Herramientas para la programación Simbólica

Dr. Luis Alberto Pineda Cortés

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Expresiones simples

- Enunciados (*statement*):
  - se avalúan por sus efectos
  - asignaciones, input-output, control
- Expresiones
  - Su evaluación "regresa" valores
  - funciones: no hay efectos colaterales a la evaluación
- Scheme
  - Lenguaje funcional
  - No hay distinción entre "procedimientos" y funciones
  - Construcciones: definiciones y expresiones

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Literales, llamadas a procedimientos y variables

- Literales o constantes
  - Símbolos y numerales: objetos representacionales (sintácticos)
  - valores expresados: objetos representados (semántica)
  - Ejemplo: evalua el numeral **2**
    - valor el número dos: 2 (en itálicas en el texto)
    - representación externa: "2"
- Variables (permiten introducir abstracción)
  - Son expresiones (identificadores: **x**, **x3**, **foo**, **londid**, **+**, **/**, **zero?**, etc.)
  - Tienen valor o denotan un objeto (valores denotados).
  - Pueden ser ligadas (bound to) y denotan valor de su "binding".
- Identificadores
  - Keywords: **define**, **if**
  - Procedimientos estándar: **+, add**, **zero?**

Literales, llamadas a procedimientos y variables

- Sintaxis
  - (*operador operando-1 ... operando-2*)
  - tanto operandos como operadores son expresiones
  - la evaluación de *operadores* "regresa" procedimientos
  - la evaluación de *operandos* "regresa" argumentos o parámetros
  - el orden de evaluación de operandos no se especifica en Scheme
  - ejemplos:
    - (+ 2 3) valor: 5
    - (+ x (p 2 3)) valor: 9 is x vale 3 y p es el producto
    - ((g 2) 3 4) valor: 7 si (g 2) es la suma

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Definiciones y programas

- Un programa en Scheme consiste de:
  - Definiciones
  - expresiones
- Definiciones
  - (**define** *variable expresión*)
  - *keywords* tienen una secuencia de evaluación pre-establecida
  - **define**: la expresión se evalúa primero y después el valor se liga a la variable
- Ciclo *lee-evalúa-imprime*

Ciclo *lee-evalúa-imprime*

- > 3
- 3
- > * ; el valor es el procedimiento de multiplicación
- #<procedure>
- > (* 2 3)
- 6
- > (define x 3)
- > x
- 3
- > (+ x (* 2 3))
- 9

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Evaluación condicional

- Condicional (definición)
  - (**if** *test-exp then-exp else-exp*)
  - Si *test-esp* es verdad evalúa *then-exp* si es falso evalúa *else-exp*
- Ejemplo:
  - > (if #t 1 2)
  - 1
  - > (zero? 5)
  - #f
  - > (if (zero? 5) 1 (+ 1 2))
  - 3
  - > (define true #t)
  - > (define false #f)
  - > (if (zero? 0) (if false 1 2) 3)
  - 2
  - > (if (if true false true) 2 3)
  - 3
- Guardias (secuencia de evaluación de **if**)
  - (if (zero? a) 0 (/ x a))

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Tipos de datos

- Consideraciones para los tipos de datos
  - El conjunto de valores del tipo de dato
  - Los procedimientos que operan en el tipo de dato
  - Las representaciones "internas" de los tipos de datos (como literales) o "externas" como caracteres impresos
- Verificación del tipo de datos (type cheking)
  - Verificación dinámica en tiempo de ejecución (Scheme)
  - Verificación estática (en tiempo de compilación)
    - Los errores se detectan antes y la ejecución es más eficiente
    - Menos expresividad y complejidad del compilador

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Números y boleanos

- Tipo número
  - Valores: los números naturales (incluyendo el cero)
  - Operaciones: **+**, **-**, *****, **/**
  - Predicados: **number?**, **=**
- Tipo boleano
  - Valores: {#t, #f}
  - Operaciones: se usan en condicionales
  - Predicados: **boolean?**, **eq?**
  - Ejemplos:
    - > (eq? (boolean? #f) (not #f))
    - #t

Caracteres

- Tipo carácter (character)
  - Valores: representación interna (e.g., ascii)
  - Representación externa precediendo con "#\"
    - (e.g., #\a, #\%, #\space, #\newline)
  - Operadores: **char-> integer**
  - Predicados: **char?**, **char=?**, **char<?**, **char-alphabetic?**, **char-numeric?**, **char-whitespace?**
  - Ejemplos:
    - > (char? #\$)
    - #t
    - > (char=? #\space, #\newline)
    - #f
    - > (char<? #\a #\b)

