Currificación

Teoría de Tipos:

Es una disciplina que estudia la noción de tipo de datos (**conjuntos de datos**) desde una perspectiva matemática y propiedades genéricas de los tipos de datos.
Partiendo de un contexto (una realidad, un lenguaje de programación, etc.) permite definir tipos de datos básicos y otros tipos de datos que se pueden construir a partir de ellos.

Consideremos la función suma que dados dos enteros devuelve la suma de ambos.

¿Cuál es el tipo de esta función, es decir el conjunto de pertenencia?

Usualmente, la respuesta es:

suma: (Z x Z) → Z

Esto se interpreta como: *suma* es una función que recibe un par ordenado de números enteros y devuelve un número entero. Por lo tanto *suma* es una función de un único parámetro o argumento.

Las funciones con múltiples argumentos, pueden manejarse de otra forma, un poco más compleja quizás, para explotar la capacidad de retornar funciones como resultado o de tomar funciones como parámetros.

Se considera la siguiente definición del tipo de suma:

suma: Z → Z → Z

Esto se interpreta como: *suma* es una función que recibe un entero, seguido de otro entero y devuelve la suma de ambos. Esto se deduce de la notación utilizada para dar el tipo de la función Z → Z → Z.

Esta forma de denotar los tipos de las funciones se denomina currificada. El nombre proviene de la persona que popularizó su uso: Haskell Curry. La notación currificada no es simplemente una notación, agrega poder de expresividad a los tipos de las funciones, como veremos de ahora en más.
El símbolo → (que se lee "implica"), asocia hacia la derecha, esto significa que las dos expresiones siguientes son equivalentes:

suma: Z → Z → Z

suma: Z → (Z → Z)

Por lo tanto se puede decir también que suma es una función que recibe un entero y devuelve una función que a su vez, recibe un entero y devuelve un entero

Entonces, en la aplicación de `suma` a dos enteros:

- `(suma 7 9)` se interpreta como la aplicación de la función suma a 7, luego la aplicación de la función resultante a 9.
- El tipo de la expresión `(suma 7 9)` es Z, es decir `(suma 7 9)` : Z, y en particular su valor es 16.

Para denotar la aplicación de una función f a un parámetro x, no usamos la notación usual f(x), sino que lo denotamos como (f x). Dependiendo de la precedencia que se le quiera dar a la aplicación serán necesarios o no los paréntesis, esto lo veremos un poco más adelante.

<u>Generalizando la notación currificada a la aplicación de una función con más de un parámetro:</u>

En la sintaxis currificada no se utilizan n-uplas para pasarle parámetros a la función, sino que los mismos "se van pasando de a uno" a la función.
Sintaxis tradicional: `f (x1, x2, … , xn)`
Sintaxis currificada: `(f x1 x2 … xn)`

<u>Por ejemplo</u>:

En vez de escribir `doble (5)`, escribimos `(doble 5)` y en vez de escribir `suma(3,4)` escribimos `(suma 3 4)`. En este último caso, en vez de pasar los parámetros 3 y 4 en simultáneo, es como si primero le pasáramos el 3 y enseguida le pasáramos el 4.

A continuación se estudiará la aplicación de suma a un entero:

- (suma 7) se interpreta como la aplicación de la función suma a 7.
- Por lo tanto el tipo de la expresión (suma 7) es Z → Z, es decir el una función que recibe un entero y devuelve un entero.
- (suma 7) 4 es un entero y su valor es 13
- (suma 7) x = 7 + x
- (suma 21) también es una función de tipo Z → Z diferente a la anterior.

Para analizar:

En la siguiente expresión `f: A → B` decimos que f es una función que recibe un parámetro de tipo A y devuelve un elemento de tipo B. Es decir, f es de tipo `A → B` o es un elemento del conjunto `A → B`.

- ¿Cómo se interpreta la siguiente expresión: `A → B`?
- ¿Y `N → N → N`?

En la notación currificada, así como podemos escribir funciones que devuelvan funciones podemos escribir funciones que reciban funciones como parámetros.

Esto lo denotamos en el tipo de la función cambiando la precedencia de los implica:

Por ejemplo:

`f: (Z → Z) → Z`

En este caso f es un función que recibe una función de Z en Z como parámetro y devuelve un entero.

¿Cuál es el tipo de g?

g: `(Z → Z) → Z → Z`

¿Cuántos parámetros recibe? ¿Cuáles y en qué orden?

¿Cuál es el tipo de h?

h: `Z → (Z → Z) → Z → Z`

¿Cuántos parámetros recibe? ¿Cuáles y en qué orden?

En este contexto;

- x, y :: Z
- La aplicación de g a f: (g f)es de tipo Z, es un entero
- La aplicación de h a x a f: (h x f) es de tipo Z, es un entero
- La aplicación de h a y:(h y) es una función de tipo (Z → Z) → Z → Z y como se ve es una función que recibe una función de Z → Z, luego un entero y devuelve un entero.

Como se vio, el implica asocia hacia la derecha, por lo tanto al colocar paréntesis en los tipos que no acompañen la asociatividad del implica estoy cambiando la precedencia, es por esto que no es lo mismo (Z → Z) → Z → Z que Z → Z → Z → Z.

Describir ambos tipos y explicar las diferencias.

La aplicación de funciones, por el contrario del implica, asocia hacia la izquierda, por lo tanto las siguientes expresiones son equivalentes:

- `(h x f)`
- `((h x) f)`

Sean

```
f: (Z → Z) → Z
p: Z → Z
h: Z → (Z → Z) → Z → Z
x: Z
```

Veremos cuáles de las siguientes expresiones son correctas, es decir, tienen tipo, y cuáles no tienen tipo. Explicar las respuestas e indicar el tipo en el caso de las expresiones correctas:

`f x` - incorrecta
`f p` - tiene tipo
`f p x` - no tiene tipo
`f (p x)` - incorrecta
`h (f p)` - correcta
`h f p` - incorrecta
`h x f` - correcta
`h x p (f p)` - correcta
`h (f p) (h x p)` - correcta
`h (f p) h x p` - incorrecta
`h (f p) (h x p) (f p)` - correcta
`f (p x) (h x p x)` - incorrecta

Realizar el siguiente ejercicio:
Sean

```
f : N → N
g : N → N
h : (N → N) → N → N
t : ((N → N) → N → N) → N → N
p : N → (N → N) → N → N
x, y: N
```

Indicar si las siguientes expresiones tienen tipo, en caso afirmativo indicarlo, justificando en todo caso la respuesta.

1. h x y 2. h f 3. g (h f) 4. g (h f 3) 5. (p x) g	6. t h 7. t (h f) 8. h (p x g) 9. g (t h x) 10. t (p x) x	11. f (p 3 f 2) 2 12. h (t h) 13. f (p x g y) (h g) 14. p (g y) (h f) (t h y) 15. g (h g x)