) = (3\ 0\ -12)$, $p''_1(x) = (0\ -24)$

1.3. Operaciones

La operación principal (o función) vamos a definirla según la solicitud del problema, esto quiere decir plantear la función `derivar` que dado un polinomio en representación `<poly>`, con contrato:

$$\text{derivar} :: \langle \text{poly} \rangle \rightarrow \langle \text{poly} \rangle$$

Como en este proceso de derivación necesitamos saber la posición de cada coeficiente, necesitamos mandar en la recursión un parámetro extra como contador, esto modificaría nuestro contrato previo, agregando una mayor complejidad al uso de nuestra función, como solución a este problema definimos una función `multPoly`, con contrato:

$$\text{multPoly} :: \langle \text{poly} \rangle \times \langle \text{number} \rangle \rightarrow \langle \text{poly} \rangle$$

que además de manejar el contador, esta función tiene el objetivo de ir calculando el valor de cada nuevo coeficiente en el polinomio derivado. Haciendo así que nuestra función `derivar` se definia como una función del tipo `wrapper`. Sus esqueletos tendrían la forma:

```
(define (derivar poly)
  (.. multPoly ..))

(define (multPoly poly n)
  (if (null? poly) ..
      (.. multPoly ..)))
```

Los tests correspondientes serán entonces:

```
(test (derivar '(2 3 0 -4)) '(3 0 -12))
(test (derivar ((derivar '(2 3 0 -4)))) '(0 -24))
```

1.4. Implementación

Ahora que ya conocemos todos los detalles y condiciones de nuestras operaciones terminamos su implementación.

```
;; derivar :: <poly> -> <poly>
(define (derivar poly)
  (cdr (multPoly poly 0)))

;; multPoly :: <poly> x <num> -> <poly>
(define (multPoly poly exp)
  (if (null? poly) '()
      (let ((coef (car poly)))
        (cons (* coef exp) (multPoly (cdr poly) (+ 1 exp))))))
```

1.5. Testeo

Ejecutamos los test que se presentaron en el punto anterior para verificar nuestra implementación, obteniendo los siguientes resultados:

```
(good (derivar '(2 3 0 -4)) '(3 0 -12) )
(good (derivar (derivar '(2 3 0 -4))) '(0 -24) )
```

indicando que nuestra implementación a funcionado correctamente.

1.6. Propuesto

Agregue una función para evaluar $p(x)$ con un x dado.

2. Más ejercicios

1. Escriba una función que reciba dos números enteros y retorne una lista formada por todos los enteros entre ellos.

```
(n-list 0 10)
-> (0 1 2 3 4 5 6 7 8 9 10)
(n-list -1 1)
-> (-1 0 1)
```

2. Defina la función `-reverse` que invierte los elementos en una lista.
3. Defina la función `is-member?` que toma como argumentos un símbolo y una lista, retorna true o false dependiendo si el símbolo estaba o no en la lista.
4. Redefina `is-member?` de modo que siga respondiendo correctamente en los siguientes casos:

```
(is-member? 2 (0 1 (2) 3))
-> true
(is-member? 3 (0 (1 2 (3))))
-> true
```