Cadenas

- Tipo cadena (string)
  - Valores: secuencia de caracteres (e.g., ascii)
  - Operadores: **string-length**, **string-append**, **string->symbol**, **string->number**, **string->list**, **string** (transf. secuencia de caracteres en una cadena), **string-ref** (string e indice a caracter)
  - Predicados: string?
  - Ejemplos:
    - > (define s "This is a.")
    - > (define ss (string-append s "longer string")
    - > (string? s)
    - > #t
    - > (string-length ss)
    - > 23
    - > (string-ref s 2)
    - > #i
    - > (string \#a \#b)
    - > "ab"
    - > (string->symbol "abc")
    - > abc
    - > (string->list s)
    - > (\#T \#h \#i \#s \#space ... \#.)

Símbolos o Indentificador (nombres de variable)

- Cual es la diferencia entre “luis” y luis en el discurso normal?
  - prevenir la evaluación de símbols: (quote *datum*) o ’*datum*
    - > (define x 3) > 99
    - > x > 99
    - > 3 > (quote 99)
    - > (quote x) > 99
    - > x
  - predicados: symbol?, eq?
    - > (define x 3) > (symbol ’x) > y
    - > (number? x) > \#t > apple
    - > #\t > (eq? ’x ’x) > (eq? y (quote apple))
    - > (symbol? x) > \#t >\#t
    - > \#f > (eq? ’x ’y) > (eq? y ’y)
    - > (number? ’x) > \#f > \#f
    - > \#f >(define y ’apple)

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Listas

- Secuencia ordenada de elementos de tipos arbitrarios
  - Ejemplos:
    - (a 3 #3) () (a) ((b c d))
- Prevenir confusión entre listas y llamadas a procedimientos
  - (quote (a b c)) versus (a b c)
- Constructores: **list**, **cons**, **append**
  - Ejemplos con “list”
    - > (list 1 2 3) > (define list-2 ’(a))
    - > (1 2 3) > (define list-3 ’((b)))
    - > (define x 3) > (list list-1 list-2 list-3 ’(((c))))
    - > (define y ’apple) > (( ) (a) ((b)) (((c))))
    - > (list x y) > (list)
    - > (3 apple) > ( )
    - > (define list-1 ’( ))

Listas

- Constructores: **list**, **cons**, **append**
  - Ejemplos con “cons” (cons valor lista) -> lista
    - > (cons ’a ’(c d)) > (cons ’a ‘( )) > list-4
    - > (a c d) > (a) > (( ) a)
    - > (list ’a ’(c d)) > (cons ’(a b) ‘( )) > (cons list-4 list-3)
    - > (a (c d)) > ((a b)) > ((( ) a) (b))
    - > (cons ’(a b) ’(c d)) > (define y ’apple)
    - > ((a b) c d) > (cons y list-2)
    - > (cons ’( ) ’(c d)) > (apple a)
    - > (( ) c d) > (define list-4 (cons list-1 list-2))
  - Ejemplos con “append”
    - > (append ’(a b) ’(c d)) > (append ’(a b) ’( ))
    - > (a b c b) > (a b)
    - > (append ’( ) ’(c d))
    - > (c d)

Listas

- Selectores de lista: **car**, **cdr**
- Si $l = (v_0\ v_1 \ldots v_n)$ entonces $(car\ l) = v_0$ y $(\text{cdr}\ l) = (v_1 \ldots v_n)$
  - Ejemplos con *car* y *cdr*
    - > (car ’(a b c)) > (car (cdr ’(a b c)))
    - > a > b
    - > (cdr ’(a b c)) > (cdr ’(a))
    - > (b c) > ( )
  - Relaciones entre d**car** y **cdr** Combinaciones de **car** y cdr
    - $(car\ (cons\ v\ l)) = v$ $(cadr\ l) = (car\ (cdr\ l)$
    - $(cdr\ (cons\ v\ l)) = l$ $(caddr\ l)) = (car\ (cdr\ (cdr\ l)))$
  - Ejemplos
    - > (card ’(a b c)) > (c)
    - > b > (caddr ’(a b c))
    - > (cddr ’(a b c)) > c

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

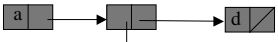
Pares (doted pairs)

(a . b)

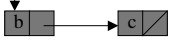
(car (a . b) = a

crd (a . b) = (b)

(a (b c) d)



