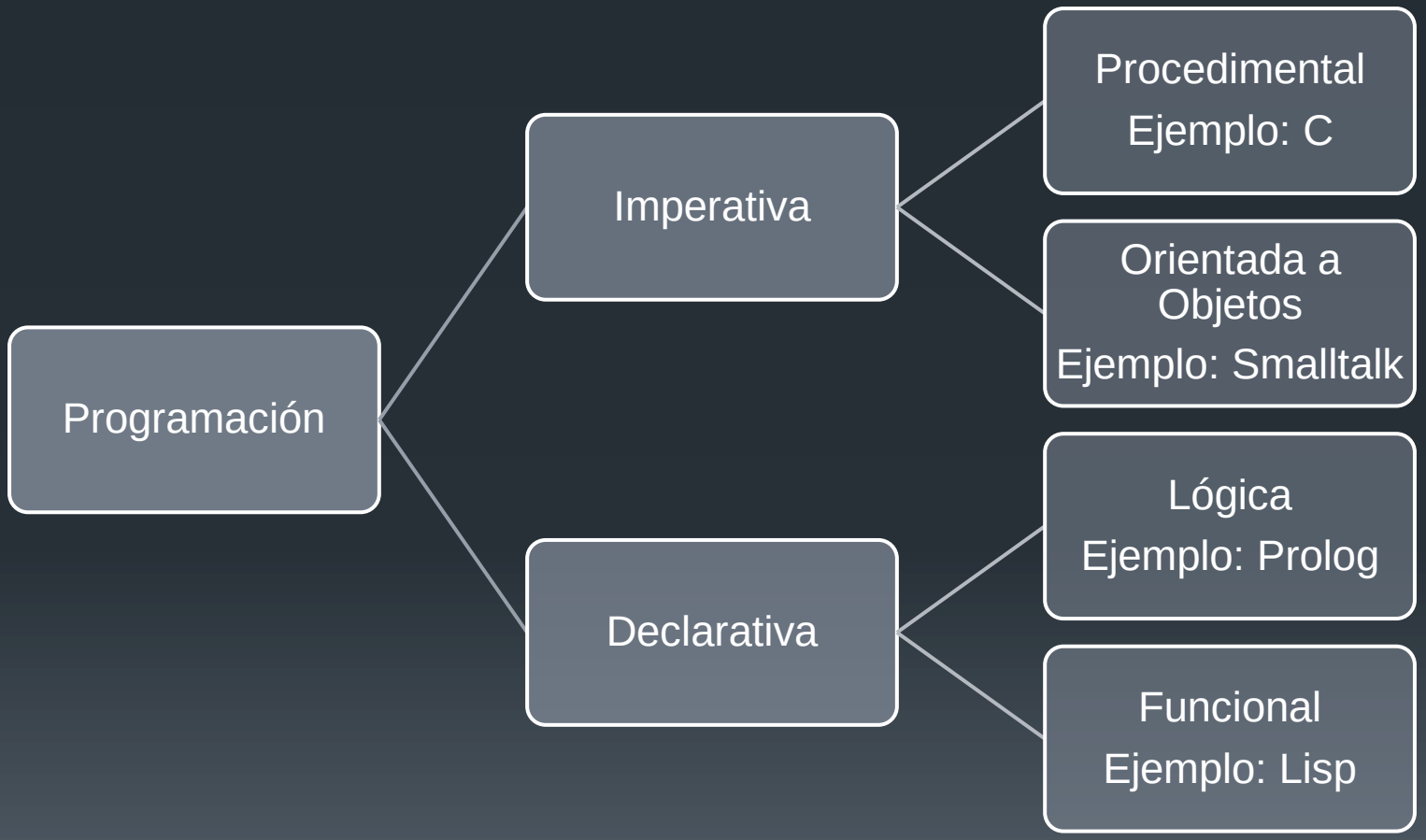
Aplicación de la Inducción Matemática Programación Funcional

Julio Ariel Hurtado Alegría

Departamento de Sistemas

Universidad del Cauca

Motivación


Programación
Imperativa
Declarativa
Procedimental
Ejemplo: C
Orientada a Objetos
Ejemplo: Smalltalk
Lógica
Ejemplo: Prolog
Funcional
Ejemplo: Lisp

Programación lógica y funcional: Antecedentes Históricos.

- 30’s Alan Turing, Alonzo Church, Stephen Kleene y Emil Post desarrollan por separado formalizaciones de la noción de algoritmo.
- Turing → Máquina de Turing
- Church → equivalencia entre lenguajes, Lambda Cálculo.
- Kleene y Post → definiciones abstractas sin vinculación directa a la implementación de un lenguaje de programación.

Programación Funcional: Concepto.

- La programación funcional define las salidas de un programa como una función matemática de sus entradas, sin noción de estado interno, y, por tanto sin efectos de borde.
- Lisp → lenguaje funcional original y el más utilizado.
- Algunos lenguajes funcionales: Scheme, Common Lisp, Ml, Miranda, Haskell, Sisal.

