Introducción a los Algoritmos - 2do. cuatrimestre 2012

Guía 1: Funciones, precedencia y tipado

Docentes: Walter Alini, Luciana Benotti, Marcos Gómez y Gisela Rossi

El objetivo de los siguientes ejercicios es introducirnos en la ***programación funcional****, es decir, al desarrollo de programas como funciones.*

La guía incluye también ejercicios de precedencia y tipado, para que puedas comprobar si tus programas están bien escritos. También aprenderás a usar un interprete de `haskell` *, y a desarrollar tus primeros programas.*

Funciones

1. **Para realizar en grupo:** Leé la información sobre *algoritmo* que se encuentra en `Wikipedia` para responder a las siguientes preguntas (no todas las respuestas están en `Wikipedia`, algunas de las preguntas son para usar tu conocimiento previo y consultar en clase si están bien).

   a) ¿Qué es un algoritmo?

   b) ¿Cuál es la relación entre un algoritmo y una función?

   c) ¿Qué ejemplos de función conocés?

   d) ¿Qué son el *dominio* y *codominio* de una función?

   e) ¿Para qué servirán las funciones en programación?

   f) ¿Qué es la aplicación/evaluación de una función?

2. En las siguientes definiciones identificá las variables, las constantes y el nombre de la función

   a) `f x = 5 * x`

   b) `duplica a = a + a`

   c) `por2 y = 2 * y`

   d) `multiplicar zz tt = zz * tt`

3. Escribí una función que dados dos valores, calcule su promedio. Indicá el tipo de esa función.

4. Tomando las definiciones del punto 2 evaluá las siguientes expresiones. Justificá cada paso utilizando la notación aprendida. Luego, controlá los resultados en `Haskell`.

   a) `(multiplicar (f 5) 2) + 1`

   b) `por2 (duplica (3 + 5))`

5. Definí las funciones que describimos a continuación, luego implementalas en `haskell` . Por ejemplo:

   Enunciado: $signo : Int \to Int$, que dado un entero retorna su signo, de la siguiente forma: retorna 1 si x es positivo, -1 si es negativo y 0 en cualquier otro caso.

   Solución:

   ```
   sgn :: Int -> Int
   sgn x | 0<x   = 1
         | x<0   = -1
         | x==0  = 0
   ```

   En `haskell` los conectivos para las condiciones se escriben así:

$\wedge$	`&&`
$\vee$	`||`
$\neg$	`not`

   a) $entre0y9 : Int \to Bool$, que dado un entero devuelve *True* si el entero se encuentra entre 0 y 9.

   b) $segundo3 : (Int, Int, Int) \to Int$, que dada una terna de enteros devuelve su segundo elemento.

   c) $ordena : (Int, Int) \to (Int, Int)$, que dados dos enteros los ordena de menor a mayor.

d) *rangoPrecio* : $Int \to String$, que dado un número que representa el precio de una computadora, retorne "muy barato" si el precio es menor a 2000, "demasiado caro" si el precio es mayor que 5000, "hay que verlo bien" si el precio está entre 2000 y 5000, y "esto no puede ser!" si el precio es negativo.

e) *absoluto* : $Int \to Int$, que dado un entero retorne su valor absoluto.

f) *esMultiplo2* : $Int \to Bool$, que dado un entero n devuelve *True* si n es múltiplo de 2.
Ayuda: usar `mod`, el operador que devuelve el resto de la división.

g) *rangoPrecioParametrizado* : $Int \to (Int, Int) \to String$ que dado un número x, que representa el precio de un producto, y un par $(menor, mayor)$ que represente el rango de precios que uno espera encontrar, retorne "muy barato" si x está por debajo del rango, "demasiado caro" si está por arriba del rango, "hay que verlo bien" si el precio está en el rango, y "esto no puede ser!" si x es negativo.

h) *mayor3* : $(Int, Int, Int) \to (Bool, Bool, Bool)$, que dada una una terna de enteros devuelve una terna de valores booleanos que indica si cada uno de los enteros es mayor que 3.
Por ejemplo: $mayor3\ (1, 4, 3) = (False, True, False)$ y $mayor3\ (5, 1984, 6) = (True, True, True)$

Precedencia y tipado

La ***precedencia*** *de los operadores nos permite escribir fórmulas grandes de manera simple y más legible. Cuando un operador tiene mayor precedencia que otro, podemos escribir una fórmula que involucra a ambos sin necesidad de poner paréntesis, y a pesar de esto, la fórmula tiene un sentido único. Un ejemplo conocido por todos es el caso de la suma respecto a la multiplicación. El operador $*$ tiene mayor precedencia que $+$ y por ello es que normalmente interpretamos la expresión $2 + 4 * 3$ como $2 + (4 * 3)$. Esta regla es la que nos permitía en la primaria "separar en términos" una expresión algebraica. Al mismo tiempo, y por la misma regla, sabemos que no es lo mismo la expresión $(2 + 4) * 3$ que $2 + 4 * 3$. No siempre los paréntesis pueden sacarse indiscriminadamente sin cambiar el sentido de la expresión.*