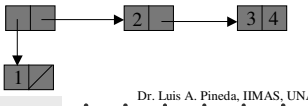
(a . ((b . (c . ( )) . (d . ( ))))



((1) (2 (3 . 4)))

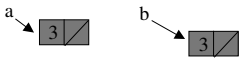
((1 . ( )) . (2 . (3 . 4)))

((1) 2 3 . 4)))



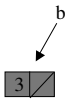
Pares (doted pairs)

- Ejemplos
  - > (define a (cons 3 ( )))
  - > (define b (cons 3 ( )))
  - > a
  - > (3)
  - > b
  - > (3)
  - (eq? a a)
  - #t
  - (eq? a b)
  - #f
  - (eq? (cons 1 2) (cons 1 2))
  - #f
  - (eq ’( ) ’( ))
  - #t



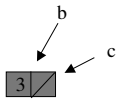
Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)



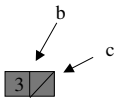
Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)
  - > (define c b)



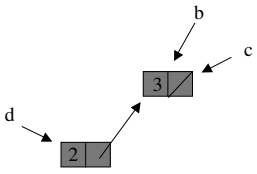
Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)
  - > (define c b)
  - > (eq? b c)
  - > #t



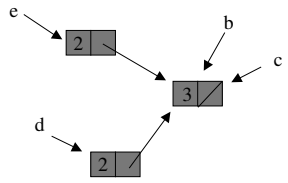
Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)
  - > (define c b)
  - > (eq? b c)
  - > #t
  - > (define d (cons 2 c))



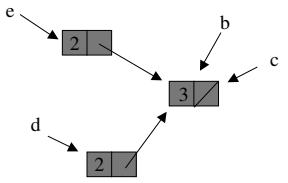
Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)
  - > (define c b)
  - > (eq? b c)
  - > #t
  - > (define d (cons 2 c))
  - > (define e (cons 2 c))



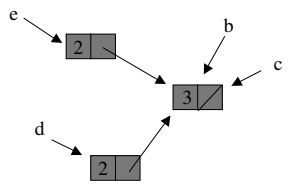
Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)
  - > (define c b)
  - > (eq? b c)
  - > #t
  - > (define d (cons 2 c))
  - > (define e (cons 2 c))
  - d
  - (2 3)
  - (eq? d e)
  - #f



Pares (compartiendo pares unteados)

- Ejemplos
  - > b
  - > (3)
  - > (define c b)
  - > (eq? b c)
  - > #t
  - > (define d (cons 2 c))
  - > (define e (cons 2 c))
  - d
  - (2 3)
  - (eq? d e)
  - #f
  - (eq? (cdr d) (cdr e))
  - #t



Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Vectores

- Motivación:
  - Las listas no proveen identificadores para sus elementos
  - List no define registros (records) con datos heterogéneos
  - List no define arreglos (arrays) con datos homogéneos
  - Acceso en registros y arreglos es random (el tiempo de acceso es igual para todos los elementos)
- Constructor: **vector** (listas o strings a vectores)
  - > (define v1 (vector 1 2 (+ 1 2))) > v2
  - > v1 > #(a b)
  - > #(1 2 3) > (vector v1 v2)
  - > (define v2 (quote #(a b))) > #(#(1 2 3) #(a b))

Vectores

- Operadores:
  - vector-length, vector-ref, vector->list, list->vector, make-cell (vec. un elem.)
- Predicados:
  - vector?, eq? (como en cons), cell?
- Ejemplos:
  - > (define v3 '#(first second last))
  - > (vector? v3)
  - > #t
  - > (vector-ref v3 0)
  - > first
  - > (vector-length v3)
  - > 3
  - > (vector->list v3)
  - > (first second last)
  - > (vector-ref v3 (- (vector-length v3) 1))
  - > last
  - > (vector-ref '#(another #((heterogeneous) "vector")) 1)
  - > #((heterogeneous) "vector")

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Procedimientos

- Distinción entre el nombre y el procedimiento
  - Predicador: **procedure?**
    - > (procedure? 'car)
    - > #f
    - > (procedure? car)
    - > #t
  - Los procedimientos pueden ser argumentos de otros procedimientos
    - > (procedure? (car (list cdr))
    - #t

Procedimientos

- Los procedimientos son datos
  - > (if (procedure? 3) car cdr))
  - > #procedure
  - > ((if (procedure? 3) car cdr) '(x y z))
  - > (y z)
  - > (((if (procedure? procedure?) cdr car)
    (cons car cdr))
    '(car in the car))
  - > (in the car) ; (cdr '(car in the car))
  - > (((if (procedure? procedure?) car cdr)
    (cons car cdr))
    '(x y z))
  - > x

Procedimientos (apply)

- Procedimiento normal: aplicación funcional
  - > (+ 1 2)
- **apply**: recibe un proc. y una lista de argumentos
  - > (apply + '(1 2))
  - > 3
  - > (define abc '(a b c))
  - > (apply cons (cdr abc))
  - > (b . c)
  - > (apply apply (list procedure? (list apply)))
  - > #t

Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.

