Capítulo 3

Abstracción de Datos y Números

El cómputo numérico ha sido tradicionalmente ignorado por la comunidad de Lisp. Hasta antes del Common Lisp nadie había ideado una estrategia detallada para ordenar el cómputo numérico y, con la excepción del sistema MacLisp, habían muy pocos esfuerzos por ejecutar eficientemente código numérico. El reporte del estándar de Scheme reconoce el trabajo del comité de Common Lisp e incorpora muchas de sus propuestas [6].

3.1 Tipos Numéricos

Matemáticamente los números pueden organizarse en una torre de subtipos en la que cada nivel es un subconjunto del nivel que se encuentra sobre él:

número
complejo
real
racional
entero

La regla uniforme de coerción hacia abajo es que un número de un cierto tipo es también de un tipo inferior si la "inyección" (coerción hacia arriba) o la "proyección" (coerción hacia abajo) del número lo deja sin cambios. Por ejemplo 10 es un entero.

Por lo tanto, 10 también es un número racional, un real y un complejo. Puesto que la torre es una estructura matemática, Scheme proporciona predicados y procedimientos para acceder a ella.

El estándar de Scheme no requiere que todas las implementaciones de este lenguaje proporcionen la torre completa, sino que es suficiente con un subconjunto coherente que sea consistente tanto con los propósitos de la implementación como con el espíritu del lenguaje. Sin embargo aunque el intérprete no proporcione todos estos tipos, normalmente se incluyen predicados que permiten identificarlos. De tal forma, **number?**, **complex?**, **real?**, **rational** e **integer** se incluyen en la mayor parte de las implementaciones de Scheme.

3.2 Operadores Numéricos

Los operadores de la suma y la multiplicación son + y *, respectivamente. Estos operadores pueden tomar cualquier número de argumentos (se retorna el elemento identidad en caso de que no se proporcionen argumentos). Considere los siguiente ejemplos:

> (+ 3 6 7) $\Longrightarrow$ **16**
> (* 2 9 4) $\Longrightarrow$ **72**
> (+ 3 6) $\Longrightarrow$ **9**
> (* 4 5) $\Longrightarrow$ **20**
> (+ 6) $\Longrightarrow$ **6**
> (* 8) $\Longrightarrow$ **8**
> (+) $\Longrightarrow$ **0**
> (*) $\Longrightarrow$ **1**

Los operadores de la resta y la división son - y /, respectivamente. Estos operadores pueden tomar uno, dos o más argumentos dependiendo de la versión de Scheme de que se trate. Cuando se usa un solo argumento, estos operadores retornan el inverso aditivo o multiplicativo de su argumento. Considere los siguientes ejemplos:

> (- 9 6) $\Longrightarrow$ **3**
> (/ 7 3) $\Longrightarrow$ **7/3**
> (- 7) $\Longrightarrow$ $-$**7**
> (/ 5) $\Longrightarrow$ **1/5**

Para comparar números, se proporcionan los siguientes procedimienros transitivos:

$(= z_1\ z_2\ z_3\ \ldots)$
$(< x_1\ x_2\ x_3\ \ldots)$
$(> x_1\ x_2\ x_3\ \ldots)$
$(<= x_1\ x_2\ x_3\ \ldots)$
$(>= x_1\ x_2\ x_3\ \ldots)$

También pueden verificarse algunas propiedades de los números:

(**zero?** z)
(**positive?** x)
(**negative?** x)
(**odd?** n)
(**even?** n)

También podemos obtener los valores máximo y mínimo de un conjunto dado de números:

(**max** $x_1\ x_2\ \ldots$)
(**min** $x_1\ x_2\ \ldots$)

El valor absoluto (o la magnitud) de un número se calcula usando **abs:**

(**abs** -13) $\Longrightarrow$ 13

Para los números positivos n_1 y n_2, si n_3 y n_4 son enteros tales que $n_1 = n_2 n_3 + n_4$ y $0 \leq n_4 < n_2$, se proporcionan los siguientes procedimientos:

(**quotient** $n_1\ n_2$) $\Longrightarrow n_3$
(**remainder** $n_1\ n_2$) $\Longrightarrow n_4$
(**modulo** $n_1\ n_2$) $\Longrightarrow n_4$

El valor que regresa **quotient** siempre tiene el signo del producto de sus argumentos. Los procedimientos **remainder** y **modulo** difieren cuando se les proporciona argumentos negativos. **Remainder** retorna el cero o el signo del dividendo, mientras que el **modulo** regresa siempre un valor con el mismo signo del divisor. Considere los siguientes ejemplos:

