FP - John Backus

Unidad 2 - Programación III

John Warner Backus

Científico de la computación estadounidense.

- Trabajó en **IBM**
- Creador de **FORTRAN** (1957)
- Diseñó el sistema de programación funcional **FP** (1977)
- **Premio Turing** (1977)

"Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs"

Fundamentos

Sistema FP

El sistema FP es un lenguaje de programación funcional basado en la definición y composición de funciones sobre objetos.

- Programación funcional
- Definición de funciones
- Átomos
- Aplicaciones

Un programa funcional es una **expresión** compuesta por un algoritmo y sus entradas, cuyo resultado se obtiene evaluando la expresión.

Elementos del sistema FP

- **Conjunto O** de objetos
- **Operación** → Aplicación
- **Conjunto F** de funciones, objetos → objetos
- **Conjunto FF** de formas funcionales, funciones + objetos → funciones
- **Conjunto D** de definiciones de funciones

Objetos

Un objeto puede ser:

- Un **átomo** (letras, cadenas, números), menos las palabras reservadas
- Una **secuencia** de objetos `<x1, x2, ... , xn>`
- **Indefinido** (`⊥`)

Aclaraciones

- El átomo `⌀` representa la secuencia vacía, y es el único objeto que es a la vez átomo y secuencia
- Los átomos `T` y `F` se utilizan para booleanos
- Si una secuencia contiene `⊥`, está indefinida. Por ejemplo: `<5, A, 7, 4, ⊥, 3> = ⊥`

Ejemplos - Objetos

```
⊥
3.2
ø
PEPE
<PEPE, 3.2, ⊥>
<A, <<B>, C>, D>
<>              Obs: <> equivale a ø
```

Aplicación

Si f es una función y x es un objeto, entonces `f : x` es una aplicación y representa el objeto que resulta cuando se le aplica f a x.

Ejemplos:

```
-  : <10, 7>                 resulta 3
1  : <JUAN, CARLOS, SOFIA>   resulta JUAN
3  : <PRIMERO, SEGUNDO>      resulta ø
tl : <1, 2, 3>               resulta <2, 3>
```

Funciones (F)

Todas las funciones f del conjunto F convierten objetos en otros objetos, y preservan el valor indefinido (`f : ⊥ = ⊥`).

Las **funciones primitivas** son las funciones básicas provistas por el sistema FP.

Selectores

```
Selector desde la izquierda
  3  : <34, 1, 25>                        resulta 25
  4  : <A, B, C>                          resulta ⊥

Selector desde la derecha
  3r : <PRIMERO, SEGUNDO, TERCERO>        resulta PRIMERO
  4r : <1, 2, 3>                          resulta ⊥

Cola desde la izquierda
  tl : <A, B, C>                          resulta <B, C>

Cola desde la derecha
  tlr : <A, B, C>                         resulta <A, B>

Identidad
  id : <A, B, C>                          resulta <A, B, C>
```

Predicados

```
¿Es átomo?
  atom : 5                 resulta T
  atom : <A, B, C>         resulta F

¿Es igual?
  eq : <A, A>              resulta T
  eq : <A, 7>              resulta F
  eq : <A, B, C>           resulta ⊥

¿Es nulo?
  null : ø                 resulta T
  null : <A, 7>            resulta F
```

Funciones aritméticas

```
Suma
  + : <2, 7>          resulta 9
  + : <3, A, 7>       resulta ⊥

Resta
  - : <9, 7>          resulta 2
  - : <7, 9>          resulta -2

Producto
  × : <2, 7>          resulta 14
  × : <0, 5>          resulta 0

Cociente
  ÷ : <10, 2>         resulta 5
  ÷ : <10, 0>         resulta ⊥
```

Funciones lógicas

```
AND lógico
  and : <T, F>          resulta F
  and : <1, 0>          resulta ⊥

OR lógico
  or : <T, F>           resulta T
  or : <F, F>           resulta F

NOT lógico
  not : F               resulta T
  not : T               resulta F
```

Funciones para manipular secuencias

```
Longitud
  length : <2, A, 7>                        resulta 3
  length : ø                                 resulta 0

Invertir
  reverse : <2, A, 7>                       resulta <7, A, 2>
  reverse : ø                                resulta ø

Transponer
  trans : <<1, 2, 3>, <A, B, C>>            resulta <<1, A>, <2, B>, <3, C>>

Distribuir desde la izquierda
  distl : <A, <1, 2, 3>>                    resulta <<A, 1>, <A, 2>, <A, 3>>
```

Funciones para manipular secuencias (cont.)

```
Distribuir desde la derecha
  distr : <<1, 2, 3>, A>                    resulta <<1, A>, <2, A>, <3, A>>

Concatenar a la izquierda
  apndl : <<A, B>, <C, D>>                  resulta <<A, B>, C, D>

Concatenar a la derecha
  apndr : <<A, B>, <C, D>>                  resulta <A, B, <C, D>>

Rotar hacia la izquierda
  rotl : <A, B, C, D>                       resulta <B, C, D, A>

Rotar hacia la derecha
  rotr : <A, B, C, D>                       resulta <D, A, B, C>
```

Formas funcionales (FF)

Composición

$$f \circ g : x \equiv f : (g : x)$$

Ejemplo:

```
1 o tl : <A, B, C>
```

```
≡ 1 : (tl : <A, B, C>)
≡ 1 : <B, C>
resulta B
```

Construcción

$$[f_1, \ldots, f_n] : x \equiv \langle f_1 : x,\ f_2 : x,\ \ldots,\ f_n : x \rangle$$