Programación Funcional: Características prácticas

- Valores de Funciones de Primer Orden y Funciones de Orden Superior.
- Polimorfismo.
- Tipo Lista y sus Operadores.
- Recursión.
- Retorno de datos estructurados.
- Constructores de objetos estructurados.
- Recolección de Basura.

Ventajas y Desventajas de los lenguajes funcionales.

- Ventajas:
  - Más fáciles de escribir, depurar y mantener que los lenguajes imperativos gracias a la ausencia de efectos de borde.
- Desventajas:
  - Se quedan cortos en portabilidad, riqueza de librerías, interfaces con otros lenguajes y herramientas de depuración.

Caso de Estudio: Scheme

- Procede de Lisp que es el lenguaje funcional más utilizado.
- Características adicionales de Lisp aplicables a Scheme:
  - Homogeneidad de programas y datos.
  - Autodefinición.
  - Interacción con el usuario a través de un ciclo “leer-evaluar-imprimir”.

Scheme: Características

- Notación *Cambridge-Polish*
- La utilización de paréntesis indican la aplicación de una función o el uso de un macro.
- El símbolo de quoting (‘) evita que el interpretador evalúe una expresión.
- El tipo de los elementos es determinado en tiempo de ejecución.

Scheme: Predicados

- A pesar de ser un lenguaje funcional Scheme ofrece entre otros los siguientes predicados:

(boolean? x)
(char? x)
(string? x)
(symbol? x)
(member? x lst)
(pair? x)
(list? x)

#t
#f

Scheme: Funciones

- Las funciones se definen a través de la palabra reservada “lambda”
  - (lambda (x) (* x x))

- Expresiones condicionales pueden ser escritas utilizando un “if”.

Ejemplo:
(if (< 2 3) 4 5) → 4

Scheme: Asociación (*Binding*)

- Para asociar un nombre a una función se utilizan las expresiones “let” (localmente) o “define” (globalmente).

```
(let ((a 3)
      (b 4)
      (square (lambda (x) (* x x)))
      (plus +)
      )

(sqrt (plus (square a) (square b))))
```

Lista de pares nombre valor

Aplicación de funciones definidas

Scheme: Listas y Números

- Los principales operadores definidos para la manipulación de listas son los siguientes:
  - `car` → retorna la cabeza de la lista.
  - `cdr` → retorna el resto de la lista.
  - `cons` → agrega un elemento a la cabecera de la lista.
  - `null?` → determina si u argumento es la lista vacía.

- Tipos Numéricos: integer, rational, real, complex, number.

Scheme: Flujo de Control y Asignaciones.

- Una secuencia de intrucciones al estilo “if-else” puede implementarse con la función “cond”.

```
(cond
 ((< 3 2) 1)
 ((< 4 3) 2)
 (else 3))          ⇒ 3
```

Listas: car, cdr, cons

- (car '(1 2 3)) → 1
- (cdr '(1 2 3)) → (2 3)
- (car '()) o (cdr '()) → error
- (cons 0 '(1 2 3)) → (0 1 2 3)
- (car (cons x lst)) → x
- (cdr (cons x lst)) → l

Largo de una Lista

```
(define largo
        (lambda (lst)
                (if (null? lst)
                        0
                        (+ (largo (cdr lst) ) 1)
                )
        )
)
```

Suma de los elementos de una Lista

```
(define (sum l)
        (cond
           ((null? l) 0)
           (else (+ (car l) (sum (cdr l))))
        )
)
```

Map

```
(define (map proc lst)
        (cond
            ((null? lst) '())
            (else (cons (proc (car lst)) (map proc (cdr lst))))
         )
)
```

- Ejemplo :

(map sqrt '(25 9 16)) → (5 3 4)

Invertir una lista

```
(define (rev l) (rev2 l '()))
(define (rev2 l r)
    (cond
      ((null? l) r)
       (else (rev2 (cdr l) (cons (car l) r)))
      )
  )
```

Recursión por la cola

```
(define (largo l)
            (cond
                ((null? l) 0)
                (else (+ 1 (largo (cdr l))))
            )
)

 (define (largo l) (tail-len l 0))
 (define (tail-len l s)
            (cond
                ((null? l) s)
                (else (tail-len (cdr l) (+ s 1)))
             )
 )
```

Funciones de Orden Superior

- (define (generica f x y) (f x y))
- (generica > 5 2)
- (generica (lambda (a b) (- b (* a 2))) 3 10)
- (generica append '(a b c) '(x y z))

Desarrollo P. Funcional Racket

- http://racket-lang.org/

Aplicación de la Inducción Matemática Programación Funcional

Julio Ariel Hurtado Alegría

Departamento de Sistemas

Universidad del Cauca