Índice

FP

A saber, FP tiene átomos y secuencias. Los átomos pueden ser lógicos (true, false) o numéricos. Todo lo demás, se hace con funciones. Recordar que FP trabaja siempre sobre el ambiente, no se le pasan parámetros a las funciones.

Tipos de Funciones

Notación: $\underbrace{tl}_{función} : \underbrace{< a\ b\ c >}_{ambiente\ inicial} = \underbrace{< b\ c >}_{ambiente\ resultante}$

Funciones primitivas

- Selectoras
  - $1° : < a\ b\ c > = a$
  - $2° : < a\ b\ c > = b$
  - $1°r : < a\ b\ c > = c$
  - $2°r : < a\ b\ c\ d\ e > = d$
  - ...
  - $tl : < a\ b\ c > = < b\ c >$
  - $tlr : < a\ b\ c > = < a\ b >$
- $id : < a\ b > = < a\ b >$ (Identidad)
- $atom : < a > = false$; $atom : b = true$
- $null : < a > = false$; $null : <> = true$; $null : \emptyset = true$
- Relacionales (requieren que el ambiente sea una secuencia de dos átomos)
  - $> : < 4\ 8 > = false$
  - $< : < 12\ 34 > = true$
  - $eq : < 5\ 5 > = true$ (No sé si compara secuencias)
- $iota : 4 = < 1\ 2\ 3\ 4 >$ (Secuencia de números de 1 a n)
- $reverse : << a\ b > c < d\ e >> = << d\ e > c < a\ b >>$
- $distl : < y < a\ b\ c >> = << y\ a >< y\ b >< y\ c >>$ (Distributiva)
- $distr : << a\ b\ c > y > = << a\ y >< b\ y >< c\ y >>$
- $length : < \underbrace{< a\ b >}_{1} \underbrace{c}_{2} \underbrace{< d\ e\ f >}_{3} \underbrace{g}_{4} > = 4$
- Aritméticas (requieren que el ambiente sea una secuencia de dos átomos)
  - $+ : < 2\ 6 > = 8$
  - $- : < 5\ 2 > = 3$
  - $\times : < 7\ 3 > = 21$
  - $\div : < 8\ 4 > = 2$
- Lógicas (requieren que el ambiente sea una secuencia de dos átomos)
  - $and : < true\ false > = false$
  - $or : < false\ true > = true$
  - $not : true = false$
- $trans : << a\ b\ c >< d\ e\ f >> = << a\ d >< b\ e >< c\ f >>$ (Transposición)
- $apndl : << x\ y >< a\ b\ c >> = << x\ y > a\ b\ c >$
- $apndr : << a\ b\ c > y > = < a\ b\ c\ y >$
- $rotl : < a\ b\ c\ d\ e > = < b\ c\ d\ e\ a >$
- $rotr : < a\ b\ c\ d\ e > = < e\ a\ b\ c\ d >$

Formas Funcionales

- Composición ∘ (Ej: $(tl \circ distl) : < a < b\ c >> = tl: << a\ b >< a\ c >> = << a\ c >>$)
- Constante ¯ (Ej: $\overline{x}$ dará indefinido si el ambiente está indefinido, sino siempre x)
- Construcción [ ]
  - Ej: $[\text{tl}, \text{apndl}, \text{distl}, \text{null}]: < a < b\ c >> = < \underbrace{< b\ c >}_{\text{tl}} \underbrace{< a\ b\ c >}_{\text{apndl}} \underbrace{<< a\ b >< a\ c >>}_{\text{distl}} \underbrace{\text{false}}_{\text{null}} >$
- Condicional → ,
  - Ej: $\left(\underbrace{null \circ tl}_{cond} \rightarrow\ ,1°, \underbrace{tlr}_{else}\right) : < a > = a$
  - Ej: $\left(\underbrace{null \circ tl}_{cond} \rightarrow\ ,1°, \underbrace{tlr}_{else}\right) : < a\ b > = < a >$
- Inserción /
  - Ej: $/+ < 1\ 2\ 3 > =\ +: < 1\ \ /+ < 2\ 3 >> =\ +: < 1 +: < 2\ 3 >> =\ +: < 1\ 5 > = 6$
- Aplicar a todo α
  - Ej: $\alpha\ tl: << a\ b\ c >< d\ e\ f >< g\ h\ i >> = << b\ c >< e\ f >< h\ i >>$
- Iteración while
  - Ej: $\left(while\ \underbrace{not \circ null}_{cond}\ \ \underbrace{tl}_{funcion}\right) : < a\ b\ c > = << c >>$

Funciones definidas por el programador

Tus propias funciones, que luego se utilizan como cualquier otra. Por ejemplo:

$$\underbrace{def}_{\substack{palabra\ reservada}}\ \ \underbrace{pertenece}_{nombre} \equiv \underbrace{/or \circ\ \alpha\ eq\ \circ distl}_{implementación}$$

APL

Tipos de funciones

Próximamente

Modo programa

Próximamente una explicación. Por ahora, hay que saber que ∇ abre y cierra el programa. Se pueden crear funciones y procedimientos.

