UNIVERSIDAD AUTONOMA DE GUADALAJARA

INCORPORADA A LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA DE INGENIERIA EN COMPUTACION

"SEMBLANZA DE LISP: LENGUAJE DE PROGRAMACION PARA EL DESARROLLO DE LA INTELIGENCIA ARTIFICIAL"

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A:

JOSE HUMBERTO SANDOVAL ROJO

GUADALAJARA, JAL. AGOSTO 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

Aunque son incesantes los progresos y los nuevos desarrollos de la Inteligencia Artificial, todavía no se alcanzan las facultades propias de un niño de 5 años. Es por esto que estamos muy a tiempo de alcanzar o incluso superar los sistemas hasta ahora desarrollados. Lamentablemente en nuestro país la información existente además de ser escasa, está poco condensada y no está en español.

Es ahí donde surge la necesidad de dar a conocer (evaluar) uno de los modelos computacionales que nos permiten desarrollar sistemas de I. A., dicho modelo es mejor conocido como LISP, un lenguaje de programación que toma su nombre de LISt Programing (programación en lista).

Lo anterior es esencial para poder entender la implementación de un programa completo dentro del campo de la Inteligencia Artificial (Entendimiento del Lenguaje Natural), que se diseñará con el efecto de servir de ejemplo práctico.

Se anticipa que, para entender algunos conceptos y estructuras empleadas aquí, se necesita tener conocimientos previos sobre programación, por lo que muchos terminos se dan por entendidos, omitiendo así su explicación detallada. Es por esto, que este trabajo va dirigido a personas con conocimientos en computación, teniendo como principal objetivo, el de servir como guia de referencia, porque el mejor método para aprender lisp, es através de la práctica, que con el paso del tiempo se traducirla como experiencia.

Para lograr tal objetivo, este trabajo se fragmento como a continuación se describe:

ANTECEDENTES. Se hizo una recopilación de información ilustrativa y breve, haciendo referencia a la manera como el lenguaje de programación LISP fue desarrollado, así como las personas que intervinieron en su implementación. Tambien se mencionan algunos de los diferentes dialectos que existen y sus diversos campos de aplicación.

CAPITULO I "CARACTERISTICAS DE LISP". Tomando en cuenta la gran variedad de tópicos relacionados con el lenguaje y haciendo una evaluación de acuerdo a una serie de criterios personales, se seleccionaron los sub-temas a tratar en este capítulo, donde se hace notar el porque LISP es usado como lenguaje de I. A., además de proveer la información basica necesaria para entenderlo y posteriormente poderlo programar.

CAPITULO II "FUNCIONES LISP". En este capítulo se definen los primitivos escritos en lenguaje máquina, esto es, se describen las principales funciones escritas en muLISP. Las funciones en que se subdivide este capítulo son tomadas de [Rich, A.] para diferenciar las funciones LISP.

Se realizó una interpretación de acuerdo a juicios muy particulares, para que cada función descrita se comprenda con rapidez y claridad [Charniack y Mcdermott], [Winston y Horn] y [Rich, A.].

Además se elaboró una traducción técnica, para adecuar los terminos del Ingles al Español sin distorcionar su significado.

CAPITULO III "APLICACION DE LISP (PROGRAMA)". En este capitulo se implemento (diseño) una aplicación practica de el lenguaje de progamación, en una de las ramas de la I. A. denominada Entendimiento del Lenguaje Natural que en este caso es el Español (utilizando la técnica de Emparejamiento Aproximado), explicando cada una de las funciones desarrolladas para integrar el programa, proporcionando la información necesaria para manejar el sistema.

CAPITULO IV "CONCLUSIONES Y RECOMENDACIONES". En este capitulo se externan las impresiones finales, asi como las ventajas y desventajas de la técnica empleada para desarrollar el programa, haciendo las aclaraciones pertinentes.

ANTECEDENTES

El concepto de procesamiento en lista fue desarrollado por Newel y Simon (Universidad de Carniage-Mellon de U.S.A.), creando el lenguaje IPL para la teoria logistica [Presser, Cárdenas y Marin].

En la conferencia Dartmounth sobre Inteligencia Artificial, cuyo organizador fue Jhon McCarthy, Newel y Simon demostraron las ventajas que ofrece el procesamiento en lista para el desarrollo de la Inteligencia Artificial. Fue entonces cuando McCarthy se propuso implementar un mejor lenguaje que utilizara dicho concepto. Intentó modelarlo en el recién desarrollado Fortran, en una IBM 704, haciendo cuatro funciones para recuperar el contenido de las localidades de memoria, representando a cada elemento mediante dos celdas contiguas; una de ellas es el elemento en cuestion (CAR) y la otra (CDR) es la dirección del siguiente elemento en la lista. Los otros dos comandos CPR y CTR ya no se utilizan.

