Índice

[FP 2](#_Toc268880203)

[Tipos de Funciones 2](#_Toc268880204)

[APL 4](#_Toc268880205)

[Tipos de funciones 4](#_Toc268880206)

[Modo programa 4](#_Toc268880207)

[Lambda 5](#_Toc268880208)

[Introducción 5](#_Toc268880209)

[Notación BNF 5](#_Toc268880210)

[Reglas de sustitución 6](#_Toc268880211)

[Reglas de conversión 6](#_Toc268880212)

[Reglas de reducción 6](#_Toc268880213)

[Orden de evaluación 6](#_Toc268880214)

[Orden normal 7](#_Toc268880215)

[Orden aplicativo 7](#_Toc268880216)

[Combinadores 7](#_Toc268880217)

[Representación de operadores lógicos 7](#_Toc268880218)

[Representación de números naturales 8](#_Toc268880219)

[LISP 9](#_Toc268880220)

[Funciones Thunks 9](#_Toc268880221)

[Pattern Matching 9](#_Toc268880222)

# 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:

#### Funciones primitivas

* Selectoras
  + …
* (Identidad)
* ;
* ; ;
* Relacionales (requieren que el ambiente sea una secuencia de dos átomos)
  + (No sé si compara secuencias)
* (Secuencia de números de 1 a n)
* (Distributiva)
* Aritméticas (requieren que el ambiente sea una secuencia de dos átomos)
* Lógicas (requieren que el ambiente sea una secuencia de dos átomos)
* (Transposición)

#### Formas Funcionales

* Composición (Ej: )
* Constante (Ej: dará indefinido si el ambiente está indefinido, sino siempre x)
* Construcción [ ]
  + Ej:
* Condicional
  + Ej:
  + Ej:
* Inserción /
  + Ej:
* Aplicar a todo α
  + Ej:
* Iteración while
  + Ej:

#### Funciones definidas por el programador

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

# 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

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

# Lambda

## Introducción

Formato genérico de una expresión Lambda: , donde es el parámetro, 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: , , 3, etc.
* Abstracción
* Aplicación: , siendo y dos abstracciones

Las variables pueden estar ligadas o libres.

La asociatividad es de izquierda a derecha.

## Notación BNF

:

* 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:

1. y no libre en

## Reglas de conversión

**Regla α** (Cambio de variable)

## Reglas de reducción

Notación:

**Regla β**

Ejemplo:

**Regla η**

## Orden de evaluación

Llevar una función, a su expresión normal, es decir, una expresión que no contenga

### Orden normal

Equivalente al paso por nombre

Ejemplo:

(

### Orden aplicativo

Equivalente al paso por valor

Ejemplo:

(

## Combinadores

1. **(Identidad)**
2. **(Selector 1° de 2)**
3. **(Selector 2° de 2)**
4. **(Compositor)**
5. **(Permutador)**
6. ;
7. ;
8. **(Duplicador)**
9. **(Paradójico)**  (Da los puntos fijos de una función, en este caso f = a)

## Representación de operadores lógicos

se representa como

(porque )

(porque )

* porque

## Representación de números naturales



Función sucesor

# 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.