De la misma manera, cuando un operador ***asociativo*** *se aplica múltiples veces es posible eliminar paréntesis, ya que el orden en que se efectúa la operación no altera el resultado. Por ejemplo $(2+4)+7$ es igual a $2+(4+7)$ y por lo tanto podemos escribir simplemente $2 + 4 + 7$.*

Los operadores $+$, $*$, mín, máx son asociativos.

tabla de precedencia	*más arriba → mayor precedencia*
$\sqrt{}, (\cdot)^2$	*raíces y potencias*
$*, /$	*producto y división*
máx, mín	*máximo y mínimo*
$+, -$	*suma y resta*
$=, <, \leqslant, >\geqslant$	*operadores de comparación*

Por otro lado, la noción de ***tipado*** *nos permite distinguir expresiones que están "bien escritas", es decir que tienen sentido, que representan algún valor. El tipo de un operador o función nos dice cuál es la clase de valores que toma y cuál es la clase del valor que devuelve. En las expresiones algebraicas en general todos los valores son de tipo número, que denotamos con Num (y los* ***subtipos*** *Nat, Int, ...). Como ejemplo tomemos el caso de la suma: tanto a la izquierda del símbolo "+" como a la derecha debe haber expresiones que representen números, y el resultado total también es un número.*

Además del tipo Num utilizaremos el tipo de los valores booleanos, que denotamos con Bool. Este tipo incluye las constantes True y False que representan las nociones de verdadero y falso respectivamente. Los valores booleanos son importantes porque son el resultado de los operadores de comparación $=, <, \leqslant, >, \geqslant$.

Las nociones de precedencia y tipado son importantes para asegurar que escribimos expresiones que tienen un sentido único y bien definido. Los siguientes ejercicios tratan sobre estas nociones en el marco de las expresiones algebraicas.

6. Sacá todos los paréntesis que sean *superfluos* según las reglas de precedencia y asociatividad de los operadores aritméticos. Por ejemplo:

$$
\begin{aligned}
& (8-6) * x = \underline{(6 * (x^2))} + 3 \\
\equiv\ & \{ \text{ precedencia de } * \text{ por sobre } + \} \\
& (8-6) * x = 6 * \underline{(x^2)} + 3 \\
\equiv\ & \{ \text{ precedencia de } {}^2 \text{ sobre } * \} \\
& (8-6) * x = 6 * x^2 + 3
\end{aligned}
$$

a) $(-(5 + x) + ((3 * 6)/(4 * 5))) * (8 * 5)$

b) $(((2)^2 + 5) - (4 * 2)/(4)) + 1 = ((5 * x)/8) + (3 * 5)^3$

7. Introducí paréntesis para hacer *explícita* la precedencia.

   a) $5 * 3 + 4 \geqslant 7 - 7 + 3$

   b) $3 + 4 * x = 4$

8. Escribí el *tipo* de los siguientes operadores y funciones: $*$, $/$, ^{2}, $\leqslant$, $\geqslant$, $=$. ¿Cuál es el tipo de esos mismos operadores en `haskell`? ¿Qué sucede con el operador $-$?

   Como ejemplo presentamos el caso del operador $+$. Este operador toma dos números y devuelve otro número. Podemos escribir su tipo en *notación funcional* listando el tipo de los parámetros y a continuación el tipo resultado:

   $$+ : Num \to Num \to Num$$

   Otra forma de escribir el tipo de este operador, útil para armar el árbol de tipado de una expresión, es en *notación de árbol*:

   $$\frac{Num + Num}{Num}$$

   En `haskell` el operador `(+)` tiene el mismo tipo. En `ghci` podemos corroborarlo de la siguiente manera:

```
Main> :t (+)
(+) :: Num a => a -> a -> a
```

   Esta respuesta se interpreta de la siguiente manera. El tipo del operador `(+)` es lo que está a la derecha de `=>`, es decir $a \to a \to a$, donde la variable a indica cualquier tipo. Por otro lado, lo que está a la izquierda de `=>` nos indica que el tipo a es un subtipo de *Num*, es decir, *Nat*, *Int*.

Un árbol de tipado es una herramienta que nos permite verificar que una expresión está bien escrita, es decir, que respeta la sintaxis del lenguaje. Consiste en ir armando los tipos de cada subexpresión de la manera en que se iría resolviendo, es decir teniendo en cuenta la precedencia de los distintos operadores.