McCarthy sostenia que podia ser escrita una función universal (EVAL), que interpretara y evaluara la forma de cualquier función lisp. Para entonces lisp todavia no existia, pues no tenia compilador, pero uno de sus estudiantes Steve Rusell, mezcló la teoria con la practica y propuso la forma de correr el lenguaje [Charniack y McDermott].

En si, el lenguaje de programación lisp, se basa en el documento de Jhon McCarthy [1960] titulado: " Funciones recursivas de expresiones simbólicas y su computación por

máquina ". En donde además de exhortar las ventajas que ofrecía lisp para propositos matemáticos, escribió la forma de usar una expresión computacional, en teoría de automatas, que iba más allá, de la entonces tradicional "Prueba de Turing" (temprana evocación a la Inteligencia Artificial, en la que la computadora, trataba de simular a el hombre). Eval fué escrito y LISP nació.

Lisp tiene muchos derivados (diferentes dialectos, con sus caracteristicas muy particulares, pero siguiendo la teoría de lisp); Maclisp, Interlisp, Common Lisp, muLISP, entre otros, los tres primeros son dialectos desarrollados para sistemas grandes (main frames), el último fue desarrollado para microcomputadoras y es el dialecto que se tomará como referencia.

De cualquier forma, el lenguaje y sus muchos derivados continuan acaparando los trabajos serios sobre la Inteligencia Artificial y sus diversos campos de acción como lo son: la robotica, demostración de teoremas, entendimiento de lenguaje natural, teoría de juegos, sistemas expertos y visión.

CAPITULO I

CARACTERISTICAS DE LISP

1.1 LISP

Los programas sobre Inteligencia Artificial son particularmente exigentes y extremadamente variables, esto es porque los datos necesitan un area de almacenaje que varie en extensión, por la adición ó eliminación de datos, durante el tiempo de la ejecución [Presser, Cárdenas y Marin].

Lisp proporciona el medio adecuado para desarrollar programas en el extenso campo de acción de la Inteligencia Artificial.

Una de las ventajas que ofrece el procesamiento en lista, es que los datos se almacenan como elementos dentro de las listas, mismas que pueden sufrir modificaciones durante el computo. Como los elementos de la listas estan encadenados por apuntadores, no hay problemas para formar listas de asociación, o sea, listas de elementos que representan los atributos o propiedades de algun objeto determinado.

En realidad, son muchas las razones por las cuales lisp se utiliza como desarrollo de proyectos de Inteligencia Artificial. A continuación se presentan algunas de ellas:

a).- Es mucho más flexible que los demás lenguajes de programación, al permitirnos modificar el método para definir los programas.

b).- Su estructura de datos principal es la de árboles

binarios, lo cual nos permite la representación de datos-objetos, caracteristicos de los problemas de I. A.

c).- *Por su recursividad, nos permite definir y desarrollar complejos conceptos matematicos.*

d).- *Proporciona un alto grado de interactividad con el usuario, situación esencial para la comunicacion hombre-maquina.*

e).- *La asignación dinamica y el cicleo de los recursos de almacenamiento de datos, mejor conocida como: "Garbage Collection" (colección automatica de desperdicios), nos permite responder a preguntas con dificultad arbitraria, necesarias al tratar de simular inteligencia.*

1.2 ATOMOS Y LISTAS

La manipulación de símbolos es una herramienta esencial para hacer inteligentes a las computadoras. Entendiendo que la manipulación de símbolos se refiere a el hecho de trabajar con palabras y oraciones.

En lisp, los objetos fundamentales formados por bits son llamados átomos, hay dos tipos de átomos: los que no se pueden separar como 22 o 7.12, son llamados átomos numéricos ó números, el otro tipo, es una hilera de letras y números, que comienzan con una letra y no tienen espacios en blanco como clase, pedro o A23, estos son llamados átomos simbólicos ó símbolos [Winston y Horn].

Grupos de átomos forman las llamadas listas, que pueden ser agrupadas para formar listas de alto nivel (listas de listas). Una lista consiste de un paréntesis izquierdo seguido por cero ó mas átomos ó listas y terminada en un paréntesis derecho. A cada miembro de la lista (átomo ó lista), es llamado elemento, por ejemplo en la siguiente lista hay tres elementos: ((AB) M (OPK)), de los cuales uno es un átomo y dos son listas.

Atomos y listas son llamadas expresiones simbólicas (en algunos textos aparece como s-expression).

Los terminos expresión y forma pueden ser aplicados a la misma entidad, dependiendo del contexto. Si la lista es cosiderada como dato puede ser llamada expresión; pero si es considerada como parte de un procedimiento, la misma lista puede ser considerada una forma.

En lisp, la lista vacia es una construcción común, y tiene un nombre especial, NIL ( ). Unicamente la lista vacia puede ser considerada como un átomo ó como una lista [Charniack y McDermott].