A continuación pongo el único ejemplo que vimos en clase.

Ejemplo

$A \leftarrow 10$
$x \leftarrow 7$

$\nabla R \leftarrow fun\ y\ ; A$
$[1]\ A \leftarrow 5$
$[2]\ R \leftarrow x \leftarrow x + A + y\ \nabla$

$\nabla\ proc\ x$
$[1]\ fun\ A \times 2\ \nabla$

Invoco	Ambiente	Salida
1. proc 8	(Desde el comienzo) A ← 10 x ← 7	1. 33
2. A	(Empieza 1.) x ← 8 y ← 20 A ← 5	2. 10
3. x	x ← 33 R ← 33 Desapilo A, R, x, y	3. 7
4. fun 22	(Empieza 4.) y ← 22 A ← 5	4. 34
5. A	x ← 34 R ← 34	5. 10
6. x	Desapilo y, A, R	6. 34

Lambda

Introducción

Formato genérico de una expresión Lambda: $(\lambda x.\, x + 1)5$, donde x es el parámetro, $x + 1$ es el cuerpo, y 5 es el valor en el cual se evalúa la función. En este caso, daría como resultado, 6.

Hay tres tipos de datos:

- Expresión simple: x, y, 3, etc.
- Abstracción $\lambda x.\, M$
- Aplicación: AB, siendo A y B dos abstracciones

Las variables pueden estar ligadas o libres.
La asociatividad es de izquierda a derecha.

Notación BNF

$G = (T, N, S, P), con$:

- G: Gramática
- T: Terminales. Elementos finales, aquellos que no deben definirse.
- N: No terminales (O símbolos). todo lo que debe definirse
- S: Símbolo distinguido. No estoy seguro de qué sería
- P: Producciones. Lo que se puede obtener usando esta gramática

Por ejemplo, en los números binarios:

- T: {0, 1}
- N: {S}
- S: 0 | 1 | 0S | 1S
- P:0, 1, 01, 10, 11, 001, 010, 011, 100, 101, 110, 111, …

Esta notación considera 1, 01, 001, etc, como valores distintos. Si no se quisiera, no se permite el uso de 0S.

Ejemplo de notación BNF para la sentencia IF:

```
<sentencia if> ::= if <condición><sentencia> | if <condición> <sentencia> else
<sentencia>
```

Otra forma de escribir lo mismo, es usando corchetes para indicar que algo es optativo:

```
<sentencia if> ::= if <condición> <sentencia> [else <sentencia>]
```

Para Lambda Calculus:

```
<expresión λ> ::= <expresión simple> | <abstracción> | <aplicación>

<expresión simple> ::= <átomo> | (<expresión λ>)

<abstracción> ::= λ <variable> . <expresión λ>

<aplicación> ::= <expresión simple><expresión simple> | <aplicación> <expresión simple>
```

```
<átomo> ::= <var> | <cte>

<var> ::= a | b | … | x | y | z

<cte> ::= <número>

<número> ::= <digito> | <digito><numero>

<digito> ::= 0 | 1 | 2 | … | 8 | 9
```

Donde lo pintado de rojo son los elementos terminales (notar los paréntesis y el punto).

A saber:

- `::=` significa Definición
- `|` significa OR

Reglas de sustitución

Notación: $M\rfloor_x^N = Sustituye\ las\ x\ por\ N, en\ M$

1. $Si\ M = z;\ M\rfloor_x^N = z; si\ z \neq x$
2. $Si\ M = x;\ M\rfloor_x^N = N$
3. $Si\ M = AB;\ M\rfloor_x^N = A\rfloor_x^N\ B\rfloor_x^N$
4. $Si\ M = \lambda x.P;\ M\rfloor_x^N = \lambda x.P$
5. $Si\ M = \lambda z.P;\ M\rfloor_x^N = \lambda z.P\rfloor_x^N, si\ z \neq x$ y z no libre en N

Reglas de conversión

Regla α (Cambio de variable)

$$\lambda x.M \overset{\alpha}{\Rightarrow} \lambda y.M\rfloor_x^y$$

Reglas de reducción

Notación:

