Programación funcional II

Ricardo Pérez López

IES Doñana, curso 2019/2020

1. Abstracciones funcionales

1.1. Expresiones lambda

1.1. Expresiones lambda

- Las **expresiones lambda** (también llamadas **abstracciones lambda** o **funciones anónimas** en algunos lenguajes) son expresiones que capturan la idea abstracta de «**función**».
- Son la forma más simple y primitiva de describir funciones en un lenguaje funcional.
- Su sintaxis (simplificada) es:

```
<expr_lambda> ::= lambda [<lista_parámetros>] : <expresión>
<lista_parámetros> := <identificador> (, <identificador>)*
```

- Por ejemplo:

```
lambda x, y: x + y
```

Parámetros y cuerpos

- Los identificadores que aparecen entre la palabra clave `lambda` y el carácter de dos puntos (`:`) son los **parámetros** de la expresión lambda.
- La expresión que aparece tras los dos puntos (`:`) es el **cuerpo** de la expresión lambda.
- En el ejemplo anterior:

```
lambda x, y: x + y
```

  - Los parámetros son `x` e `y`.
  - El cuerpo es `x + y`.
  - Esta expresión lambda captura la idea general de sumar dos valores (que en principio pueden ser de cualquier tipo, siempre y cuando admitan el operador `+`).

Aplicación funcional

- De la misma manera que decíamos que podemos aplicar una función a unos argumentos, también podemos aplicar una expresión lambda a unos argumentos.
- Recordemos que *aplicar* una función a unos argumentos producía el valor que la función asocia a esos argumentos en el conjunto imagen.
- Por ejemplo, la aplicación de la función *max* sobre los argumentos 3 y 5 se denota como $max(3, 5)$ y eso denota el valor 5.
- Igualmente, la aplicación de una expresión lambda como

```
lambda x, y: x + y
```

sobre los argumentos `4` y `3` se representa así:

```
(lambda x, y: x + y)(4, 3)
```

- Si hacemos la siguiente definición:

```
suma = lambda x, y: x + y
```

a partir de ese momento podemos usar `suma` en lugar de su valor (la expresión lambda), por lo que podemos hacer:

```
suma(4, 3)
```

en lugar de

```
(lambda x, y: x + y)(4, 3)
```

- En nuestro modelo de sustitución, la evaluación de la aplicación de una expresión lambda consiste en sustituir, en el cuerpo de la expresión lambda, cada parámetro por el argumento correspondiente (por orden) y reducir la expresión resultante.
- A esta operación se la denomina **aplicación funcional** o **β-reducción**.
- Siguiendo con el ejemplo anterior:

```
(lambda x, y: x + y)(4, 3)
```

aplicamos en el cuerpo de la expresión lambda la sustitución de los parámetros `x` e `y` por los argumentos `4` y `3`, respectivamente, lo que da:

```
4 + 3
```

que simplificando (según las reglas del operador `+`) da `7`.

- Lo mismo podemos hacer si definimos previamente la expresión lambda ligándola a un identificador:

```
suma = lambda x, y: x + y
```

- Así, la aplicación de la expresión lambda resulta más fácil y clara de escribir:

```
suma(4, 3)
```

- En ambos casos, el resultado es el mismo (7).

Variables ligadas y libres

- Si un identificador aparece en el cuerpo de una expresión lambda...
  - ... y también aparece en la lista de parámetros de la expresión lambda, a ese identificador le llamamos **variable ligada** de la expresión lambda.
  - En caso contrario, le llamamos **variable libre** de la expresión lambda.
- En el ejemplo anterior:

```
lambda x, y: x + y
```

los dos identificadores que aparecen en el cuerpo (x e y) son variables ligadas, ya que ambos aparecen también en la lista de parámetros de la expresión lambda.

- En cambio, en la expresión lambda:

```
lambda x, y: x + y + z
```

x e y son variables ligadas z es libre.