El procedimiento siempre es especificado al principio, seguido por los elementos con que se va a trabajar, llamados argumentos. La multiplicación de 7 por 8 es representada como: (TIMES 7 8), en este caso hay tres elementos, el primero es el procedimiento, seguido por dos argumentos. Esta notación de prefijo facilita el trabajo, porque el nombre del procedimiento siempre va en el mismo lugar, no importando cuantos argumentos esten envueltos.

Un procedimiento es la unidad basica en lisp, que especifica como se van a hacer las cosas. El procedimiento que regresa un valor basado solamente en los argumentos es llamado: FUNCION.

La función puede ser definida por el usuario ó provelda por el sistema, estas últimas son llamadas primitivos. Y a la colección de funciones que trabajan juntas, es a lo que se le denomina programa.

1.3 VARIABLES

Las variables son definidas bajo la función en donde son declaradas, ó en cualquier función que llame a otra función. El alcance de una variable es determinado dinamicamente por quienes la llaman durante el computo.

En una función, una variable es llamada local, si es declarada en la misma función donde es usada. Si no es local, se dice que es una variable libre o especial, pero de esta manera se dificulta el entendimiento del programa.

Si la variable es definida al principio de lisp, se le conoce como variable global (porque esta definida en todas partes) [Winston y Horn].

Cuando se utiliza el primitivo SETQ se declaran implicitamente las variables globales, en este caso, la asignación sigue siendo igual despues que la funcin es exitada, a este fenomeno se le conoce como efecto de lado.

1.4 DEFINIENDO PROGRAMAS

En lisp hay muchos caminos para completar un programa, por lo que la manera o el camino exacto para expresar una función, varia de persona a persona. El conjunto de todas estas desiciones define un estilo propio de programación.

La definición de los programas depende del estilo de programación del usuario. Una función se puede definir con el siguiente formato:

```
(PUTD (NOMBRE DE LA FUNCION)(LAMBDA((PARAMETROS))
     ((CUERPO DE LA FUNCION)) ))
```

Por ejemplo, definamos la suma de dos numeros como sigue:

```
(PUTD 'SUMA '(LAMBDA (A B)                    ;DEFINIENDO LA FUNCION
     ((AND (NUMBERP A)(NUMBERP B))            ;A y B SON NUMEROS ?
          (PLUS A B)) ))                      ;EFECTUA LA SUMA
```

Cuyos primitivos son explicados en detalle en el siguiente capitulo. Cabe hacer notar que hay dos tipos de comentarios, pueden ser de una sola linea: comenzando con un punto y coma ";" y terminando al final de la linea (como en el ejemplo anterior), ó pueden ser multilinea: son delimitados por los signos de porcientos.

El signo "'", como el que aparese antes de la palabra suma es una abreviación de una función llamada QUOTE (función que se utiliza para no evaluar el argumento).

La expresión LAMBDA ayuda como interface entre procedimientos y listas de argumentos. El ejemplo anterior muestra como la función LAMBDA puede ser definida y usada, dicha

expresión puede tener cualquier número de argumentos.

Si una función se define con LAMBDA es llamada función EVAL.

1.5 PROPIEDADES

En lisp, cada simbolo puede tener propiedades asociadas con el. Con dichas propiedades podemos darle atributos y valores a los simbolos.

Una lista de asociación es una lista que incluye sublistas, en la que el primer elemento de cada sublista es una llave (nombre de la propiedad). El usuario derfinirá los nombres de las propiedades y sus valores segun su conveniencia. Por ejemplo, las propiedades y valores de un carro podrian ser:

((COLOR AZUL) (MARCA FORD) (MOTOR V8) (MODELO 1970))

Las funciones para agregar, suprimir ó extraer las propiedades de algun simbolo son desarrolladas en la sección 2.6.

En la representación interna de los elementos en lisp, la segunda celda de un nombre apunta a una lista de propiedades conteniendo propiedades y banderas.

Esto nos da un medio propicio para construir bases de datos extendibles, de las cuales la información puede ser facilmente y rapidamente extraida.

1.6 RECURSION E ITERACION

La programación requiere seleccionar una estructura de control, dicha estructura, es en general un esquema por el cual una función puede ir desarrollando sus expresiones.

La recursión y la iteración son ejemplos de estructuras de control.

Recursión es cuando la función se llama a si misma, directamente ó atraves de un intermediario. Cualquier función se puede llamar asi misma, para ilustrar lo anterior definiremos la función factorial:

```
(PUTD 'FACTORIAL '(LAMBDA (J)                    ;DEF. FACTORIAL
    ((ZEROP J) 1)                                ;N=0 ?
    (TIMES J (FACTORIAL (DIFFERENCE J 1))) ));RECURSION
```

Como se puede apreciar, la función factorial se llama asi misma despues de haber efectuado la multiplicación de NxN-1 hasta que N toma el valor de cero.