β_{redex}: $(\lambda x.M)N$

η_{redex}: $\lambda x.Mx$

Regla β

$$(\lambda x.M)N \overset{\beta}{\Rightarrow} M\rfloor_x^N$$

Ejemplo:

$$(\lambda x.abxtx)8 \overset{\beta}{\Rightarrow} ab8t8$$

Regla η

$$\lambda x.Mx \overset{\eta}{\Rightarrow} M$$

Orden de evaluación

Llevar una función, a su expresión normal, es decir, una expresión que no contenga β_{redex}

Orden normal

Equivalente al paso por nombre

$(\lambda x.M)[(\lambda y.P)N] \overset{\beta_{on}}{\Longrightarrow} M\rfloor_x^{(\lambda y.P)N}$

Ejemplo:

$((\lambda x.axbx)[(\lambda t.atv)z] \overset{\beta_{on}}{\Longrightarrow} a[(\lambda t.atv)z]b[(\lambda t.atv)z] \overset{\beta}{\Rightarrow} a[azv]b[azv]$

Orden aplicativo

Equivalente al paso por valor

$(\lambda x.M)[(\lambda y.P)N] \overset{\beta_{oa}}{\Longrightarrow} M\rfloor_x^{P\rfloor_y^N}$

Ejemplo:

$((\lambda x.axbx)[(\lambda t.atv)z] \overset{\beta_{oa}}{\Longrightarrow} (\lambda x.axbx)(azv) \overset{\beta}{\Rightarrow} a(azv)b(azv)$

Combinadores

1. $I = \lambda x.x$ **(Identidad)** $Ia = a$
2. $K = \lambda x.\lambda y.x$ **(Selector 1° de 2)** $Kab = a$
3. $O = \lambda x.\lambda y.y$ **(Selector 2° de 2)** $Oab = b$
4. $B = \lambda x.\lambda y.\lambda z.x(yz)$ **(Compositor)** $Bfg = f \circ g$
5. $C = \lambda x.\lambda y.\lambda z.xzy$ **(Permutador)** $Cabc = acb$
6. $T = \lambda x.\lambda y.yx$; $Tab = ba$
7. $S = \lambda x.\lambda y.\lambda z.xz(yz)$; $Sabc = ac(bc)$
8. $W = \lambda x.\lambda y.xyy$ **(Duplicador)** $Wab = abb$
9. $Y = \lambda h.(\lambda x.xx)(\lambda y.h(yy))$ **(Paradójico)** $Y(\lambda x.a) = a$ (Da los puntos fijos de una función, en este caso f = a)

Representación de operadores lógicos

$p \to q, r$ se representa como pqr

$True = K$ (porque $Kqr = q$)

$False = O$ (porque $Oqr = r$)

$and = \lambda p.\lambda q.pq\ False$

- $and\ True\ True = (\lambda p.\lambda q.pq.False)True\ True \overset{\beta}{\Rightarrow} True\ True\ False = True$, porque $True = K$
- $and\ True\ False = False$
- $and\ False\ True = False$
- $and\ False\ False = False$

$or = \lambda p.\lambda q.p\ True\ q$

Representación de números naturales

$n = \lambda f.\lambda x.\underbrace{f(f(f(\dots f}_{n\ veces}\ x)))$

- $0 = \lambda f.\lambda x.x$
- $1 = \lambda f.\lambda x.f\ x$
- $2 = \lambda f.\lambda x.f(f\ x)$

Función sucesor

$suc = \lambda n.\lambda f.\lambda x.n.f(f\ x)$

$suc\ 2 = \underbrace{(\lambda n.\lambda f.\lambda x.n.f(f\ x))}_{suc}\underbrace{(\lambda f.\lambda x.f(f\ x))}_{2} = \underbrace{(\lambda f.\lambda x.f(f\ (f\ x)))}_{3} = 3$

LISP

Funciones Thunks

Funciones constantes, que se utilizan, por ejemplo, para evitar errores al momento de compilar. Ejemplo (para TLC_LISP):

```
(defun Y (A B) (and (A)(B)) ) ; Defino un AND

(defun thunknil ( ) nil)      ; Defino un nil

(defun thunkdiv0 () (/ 1 0) ) ; Defino algo que va a dar error

(y thunknil thunkdiv0)        ; Esto NO da error

(y nil (/ 1 0))               ; Esto SI da error, porque primero resuelve el (/ 1 0) para
pasárselo a Y
```

Pattern Matching

No estoy seguro de qué es exactamente, lo averiguo y lo agrego.