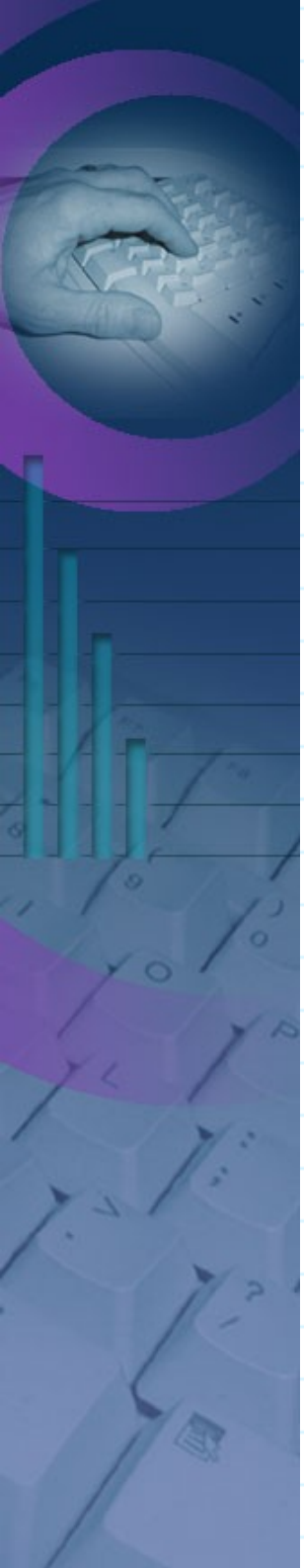


Tecnologías de Programación

Programación Funcional

Definición

La Programación funcional es un paradigma de programación declarativa basado en la utilización de funciones matemáticas.

Sus orígenes se remontan al cálculo lambda (o λ-cálculo), una teoría matemática elaborada por Alonzo Church (1930) como apoyo a sus estudios sobre computabilidad.

Características

- Los programas están constituídos únicamente por definiciones de funciones puramente matemáticas. Se verifican:
  - la Transparencia Referencial
  - la ausencia total de efectos colaterales
- No existen las asignaciones de variables.
- No existen las secuencias o iteraciones.

Lenguajes Funcionales

- Categorías de Lenguajes:
  - ***Puros**:* tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial.
    - Ejemplo: Haskel, Miranda.
  - ***Híbridos***: admiten secuencias de instrucciones o la asignación de variables.
    - Ejemplo: Scala, Lisp, Scheme, Ocaml y Standard ML.

Cálculo λ

- Es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión.
- Consiste en una regla de transformación simple (sustitución de variables) y un esquema simple para definir funciones.

Descripción informal

- Todas las expresiones representan funciones de un sólo argumento.
- Este argumento, a su vez, sólo puede ser una función de un sólo argumento.
- Todas las funciones son anónimas, y están definidas por expresiones lambda, que dicen que se hace con su argumento.
- La aplicación de funciones es asociativa a izquierda: f x y = (f x) y.

Descripción Informal

- Ejemplo: “sumar 2”
  - *f(x) = x+2 =====> λ x. x + 2*
    - f(3) = (λ x. x + 2) 3 → 5
  - Considerando la función que aplica una función al número 3: λ f. f 3.
    - (λ f. f 3) (λ x. X + 2).

(λ f. f 3)(λ x. x + 2) , (λ x. x + 2) 3 y 3 + 2

Definición Formal

- Gramática:
  - Conjunto infinito numerable de identificadores, por ejemplo {a, b, c,..., x, y, z, x1, x2,...}.
  - 1. <expr> ::= <identificador>
  - 2. <expr> ::= (λ <identificador>**.** <expr>)
  - 3. <expr> ::= (<expr> <expr>)

Variables Libres y Ligadas

- Las apariciones (ocurrencias) de variables en una expresión son de tres tipos:
  - Ocurrencias de ligadura (binders):
    - Son aquellas que están entre el λ y el punto.
    - Ejemplo: (λ x y z. E) Los binders son x,y y z.
  - Ocurrencias ligadas (bound occurrences):
    - Ejemplo: λ V. E → V está ligada en E
  - Ocurrencias libres (free occurrences):
    - Ejemplo: (E E′)

α Conversión

- Los nombres de las variables ligadas no son importantes.
  - λx.x y λy.y son la misma función.