(**quotient** 10 4) $\Longrightarrow$ **2**
(**quotient** 10 -4) $\Longrightarrow$ **−2**
(**remainder** 10 -4) $\Longrightarrow$ **2**
(**modulo** 10 -4) $\Longrightarrow$ **−2**
(**remainder** 10 4) $\Longrightarrow$ **2**
(**modulo** 10 4) $\Longrightarrow$ **2**

Los procedimientos **gcd** y **lcm** regresan el máximo común divisor y el mínimo común múltiplo de sus argumentos, respectivamente. El resultado es siempre un valor no negativo. Si no se proporcionan argumentos, se retornan cero y uno, respectivamente. Considere los siguientes ejemplos:

> (**gcd** 68 24 12 48) $\Longrightarrow$ **4**
> (**gcd**) $\Longrightarrow$ **0**
> (**lcm** 68 24 12 48) $\Longrightarrow$ **816**
> (**lcm** 68 -24 12 -48) $\Longrightarrow$ **4**
> (**lcm** 15 25 -20 40) $\Longrightarrow$ **600**
> (**lcm**) $\Longrightarrow$ **1**

Los procedimientos **numerator** y **denominator** son auto–explicables y calculan sus resultados suponiendo que el argumento que se proporciona está representado como una fracción en términos del menor orden posible. Otra presuposición de estos procedimientos es que el denominador es siempre positivo y que el denominador de cero es uno. Considere los siguientes ejemplos:

> (**numerator** (/ 15 12)) $\Longrightarrow$ **5**
> (**numerator** (/ 3 4)) $\Longrightarrow$ **3**
> (**numerator** (/ 4)) $\Longrightarrow$ **1**
> (**denominator** (/ 15 12)) $\Longrightarrow$ **4**
> (**denominator** (/ 3 4)) $\Longrightarrow$ **4**
> (**denominator** (/ 4)) $\Longrightarrow$ **4**

Otros procedimientos útiles que regresan enteros son:

(**floor** x)
(**ceiling** x)
(**truncate** x)
(**round** x)

Debe mencionarse que argumentos que se pasan a estos procedimientos son inexactos, entonces la respuesta será un entero inexacto. El procedimiento **floor** regresa el entero más grande que no exceda x. El procedimiento **ceiling** regresa el entero más pequeño que no sea inferior que x. El procedimiento **truncate** regresa el entero más cercano a x cuyo valor absoluto no sea mayor que el valor absoluto de x. El procedimiento **round** regresa el entero más cercano a x, redondeado a un valor par cuando x se encuentre exactamente en el punto medio entre dos valores enteros. Considere los siguientes ejemplos:

$>$ (**floor** 3.8) $\Longrightarrow$ **3**.
$>$ (**floor** 3.5) $\Longrightarrow$ **3**.
$>$ (**floor** 3.1) $\Longrightarrow$ **3**.
$>$ (**floor** -3.8) $\Longrightarrow$ $-$**4**.
$>$ (**floor** -3.1) $\Longrightarrow$ $-$**4**.
$>$ (**ceiling** 3.2) $\Longrightarrow$ **4**.
$>$ (**ceiling** -3.2) $\Longrightarrow$ $-$**3**.
$>$ (**ceiling** 3.8) $\Longrightarrow$ **4**.
$>$ (**ceiling** -3.8) $\Longrightarrow$ $-$**3**.
$>$ (**truncate** 3.8) $\Longrightarrow$ **3**.
$>$ (**truncate** 3.1) $\Longrightarrow$ **3**.
$>$ (**truncate** 3.5) $\Longrightarrow$ **3**.
$>$ (**truncate** -3.8) $\Longrightarrow$ $-$**3**.
$>$ (**truncate** -3.5) $\Longrightarrow$ $-$**3**.
$>$ (**truncate** -3.1) $\Longrightarrow$ $-$**3**.
$>$ (**round** 3.8) $\Longrightarrow$ **4**.
$>$ (**round** 3.5) $\Longrightarrow$ **4**.
$>$ (**round** 3.2) $\Longrightarrow$ **3**.
$>$ (**round** -3.8) $\Longrightarrow$ $-$**4**.
$>$ (**round** -3.5) $\Longrightarrow$ $-$**4**.
$>$ (**round** -3.2) $\Longrightarrow$ $-$**3**.