Ejemplo:

```
[tl, tlr] : <A, B, C>
```

```
≡ <tl : <A, B, C>, tlr : <A, B, C>>
≡ <<B, C>, <A, B>>
```

Condición

$$(p \to f;\ g) : x \equiv (p : x) = T \to f : x\ ;\ (p : x) = F \to g : x\ ;\ \bot$$

Ejemplo:

```
(not o atom → 1; id) : <A, B, C>
```

```
not o atom : <A, B, C> = not : (atom : <A, B, C>) = not : F = T
→ 1 : <A, B, C>
resulta A
```

Constante

$$\overline{X} : y \equiv y = \bot \to \bot \; ; \; X$$

Ejemplo:

```
+ o [id, 1̄] : 3
```

```
≡ + : ([id, 1̄] : 3)
≡ + : <id : 3, 1̄: 3>
≡ + : <3, 1>
resulta 4
```

Inserción

$$/f : x \equiv x = \langle x_1 \rangle \rightarrow x_1 \; ; \; x = \langle x_1, \ldots, x_n \rangle \rightarrow f : \langle x_1, \; /f : \langle x_2, \ldots, x_n \rangle \rangle \; ; \; \perp$$

Ejemplo:

```
/+ : <1, 2, 3>
```

```
≡ + : <1, /+ : <2, 3>>
```

```
≡ + : <1, + : <2, 3>>
≡ + : <1, 5>
resulta 6
```

Aplicación a todos

$$\alpha f : x \equiv x = \emptyset \to \emptyset \; ; \; x = \langle x_1, \ldots, x_n \rangle \to \langle f : x_1, \ldots, f : x_n \rangle \; ; \; \bot$$

Ejemplo:

```
α 1 : <<A, B, C>, <4, 5, 6>>
```

```
≡ <1 : <A, B, C>, 1 : <4, 5, 6>>
≡ <A, 4>
```

Binario a unario

$$(bu\ f\ x) : y \equiv f : \langle x, y \rangle$$

Ejemplo:

```
(bu + 1) : 2
```

```
≡ + : <1, 2>
resulta 3
```

While

$$(while\ p\ f) : x \equiv p : x = F \to x\ ;\ p : x = T \to (while\ p\ f) : (f : x)\ ;\ \bot$$

Ejemplo:

```
(while (not o null o tl) tl) : <A, B, C, D, E, F, G, H>
```

```
not o null o tl : <A, B, C, D, E, F, G, H> = T
→ (while ...) : (tl : <A, B, C, D, E, F, G, H>)
→ (while ...) : <B, C, D, E, F, G, H>
→ (while ...) : <C, D, E, F, G, H>
→ ... → (while ...) : <G, H>
→ (while ...) : <H>
not o null o tl : <H> = not : (null : ∅) = not : T = F
resulta <H>
```

Definición de funciones (D)

Función iota

```
Def iota ≡ funrec o [id, <>]
Def funrec ≡ < o [1, 1] → 2; funrec o [- o [1, 1], apndl]
```

Ejemplo:

```
iota : 5    resulta <1, 2, 3, 4, 5>
```

Función factorial

```
Def !    ≡ eq0 → 1; × o [id, ! o sub1]
Def eq0  ≡ eq o [id, 0]
Def sub1 ≡ - o [id, 1]
```

Definición más funcional:

```
Def fact ≡ eq0 → 1; (/×) o iota
```

Expansión de ! : 4

```
! : 4
≡ × o [id, ! o sub1] : 4          // eq0 : 4 = F
≡ × : <4, ! : 3>
≡ × : <4, × : <3, ! : 2>>
≡ × : <4, × : <3, × : <2, ! : 1>>>
≡ × : <4, × : <3, × : <2, × : <1, ! : 0>>>>
≡ × : <4, × : <3, × : <2, × : <1, 1>>>>   // eq0 : 0 = T → 1
≡ × : <4, × : <3, × : <2, 1>>>
≡ × : <4, × : <3, 2>>
```

Función producto interno

```
Def IP ≡ (/+) o (α×) o trans
```

Ejemplo: IP : <<1, 2, 3>, <4, 5, 6>>

```
trans : <<1, 2, 3>, <4, 5, 6>>
  = <<1, 4>, <2, 5>, <3, 6>>
```

```
(α×) : <<1, 4>, <2, 5>, <3, 6>>
  = <× : <1, 4>, × : <2, 5>, × : <3, 6>>
  = <4, 10, 18>
```

```
(/+) : <4, 10, 18>
  = + : <4, + : <10, 18>>
  = + : <4, 28>
  resulta 32
```

Función producto matricial

```
Def MM ≡ (α α IP) o (α distl) o distr o [1, trans o 2]
```

Ejemplo:

```
MM : <<<1, 2, 3>, <4, 5, 6>, <7, 8, 9>>,
      <<1, 1, 1>, <0, 0, 0>, <0, 1, 0>>>

resulta <<1, 4, 1>, <4, 10, 4>, <7, 16, 7>>
```

Resumen

- **Objetos**: átomos, secuencias e indefinido
- **Aplicación**: `f : x` como operación fundamental
- **Funciones primitivas**: selectores, predicados, aritméticas, lógicas, manipulación de secuencias
- **Formas funcionales**: composición, construcción, condición, constante, inserción, α, bu, while
- **Definición de funciones**: iota, factorial, producto interno, producto matricial

¡Muchas Gracias!