α Conversión

- Sean V y W variables, E es una expresión lambda, y

  E[V := W]

  representa la expresión E con todas las ocurrencias libres de V en E reemplazadas con W, entonces

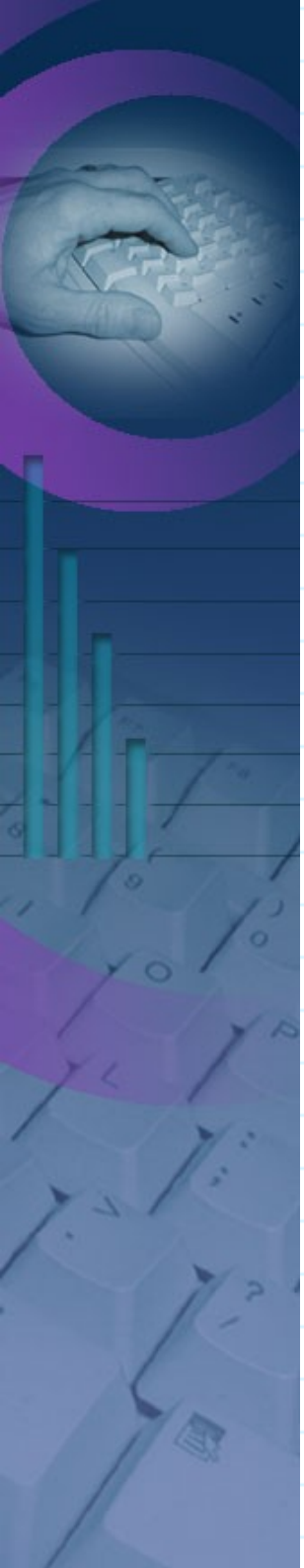
  λ V. E == λ W. E[V := W]

β-reducción

- La regla de beta reducción expresa la idea de la aplicación funcional. Enuncia que:

  ((λ V. E) E′) == E[V := E′]

  ((λ V. E) E′) es llamada un **beta redex**.



Tecnologías de Programación

Scheme

Scheme

- Introducción:
  - es un lenguaje funcional, derivado de LISP.
  - compacto con un alto grado de abstracción.
  - permite resolver problemas complejos con programas cortos y elegantes.

Sintaxis

- Palabras clave, variables y los símbolos son llamados Identificadores. Los identificadores pueden estar formados por:
  - [a-z]
  - [A-Z]
  - [0-9]
  - ? ! . + - * / < = > : $ % ∘ & _ ~ @
  - Ejemplo: Hola, n, x, x3, ?$&*!!!

Sintaxis

- Todos los identificadores deben estar delimitados por:
  - Un espacio en blanco
  - Comillas dobles (“)
  - Paréntesis
  - carácter de comentario (;)
- No hay límite de longitud
- No es case-sensitive:
  - Abc, abc, aBc son todos el mismo identificador

Sintaxis

- Las estructuras y las Listas se encierran entre paréntesis:
  - Ejemplo: (a b c) o (* (- x 2) y)
- () => Lista vacía
- Valores Booleanos:
  - #t → verdadero
  - #f → falso

Sintaxis

- Vectores:
  - comienzan con #(
  - Finalizan con)
  - Ejemplo: #(esto es un vector de símbolos)
- String: encerrados en “”
- Caracteres: precedidos por #\
  - Ej: #\a

Sintaxis

- Números:
  - Enteros: -123
  - Racionales: 1/2
  - En punto flotante: 1.3
  - Notación Científica: 1e12
  - Complejos en Notación Rectangular: 1.3-2.7i
  - Complejos en Notación Polar: -1.2@73

Convención de Nombres

- Predicados finalizan en ?: retornan #t o #f
  - Ejemplo: eq?, zero?, string=?
- El nombre de la mayoría de los procedimientos de string, caracteres y vectores comienzan con el prefijo string-, char- y vector-
  - Ejemplo: string-append
- Conversión entre tipos de objetos se escriben como *tipo1->tipo2.*
  - *Ej: vector->list*

Interactuando con Scheme

- Interface del tipo REPL (read-evaluate-print loop)
- Soporta diferentes versiones del lenguaje.
  - Utilizar: R5RS estándar

Interactuando con Scheme

- Probar:
  - "hola"
  - 42
  - 22/7
  - 3.141592653
  - +
  - (+ 76 31)
  - '(a b c d)

Interactuando con Scheme

- Identifique que hacen las funciones:
  - (car '(a b c))
  - (cdr '(a b c))
  - (cons 'a '(b c))
  - (cons (car '(a b c))
    (cdr '(d e f)))

Interactuando con Scheme

- Definir un procedimiento:

```
(define cuadrado
        (lambda (n)
                (* n n)))
```

- Y usarlo:
  - (cuadrado 5) ⇒ 25
  - (cuadrado -200) ⇒ 40000
  - (cuadrado 0.5) ⇒ 0.25
  - (cuadrado -1/2) ⇒ 1/4

Expresiones simples

- Notación prefija
  - (+ 2 2) ⇒ 4
  - (+ (+ 2 2) (+ 2 2)) ⇒ 8
  - (- 2 (* 4 1/3)) ⇒ 2/3
  - (* 2 (* 2 (* 2 (* 2 2)))) ⇒ 32
  - (/ (* 6/7 7/2) (- 4.5 1.5)) ⇒ 1.0

