Programación Funcional en LISP

Lógica para Ciencias de la Computación

Primer Cuatrimestre de 2019

– Material Adicional –

Introducción

- A lo largo de la carrera estudiaremos diversos paradigmas de programación:
  - Programación Orientada a Objetos.
  - Programación Lógica.
  - Programación Funcional.
- En esta clase, abordaremos uno de los lenguajes que pertenecen al paradigma funcional, el lenguaje LISP.

Origen

- LISP se originó en la década del 50, de la mano de John McCarthy.
- Se trata de un lenguaje maduro, aplicable a diversos escenarios, tales como:
  - Robótica.
  - Inteligencia Artificial.
  - Procesamiento de lenguaje natural.
  - Demostración automática de teoremas.

Principales Características

- La característica principal de LISP es que todo programa consiste de una función.
- Otras peculiaridades:
  - No posee asignaciones.
  - Su principal estructura de control es la recursión.
  - Los programas y los datos son equivalentes.
  - Su principal estructura de datos es la lista.

LISP vs. LISP Puro

- LISP incorpora ciertos vicios tomados del paradigma imperativo, que permiten expresar aspectos ajenos al paradigma funcional.
- El subconjunto de LISP netamente funcional se lo suele denominar LISP puro.
- En lo que resta, nos concentraremos exclusivamente en LISP puro.

Expresiones en LISP

- Todo programa es una s-expresión.
- Las s-expresiones pueden ser átomos o listas.
- Un átomo puede ser un átomo numérico o un átomo simbólico.
- Una lista es una secuencia de s-expresiones delimitadas por paréntesis, con sus elementos separados por espacios en blanco.

Representación de Funciones

- La aplicación de una función LISP se nota en forma prefija, mediante una lista:
  - La función **f** aplicada a los argumentos a_1, a_2, a_3, …, a_n, se denota $(f\ a_1\ a_2\ a_3\ \ldots\ a_n)$, en vez del usual $f(a_1, a_2, a_3, \ldots, a_n)$.
- Por ejemplo, la suma de 2 con el producto de 5 y -3 se denota como:

Evaluación de s-expresiones

- Como vimos antes, es posible expresar aplicaciones anidadas de funciones.
- Básicamente, el intérprete de LISP evalúa una s-expresión *S*, representando posiblemente aplicaciones anidadas de funciones, de la siguiente forma:
  - Si *S* es un átomo numérico *a*, entonces el resultado es simplemente *a*.
  - Si *S* es de la forma $(f\ a_1\ a_2\ a_3\ \ldots\ a_n)$, entonces el intérprete evalúa recursivamente cada argumento a_i, obteniendo valores $v_1, v_2, \ldots, v_n$ y luego calcula el resultado de aplicar *f* a dichos valores.

Funciones Aritméticas

- LISP cuenta, entre otras, con las siguientes funciones aritméticas predefinidas:
  - (+ Arg_1 Arg_2 … Arg_n)
  - (- Arg_1 Arg_2 … Arg_n)
  - (* Arg_1 Arg_2 … Arg_n)
  - (/ Arg_1 Arg_2 … Arg_n)
  - (1+ Arg)
  - (1- Arg)
  - (MAX Arg_1 Arg_2 … Arg_n)
  - (MIN Arg_1 Arg_2 … Arg_n)

QUOTE: “ ` ”

- Como dijimos anteriormente, cuando LISP evalúa una aplicación, evalúa recursivamente cada argumento de la misma.
- Existen situaciones en las que es necesario evitar la evaluación de ciertos argumentos en una aplicación.
- Por ejemplo, cuando un argumento es una lista que no denota la aplicación de una función sino simplemente una lista de elementos como estructura de datos.
- A tal efecto, LISP cuenta con la función QUOTE de aridad 1.

Función QUOTE: “ ` ”

- Sintaxis:
  - (QUOTE Arg)
  - o simplemente puede abreviarse ’Arg
- Esta función evita que se evalúe su argumento, y lo retorna intacto como resultado.

```
> (+ 1 2)

3

> (2 3 5)

ERROR: 2 no es una
función
```

```
> `(+ 1 2)

(+ 1 2)

> `(2 3 5)

(2 3 5)
```

(o (QUOTE (+ 1 2)))

Función EVAL

- EVAL tiene el efecto contrario que la función QUOTE.

- (EVAL Arg) fuerza una evaluación adicional de su argumento Arg.

- Ejemplo:

```
> (QUOTE (+ 2 2))
(+ 2 2)
```

```
> (EVAL (QUOTE (+ 2 2))
4
```

Funciones Elementales sobre Listas

- LISP dispone de las siguientes funciones para manipular listas:
  - (CAR List) retorna el primer elemento de la lista List.
  - (CDR List) retorna la cola de la lista List.
- Por ejemplo:

```
> (CAR `(1 2 3))
1
> (CDR `(1 2 3))
(2 3)
```

Constructores de Listas

- Como las listas desempeñan un papel central en LISP, existen múltiples formas de construir una lista:
  - (APPEND $List_1$ $List_2$ … $List_n$) Esta función concatena las listas $List_1$, $List_2$, …, $List_n$.
  - (CONS Elem List) Retorna una lista cuyo primer elemento es Elem y cuya cola es List.
  - (LIST $Elem_1$ $Elem_2$ … $Elem_n$) Esta función retorna una lista formada por los elementos $Elem_1$, $Elem_2$, …, $Elem_n$.