El procedimiento **rationalize** regresa el número racional *más simple* diferente de x pero no más que y:

(**rationalize** x y)

Un número racional r_1 es *más simple* que otro número racional r_2 si $r_1 = p_1/q_1$ y $r_2 = p_2/q_2$ y $|p_1| \leq |p_2|$ y $|q_1| \leq |q_2|$. De tal forma 1/2 es más

simple que 3/4. Aunque no todos los números racionales son comparables de acuerdo a este ordenamiento (considere por ejemplo 2/3 y 3/4) cualquier intervalo contiene un número racional que es más simple que otro número racional en ese intervalo. Advierta que 0=0/1 es el racional más simple de todos. Considere los siguientes ejemplos:

> (**rationalize** (/ 1 2) (/ 1 20)) $\Longrightarrow$ **1/2**
> (**rationalize** (/ 1 2) 1) $\Longrightarrow$ **0**

El operador raíz cuadrada (implementado como procedimiento) es:

(**sqrt** z)

Este procedimiento regresa la raíz cuadrada de z. El resultado tendrá una parte positiva real o cero y una parte imaginaria no negativa.

Finalmente el procedimiento:

(**expt** z_1 z_2)

regresa z_1 elevado a la potencia z_2: $z_1^{z_2} = e^{z_2 \log z_1}$. Recuerde que 0^0 se define como igual a 1.

La mayor parte de las versione de Scheme cuentan también con los siguientes procedimientos matemáticos adicionales:

(**exp** z)
(**log** z)
(**sin** z)
(**cos** z)
(**tan** z)
(**asin** z)
(**acos** z)
(**atan** z)

3.2.1 Problemas Resueltos

1. Defina un procedimiento **sum** que encuentre la suma de los componentes de una n-tupla. Pruebe su procedimiento con:

(sum '(1 2 3 4 5)) $\Longrightarrow$ **15**
(sum '(6)) $\Longrightarrow$ **6**
(sum '()) $\Longrightarrow$ **0**

El procedimiento es:

```
(define sum
(lambda (ls)
  (cond
((null? ls) 0)
  (else (+ (car ls) (sum (cdr ls)))))))
```

2. Defina un procedimiento **suma-parejas** que tome dos n-tuplas de la misma longitud, **ntpl-1** y **ntpl-2**, como argumentos y produzca una nueva n-tupla cuyos componentes sean la suma de los valores correspondientes de **ntpl-1** y **ntpl-2**. Pruebe su procedimiento con:

   (suma-parejas '(1 3 2) '(4 -1 2)) $\Longrightarrow$ **(5 2 4)**
   (suma-parejas '(3.2 1.5)) $\Longrightarrow$ **(9.2 -1.0)**
   (suma-parejas '(7) '(11)) $\Longrightarrow$ **(18)**
   (suma-parejas '() '()) $\Longrightarrow$ **()**

   El procedimiento es:

```
(define suma-parejas
(lambda (ls1 ls2)
  (cond
((null? ls1) '())
  (else (cons (+ (car ls1) (car ls2))
(suma-parejas (cdr ls1) (cdr ls2)))))))
```

3. Defina un procedimiento **mult-por-n** que tome como argumentos un número **num** y una n-tupla **ntpl**. Este procedimiento multiplicará cada componente de **ntpl** por **num**. Pruebe su procedimiento:

   (mult-por-n 3 '(1 2 3 4 5)) $\Longrightarrow$ **(3 6 9 12 15)**
   (mult-por-n 0 '(1 3 5 7 9 11)) $\Longrightarrow$ **(0 0 0 0 0 0)**
   (mult-por-n -7 '()) $\Longrightarrow$ **()**

El procedimiento es:

```
(define mult-por-n
(lambda (n ls)
  (cond
((null? ls) '())
  (else (cons (* (car ls) n)
(mult-por-n n (cdr ls)))))))
```

4. Defina un procedimiento **cuenta-diferentes** que tome como argumentos un elemento **a** y una lista de elementos **ls**, y retorne el número de elementos en **ls** que no son iguales al elemento **a**. Pruebe su procedimiento con:

   (cuenta-diferentes 'blue '(red white blue yellow blue red)) $\Longrightarrow$ **4**
   (cuenta-diferentes 'red '(white blue green)) $\Longrightarrow$ **3**
   (cuenta-diferentes 'white '()) $\Longrightarrow$ **0**

   El procedimiento es:

```
(define cuenta-diferentes
(lambda (a ls)
  (cond
((null? ls) 0)
((not (equal? a (car ls)))
     (add1 (cuenta-diferentes a (cdr ls))))
  (else (cuenta-diferentes a (cdr ls))))))
```

5. Defina un procedimiento **envuelve** que tome como argumentos un elemento **a** y un entero positivo **num**. Este procedimiento debe colocar **num** cantidad de paréntesis alrededor del elemento **a**. Pruebe su procedimiento con:

   (envuelve 'gift 1) $\Longrightarrow$ **(gift)**
   (envuelve 'sandwich 2) $\Longrightarrow$ **((sandwich))**
   (envuelve 'prisoner 5) $\Longrightarrow$ **(((((prisoner)))))**
   (envuelve 'moon 0) $\Longrightarrow$ **moon**