Expresiones simples

- Estructura de agregación: Listas (list)
  - (quote (1 2 3 4 5)) → lista de números
  - '(“esto” “es” “una” “lista”) → lista de strings
  - '(4.2 “hola”) → lista de múltiples tipos
  - '((1 2) (“hola” “scheme”)) → lista de listas

Expresiones Simples

- Quote (') le dice a Scheme que trate un identificador como símbolo y no como variable
- Los símbolos y variables en Scheme son similares a los símbolos y variables en expresiones matemáticas y ecuaciones
  - En expresiones Matemáticas:
    - 1 – x → pensamos en x como variable
  - En expresiones Algebraicas:
    - X^2-2 → pensamos en x como símbolo

Expresiones Simples

- Operadores de Listas
  - car: retorna el primer elemento de la lista
  - cdr (could-er): retorna el resto de la lista
    - (car '(a b c)) ⇒ a
    - (cdr '(a b c)) ⇒ (b c)
    - (cdr '(a)) ⇒ ()

Expresiones Simples

- Operadores de Listas: car – cdr
  - Resuelva:
    - (car (cdr '(a b c)))
    - (cdr (cdr '(a b c)))
    - (car '((a b) (c d)))
    - (cdr '((a b) (c d)))

Expresiones Simples

- Operadores de Listas:
  - cons: construye listas. Recibe dos argumentos. Usualmente el segundo es una lista y en ese caso retorna una lista
    - (cons 'a '()) ⇒ (a)
    - (cons 'a '(b c)) ⇒ (a b c)
    - (cons 'a (cons 'b (cons 'c '()))) ⇒ (a b c)
    - (cons '(a b) '(c d)) ⇒ ((a b) c d)

Expresiones Simples

- Operadores de Listas:
  - Resuelva:
    - (car (cons 'a '(b c)))
    - (cdr (cons 'a '(b c)))
    - (cons (car '(a b c))
      (cdr '(d e f)))
    - (cons (car '(a b c))
      (cdr '(a b c)))

Expresiones Simples

- Cons: construye pares. Cuando el segundo parámetro es una lista devuelve una lista *Propia*.
  - Lista Propia: Una lista vacia es una lista propia, y toda lista cuyo **cdr** sea una lista propia es una lista propia.
  - Listas impropias: compuestas por pares donde se marca las separación de elementos por puntos.

Expresiones simples

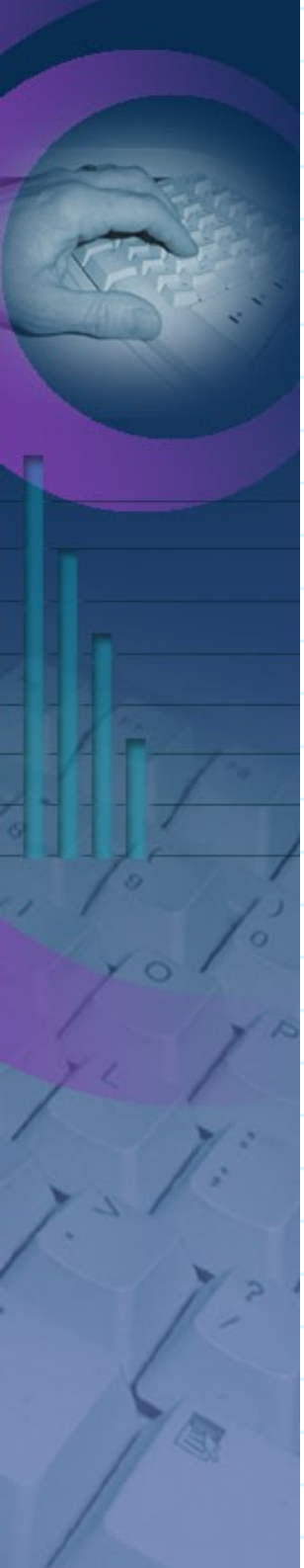
- (cons 'a 'b) → (a . b)
- (cdr '(a . b)) → b
  - A diferencia de (cdr '(a b)) → (b)
- (cons 'a '(b . c)) → (a b . c)

Evaluado de Expresiones

- (procedimiento arg1 arg2 ...)
  - Buscar el valor de *procedimiento*
  - Buscar el valor de *arg1*
  - Buscar el valor de *arg2*
  - ....
  - Aplicar el valor de *procedimiento* a los valores de *arg1, arg2, ...*

Evaluado de Expresiones

- Ejemplo:
  - (+ 3 4)
    - el valor de + es el procedimiento *adición*
    - el valor de 3 es el número 3
    - el valor de 4 es el número 4
    - aplicando el procedimiento *adición* a los números 3 y 4 devuelve 7
  - pruebe: ((car (cdr (list + - * /))) 17 5)



Programación Funcional

to be continued...