Átomos Especiales

- En LISP, los siguientes átomos presentan un comportamiento especial.
  - T (verdadero).
  - NIL (falso, indefinido, lista vacía).
  - Números
- Estos átomos son idempotentes con respecto a la evaluación.

La sobrecarga de significados de NIL constituye un aspecto cuestionable del lenguaje.

Predicados en LISP

- Un predicado puede representarse en LISP como una función que retorna un valor de verdad.
- Se adopta la convención de que NIL denota falso, y que cualquier otra s-expresión denota verdadero.
- Por ejemplo:
  - NIL equivale a falso.
  - T equivale a verdadero.
  - (+ 2 2) equivale a verdadero.

Predicados en LISP

- LISP dispone de diversos predicados predefinidos:
  - (ATOM Arg)
  - (EQUAL Arg_1 Arg_2)
  - (MEMBER Elem List)
  - (ZEROP Arg)
  - (NULL Arg)
  - (< Arg_1 Arg_2 … Arg_n)
  - (= Arg_1 Arg_2 … Arg_n)
  - (> Arg_1 Arg_2 … Arg_n)

Predicados Lógicos

- LISP también cuenta con los predicados lógicos tradicionales:
  - (NOT Arg) Retorna T si, y sólo si, la valuación de Arg es NIL. En caso contrario retorna NIL.
  - (AND Arg_1 Arg_2 … Arg_n) Evalúa sus argumentos de izquierda a derecha. Si encuentra alguno que evalúa a NIL, retorna NIL, pero si todos evalúan a algo distinto a NIL retorna el último valor obtenido.
  - (OR Arg_1 Arg_2 … Arg_n) Análoga a la anterior, sólo retorna NIL cuando todos sus argumentos evalúan a NIL, retornando en caso contrario el valor del primer argumento que evalúe a algo distinto a NIL.

Función Condicional

- Una de las funciones más importantes es la función condicional:

  (COND ($Test_1$ $Result_1$)
  ($Test_2$ $Result_2$)
  ...
  ($Test_n$ $Result_n$))

- Esta función evalúa las condiciones $Test_i$ en orden ascendente, y en caso de encontrar algún $Test_k$ que evalúe a un valor distinto de NIL, retorna la evaluación de $Result_k$.
- Si en cambio todas las condiciones evalúen a NIL, retornará NIL.
- Si las condiciones $Test_i$ son exhaustivas, entonces se suele reemplazar la última ($Test_n$) por T.

Definición de Funciones

- Como los datos y programas no difieren entre sí, incluso la definición de nuevas funciones es llevada a cabo mediante la invocación de una función:

(DEFUN FunctionName (Par_1 Par_2 … Par_n)
FunctionDescription)

- Esta expresión define una función de nombre FunctionName (átomo simbólico), con argumentos Par_1, Par_2, …, Par_n, cuyo comportamiento sigue la descripción FunctionDescription.

Definición de Funciones

- La función **DEFUN** presenta un comportamiento especial, a saber:
  - No evalúa sus argumentos.
  - Retorna como resultado el nombre de la función que acaba de ser definida.
- Ejemplo: definición de la función **suc**.

```
(DEFUN suc (N)
  (+ N 1)
)
```

Ejemplos

- Ejemplo: definición de la función factorial.

```
(DEFUN factorial (N)
        (COND
                ((ZEROP N)          1)
                (T          (*
                                N
                                (factorial (- N 1))
                            )
                )
        )
)
```

Ejemplos

- Ejemplo: definición de la función fibo.

```
(DEFUN fibo (N)
        (COND
                ((< N 2)   1)
                (T     (+
                        (fibo (- N 1))
                        (fibo (- N 2))
                       )
                )
        )
)
```

Ejemplos

- Ejemplo: definición de la función largo.

```
(DEFUN largo (L)
        (COND
                ((NULL L)  0)
                (T (+ 1 (largo (CDR L))))
        )
)
```

SETQ

- La función predefinida SETQ permite ligar un átomo simbólico a un valor.
- Su semántica es análoga a una asignación en el paradigma imperativo.
- SETQ no forma parte de LISP puro, ya que átomo simbólico queda ligado al valor luego de la aplicación de la función, lo que significa que presenta efectos colaterales.
- Además, la aplicación de SETQ retorna como resultado el valor asignado.

SETQ: ejemplos

```
> (SETQ x 10)
10
> x
10
> (* x 2)
20
> (SETQ x (+ x 5))
15
> (SETQ lista '(1 2 3))
(1 2 3)
```

LET

- SETQ liga un átomo a un valor, globalmente.
- La función LET permite ligar átomos simbólicos a valores localmente, es decir, evitando el efecto colateral del SETQ.
- Sintaxis:

  (LET ((Atom$_1$ Value$_1$) … (Atom$_n$ Value$_n$))
  Expression
)

  donde Expression puede hacer referencia a los Atom$_i$ ligados previamente.

LET: ejemplo

```
> (LET
      ((x 3) (y 2))
      (* x y)
  )
6
```

Intérprete de LISP

- Puede adoptar uno de los siguientes intérpretes de Common LISP:
  - GNU Common LISP.
  - CLISP.
- La función predefinida LOAD permite cargar un archivo con código lisp.
  - Modo de uso: (load <nom_archivo_entre_comillas>)
  - ejemplo: (load "c:/temp/pruebalisp.lisp")
- Debugger: funciones TRACE y STEP.
- Sitio de referencia sobre lisp en general: cliki.net