Abstracción funcional (lambda)

- El abstractor funcional *lambda* p
- Permite crear nuevos proc. no necesariamente ligados a variables y probablemente anónimos
  - (lambda *lista-de-variables cuerpo-del-procedimiento*)
- Cuando el proc. se llama
  - (1) las variables en la lista se ligan con las var. en el proc.
  - (2) el procedimiento se evalúa
  - Las ligas son locales al procedimiento
  - ejemplo:
    - (lambda (n) (+ n 2))

Abstracción funcional (lambda)

- Uso de procedimientos creados dinámicamente
  - El proc. puede ejecutarse directamente
  - El proc. puede asociarse a un símbolo mediante **define**
- Ejemplos:

```
> ((lambda (n) (+ n 2) 4)
> 6
> (list (lambda (n) (+ n 2)) 6)
> (#<procedure> 6)
> (define add2 (lambda (n) (+ n 2))
> (add2 6)
> 8
```

```
> (define select
      (lambda (b 1st)
          (if b
              (car 1st)
              (cadr 1st))))
> (select #f  '(a b))
> b
> (select #t '(list cdr car)) '(a b c))
> (b c)
```


Procedimientos anónimos

- Se usan normalmente como argumentos (a diferencia de muchos lenguajes)
- Se utilizan con **map** y **andmap**
- Ejemplos:

```
> (map (lambda (n) (+ n 2)) '(1 2 3 4 5))
> (3 4 5 6 7)
> (defien add2 (lambda (n) (+ n 2))
> (map add2 '(1 2 3 4 5))
> (3 4 5 6 7)
> (andmap number? ' (1 2 3 4 5))
> #t
> (map null? '( (a) ( ) ( ) (3) ))
> (#f  #t  #t #f)
```


Procedimientos anónimos

- Se usan normalmente como argumentos (a diferencia de muchos lenguajes)
- Se utilizan con **map** y **andmap**
- Ejemplos:

```
> (map car '( (a b) (c d) (e f) ))
> (a c e)
> (map list '(a b c d))
> ( (a) (b) (c) (d))
> (map (lambda (f) (f '(a b c d)))   (list car cdr cadr cddr caddr))
> (a (b c d) b (c d) c)
```


Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Procedimientos de primer orden

- Un valor es de primer-orden si puede ser pasado y recibido por procedimientos, además de guardado en estructuras de datos.
  - En Lisp todos los valores son de primer-orden
- Ejemplo:
  - Si f y g son funciones tales que Rango(g) $\subseteq$ Dominio(f) definir la composición $(f \bullet g)(x) = f(g(x))$

```
> (define compose
      (lambda (f g)
          (lambda (x) (f (g x)))))
> (define add4 (compose add2 add2))
> (add4 5)
> 9
> (compose car cdr) '(a b c d))
b
```


Scheme: dialecto de Lisp

- Expresiones simples
  - Literales, llamadas a procedimientos y variables
  - Definiciones, programas y ciclo lee-evalúa-imprime
  - Evaluación condicional
- Tipos de datos
  - Números, boleanos, caracteres, cadenas y símbolos
  - Listas
  - Pares
  - Vectores
- Procedimientos
  - Expresiones *lambda*
  - Procedimientos de primer orden
  - Procedimientos con aridad variable

Procedimientos con aridad variable

- La *aridad* es el número de argumentos que tiene un proc.
  - Procedimientos de aridad variable: **list**, **vector**, **string**, **+**
  - Es posible definir nuevos procedimientos de aridad variable
    - (lambda x x) ;equivalente al proc. **list**
- Ejemplo:
  - > (define plus
    (lambda x
    (if (null? (cdr x))
    (lambda (y) (+ (car x) y))
    (apply + x))))
  - > (plus 1 2)
  - 3
  - >((plus 1) 2)
  - 3

Dr. Luis A. Pineda, IIMAS, UNAM, 2000.