9. Completar los siguientes árboles de tipado con los tipos que corresponden a cada operación.

   a)

   $$\frac{\dfrac{(\ x\ +\ (\underline{2}\ /\ \underline{y})) \geqslant (\ \underline{x}\ +\ \underline{1}\ )}{\dfrac{\underline{\quad}\quad \dfrac{\underline{\quad}}{/}}{+}} \qquad \dfrac{\underline{\quad} \quad \underline{\quad}}{+}}{\geqslant}$$

   b)

   $$(\underline{8}\ -\ \underline{6})\ *\ \underline{x}\ =\ (\underline{6}\ *\ \underline{x}^{\,\underline{2}})\ +\ \underline{3}$$

   $$\frac{\underline{\quad}}{-} \qquad \frac{\underline{\quad}}{^2}$$

   $$\frac{\underline{\quad}}{*} \qquad \frac{\underline{\quad}}{*}$$

   $$\frac{\underline{\quad}}{+}$$

   $$\frac{\underline{\quad}}{=}$$

10. ¿Están bien escritas las siguientes expresiones? Justificá a través de un árbol de tipado utilizando las reglas que escribiste en el ejercicio 8 y eligiendo el tipo para cada variable. Para evitar errores, hace explícita la precedencia introduciendo paréntesis, y construí una tabla del tipo de las variables. Por ejemplo, para la expresión $x + 2/y \geqslant x + 1$ podemos armar el árbol de tipado de la siguiente manera:

Primero escribimos el tipo de las cosas sobre las que estamos seguros: las constantes.

$$(x + (\underset{Num}{2} / y)) \geqslant (x + \underset{Num}{1})$$

Luego asignamos tipos a las variables, según las reglas de tipado de los operadores que operan sobre ellas. Luego escribimos el tipo del resultado de estos operadores.

$$(\underset{Num}{x} + (\underset{Num}{2} / \underset{Num}{y})) \geqslant (\underset{Num}{x} + \underset{Num}{1})$$

Var	Tipo
x	Num
y	Num

Sucesivamente completamos el resto de los operadores, hasta dar el tipo de la expresión completa

$$\dfrac{(\underset{Num}{x} + (\underset{Num}{2} / \underset{Num}{y})) \geqslant (\underset{Num}{x} + \underset{Num}{1})}{\underset{Num}{Num + Num} \geqslant}$$

$$\dfrac{}{Bool}$$

Var	Tipo
x	Num
y	Num

a) $x + 2 * 3 = 1$

b) $(x - 1) * (x - 2) = x + 6$

c) $y = x = 3$

d) $10 \geqslant x \geqslant 0$

Apéndice

Niveles de Precedencia

$E(x := a)$, .	sustitución y evaluación
$\sqrt{}, (\cdot)^2$	raíces y potencias
$*, /$	producto y división
máx, mín	máximo y mínimo
$+, -$	suma y resta
$=, \leqslant, \geqslant$	operadores de comparación
$\neg$	negación
$\vee \wedge$	disyunción y conjunción
$\Rightarrow \Leftarrow$	implicación y consecuencia
$\equiv \neq$	equivalencia y discrepancia

Tipos de los operadores

$-x$	$- : Num \to Num$
x^y	$(-)^{(-)} : Num \to Num \to Num$
$\sqrt[r]{x}$	$\sqrt{} : Num \to Num \to Num$
$x * y$	$* : Num \to Num \to Num$
x/y	$/ : Num \to Num \to Num$
x máx y	máx : $Num \to Num \to Num$
x mín y	mín : $Num \to Num \to Num$
$x + y$	$+ : Num \to Num \to Num$
$x - y$	$- : Num \to Num \to Num$
$x > y$	$>: Num \to Num \to Bool$
$x \geqslant y$	$\geqslant: Num \to Num \to Bool$
$x < y$	$<: Num \to Num \to Bool$
$x \leqslant y$	$\leqslant: Num \to Num \to Bool$
$x = y$	$=: Num \to Num \to Bool$
$\neg p$	$\neg : Bool \to Bool$
$p \wedge q$	$\wedge : Bool \to Bool \to Bool$
$p \vee q$	$\vee : Bool \to Bool \to Bool$
$head.xs$	$head : [A] \to A$
$tail.xs$	$tail : [A] \to [A]$
$x \triangleright xs$	$\triangleright : A \to [A] \to [A]$
$xs \triangleleft x$	$\triangleleft : [A] \to A \to [A]$
$xs \uparrow n$	$\uparrow: [A] \to Int \to [A]$
$xs \downarrow n$	$\downarrow: [A] \to Int \to [A]$
$xs +\!\!+ ys$	$+\!\!+ : [A] \to [A] \to [A]$
$\#xs$	$\# : [A] \to Int$
$xs.n$	$(-).(-) : [A] \to Int \to A$

Operadores en `haskell`

x^2	`x^2`
x^n	`x^n` con n entero
x^p	`x**p`
$\sqrt{x}$	`sqrt x`
$\sqrt[r]{x}$	`x**(1/r)`
x máx y	`x ‘max‘ y` o bien `max x y`
x mín y	`x ‘min‘ y` o bien `min x y`
$x \geqslant y$	`x >= y`
$x \leqslant y$	`x <= y`
$x = y$	`x == y`
$\neg p$	`not p`
$p \wedge q$	`p && q`
$p \vee q$	`p || q`
$head.xs$	`head xs`
$tail.xs$	`tail xs`
$x \triangleright xs$	`x:xs`
$xs \triangleleft x$	`xs ++ [x]`
$xs \uparrow n$	`take n xs`
$xs \downarrow n$	`drop n xs`
$xs +\!\!+ ys$	`xs ++ ys`
$\#xs$	`length xs`
$xs.n$	`xs !! n`