El procedimiento es:

(**define** envuelve
(**lambda** (a n)
  (**cond**
((**zero?** n) a)
(**else** (**cons** (envuelve a (sub1 n)) '()))))))

Los siguientes procedimientos adicionales serán de utilidad para resolver los ejercicios de este capítulo:

(**define** sub1
(**lambda** (n)
  (- n 1)))

(**define** add1
(**lambda** (n)
  (+ n 1)))

(**define** error
(**lambda** args
  (**display** "Error:)
(**for-each** (**lambda** (value) (**display** " ")
  (**display** value)) args)
(**newline**)))

3.3 Ejercicios Propuestos

1. De su curso de cálculo, sabe que

$$\lim_{n\to\infty}\left(1+1+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}\right)=e \tag{3.1}$$

Defina un procedimiento **e-series** que tome como entrada un entero n, y regrese la suma de los primeros n terminos de esta serie infinita. Esto deberá darnos una aproximación de e. Evalúe su procedimiento con:

(e-series 10) $\Longrightarrow$ 9864101/3628800
(e-series 20) $\Longrightarrow$ 6613313319248080001/2432902008176640000

NOTA: No intente probar este procedimiento con números que sean muy grandes, puesto que el intérprete de Scheme tomará un tiempo muy grande para evaluar las funciones correspondientes.

2. Defina un procedimiento **producto-punto** que tome dos n-tuplas de la misma longitud, multiplique sus componentes respectivos y sume los productos resultantes. Pruebe su procedimiento con:

   (**producto-punto** '(3 4 -1) '(1 -2 -3)) $\Longrightarrow$ -2
   (**producto-punto** '(0.003 0.035) '(8 2)) $\Longrightarrow$ 0.094
   (**producto-punto** '(5.3e4) '(2.0e-3)) $\Longrightarrow$ 106.0
   (**producto-punto** '() '()) $\Longrightarrow$ 0

3. Defina un procedimiento **indice** que tome dos argumentos, un elemento **a** y una lista de elementos **ls**. Este procedimiento debe retornar el índice de **a** en **ls**. Es decir, debe retornar la posición (a partir de la cero) de **a** en **ls**. Si el elemento no se encuentra en la lista, el procedimiento debe retornar -1. Pruebe su procedimiento con:

   (**indice** 3 '(1 2 3 4 5 6)) $\Longrightarrow$ 2
   (**indice** 'so '(do re me fa so la ti do)) $\Longrightarrow$ 4
   (**indice** 'a '(b c d e)) $\Longrightarrow$ -1
   (**indice** 'gato '()) $\Longrightarrow$ -1

4. Defina un procedimiento **frente-lista** que toma como argumentos una lista de elementos **ls** y un entero positivo **num** y retorne los primeros **num** elementos de alto nivel de **ls**. Si **num** es mayor que la cantidad de elementos de alto nivel de **ls**, debe generarse un mensaje de error. Pruebe su procedimiento con:

   (**frente-lista** '(a b c d e f g) 4) $\Longrightarrow$ (a b c d)
   (**frente-lista** '(a b c) 4)

$\Longrightarrow$ Error: la longitud de (a b c) es menor que 4.
(**frente-lista** '(a b c d e f g) 0) $\Longrightarrow$ ()
(**frente-lista** '() 3) $\Longrightarrow$ Error: la longitud de () es menor que 3.

5. Defina un procedimiento **multiplo?** que tome como argumentos dos enteros **m** y **n** y retorne **#t** si **m** es un múltiplo exacto de **n**. Pruebe su procedimiento con:

   (**multiplo?** 7 2) $\Longrightarrow$ #f
   (**multiplo?** 9 3) $\Longrightarrow$ #t
   (**multiplo?** 5 0) $\Longrightarrow$ #f
   (**multiplo?** 0 20) $\Longrightarrow$ #t
   (**multiplo?** 17 1) $\Longrightarrow$ #t
   (**multiplo?** 0 0) $\Longrightarrow$ #t

6. Defina un procedimiento **n-tupla->entero** que convierta una n-tupla no vacía de dígitos a un número. Pruebe su procedimiento con:

   (**n-tupla->entero** '(3 1 4 6)) $\Longrightarrow$ 3146
   (**n-tupla->entero** '(0)) $\Longrightarrow$ 0
   (**n-tupla->entero** '()) $\Longrightarrow$ Error: el argumento () no es valido.
   (+ (**n-tupla->entero** '(1 2 3)) (**n-tupla->entero** '(3 2 1))) $\Longrightarrow$ 444