- Para que una expresión lambda funcione, sus variables libres deben estar ligadas a algún valor en el entorno **en el momento de evaluar una aplicación** de la expresión lambda sobre unos argumentos.
- Por ejemplo:

```
prueba = lambda x, y: x + y + z
prueba(4, 3)
```

da error porque `z` no está definido (no está ligado a ningún valor en el entorno).

- En cambio:

```
1 prueba = lambda x, y: x + y + z
2 z = 9
3 prueba(4, 3)
```

sí funciona (y devuelve 16) porque en el momento de evaluar la expresión lambda (en la línea 3) el identificador `z` está ligado a un valor (9).

- Observar que no es necesario que las variables libres estén ligadas en el entorno cuando se *crea* la expresión lambda, sino cuando se *aplica*.

- Una expresión lambda cuyo cuerpo sólo contiene variables ligadas, es una expresión cuyo valor sólo va a depender de los argumentos que se usen cuando se aplique la expresión lambda.
- En cambio, el valor de una expresión lambda que contenga variables libres dependerá no sólo de los valores de sus argumentos, sino también de los valores a los que estén ligadas las variables libres al evaluar la expresión lambda.
- Por ejemplo, podemos escribir una expresión lambda que calcule la suma de tres números a partir de otra expresión lambda que calcule la suma de dos números:

```
lambda x, y, z: suma(x, y) + z
```

En este caso, hay un identificador (`suma`) que no aparece en la lista de parámetros de la expresión lambda (por lo que es una variable libre).

Por tanto, el valor de la expresión lambda anterior dependerá de lo que valga `suma` (de lo que haga, de lo que devuelva...).

Variables *sombreadas*

- ¿Qué ocurre cuando una expresión lambda contiene como parámetros nombres que ya están definidos (ligados) en el entorno?
- Por ejemplo:

```
1  x = 4
2  total = (lambda x: x * x)(3)  # Su valor es 9
```

- La x que aparece en la línea 1 es diferente a la que aparece en la lista de parámetros de la expresión lambda de la línea 2.
- En este caso, decimos que **el parámetro x *hace sombra*** al identificador x que, en el entorno, está ligado al valor 4.
- Por tanto, el identificador x que aparece en el cuerpo de la expresión lambda **hace referencia al parámetro x de la expresión lambda**, y **no** al identificador x que está fuera de la expresión lambda (y que aquí está ligado al valor 4).

- Que el parámetro haga sombra al identificador de fuera significa que no podemos acceder a ese identificador externo desde el cuerpo de la expresión lambda como si fuera una variable libre.
- Si necesitáramos acceder al valor de la `x` que está fuera de la expresión lambda, lo que podemos hacer es cambiar el nombre al parámetro `x`. Por ejemplo:

```
1  x = 4
2  total = (lambda w: w * x)(3)  # Su valor es 12
```

Así, tendremos en la expresión lambda una variable ligada (el parámetro `w`) y una variable libre (el identificador `x`).

Renombrado de parámetros

- Los parámetros se pueden *renombrar* (siempre que se haga de forma adecuada) sin que se altere el significado de la expresión lambda.
- A esta operación se la denomina **α-conversión**.
- Un ejemplo de α-conversión es la que hicimos antes.
- La α-conversión hay que hacerla correctamente para evitar efectos indeseados. Por ejemplo, en:

```
lambda x, y: x + y + z
```

si renombramos `x` a `z` tendríamos:

```
lambda z, y: z + y + z
```

lo que es claramente incorrecto. A este fenómeno indeseable se le denomina **captura de variables**.

1.2. Orden de evaluación

1.3. Funciones y procesos

1.4. Registros de activación

2. Composición de funciones

3. Tipos de datos compuestos

3.1 Cadenas

3.2 Listas

3.1. Cadenas

3.2. Listas

4. Computabilidad

4.1. Funciones recursivas

Procesos lineales recursivos

Procesos lineales iterativos

4.2. Un lenguaje Turing-completo

5. Funciones de orden superior

5.1. map()

5.2. filter()

5.3. reduce()