Iteración es hacer que una función se repita una ó mas veces hasta que la condición de parada es satisfecha. Para notar la diferencia definamos una vez mas la función factorial:

```
(PUTD 'FACTORIAL '(LAMBDA (J K)                  ;DEF. FACTORIAL
    (SETQ K 1)                                   ;K=1
    (LOOP
        ((ZEROP J) K)                            ;J=0 ?
        (SETQ K (TIMES K J)                      ;K=KxJ
        (SETQ J (DIFFERENCE J 1))) )))           ;N=N-1
```

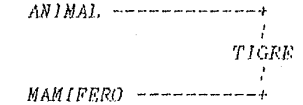
Como se puede apreciar esta definición de la función

factorial no es recursiva. El cuerpo del LOOP se repetirá hasta que la condicion N=0 sea cumplida, una vez que ocurra esto, el resultado quedará almacenado en "K".

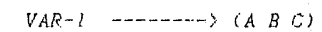
La elección de alguna estructura de control es determinada por el usuario, dependiendo de las caracteristicas del problema en cuestión.

1.7 APUNTADORES Y NOTACION DE CELDA

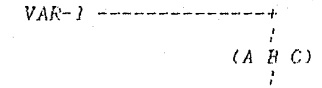
Cuando dos variables tienen el mismo valor y este valor es un atomo, las dos variables apuntan a la misma localidad de memoria (donde está el simbolo), y no se crea otra versión, pues los átomos en lisp son únicos (Charniack y McDermott).



Pero esto no ocurre con las listas; dos listas pueden ser iguales, pero cada uno puede apuntar a diferentes piezas de memoria, por ejemplo:



Es diferente a:

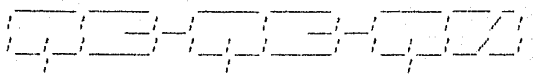


En la notación " caja-flecha ", los apuntadores son indicados por una flecha y los rectangulos son llamados celdas CONS (celda de memoria con dos apuntadores). El apuntador derecho, apunta a el elemento en cuestion (CAR) y el apuntador izquierdo apunta a el siguiente elemento (CDR).

Una lista puede ser representada por un conjunto de celdas de memoria que se enlazan por apuntadores, en la cual cada elemento apunta al siguiente de la lista, excepto el último, que apunta a NIL (representado por un linea diago-

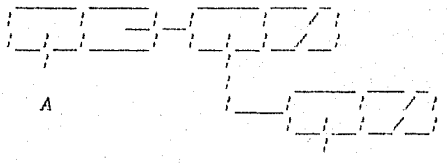
nal).

(A B C)



A B C

(A (B))

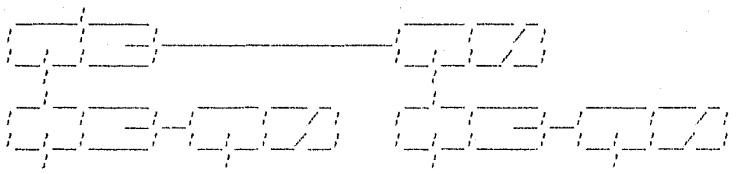


A

B

(EJEMPLO) ----> ((LAS COMPUTADORAS)(SON EFICIENTES))

EJEMPLO



LAS COMPUTADORAS SON EFICIENTES

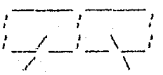
Como se puede apreciar en los dos ultimos ejemplos, los elementos de una lista pueden ser otras listas.

Lisp mantiene una lista de celdas libres de reserva para construir y modificar las estructuras de las listas.

Unas lista ordinaria es terminada por NIL, una lista terminada por un atomo que no es NIL se le conoce como notación de pares punteados (dotted pairs), y son escritos con un

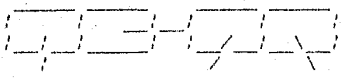
punto en medio de los dos elementos. Con este tipo de notación pueden crearse estructuras mas complejas, a la vez que toma menos espacio de memoria util que el otro tipo de notación.

(1.2)



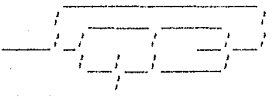
1 2

(A B.C)



A B C

Hay un tipo especial de lista llamada lista circular, en este tipo de lista el apuntador que se dirige a CDR, apunta a si mismo, lo que da como resultado un loop infinito.



1

En los ejemplos, la parte izquierda de la flecha es la forma como se emplea la función y la parte derecha es su resultado [Rich, A.].

2.1 FUNCIONES DE ASIGNACION

El proceso para establecer un valor a un simbolo es llamado asignación.

Estas funciones se utilizan para asignar valores a las variables del programa.

SET [NOMBRE,OBJETO]

Reemplaza el valor (celda CAR) del <NOMBRE> con un apuntador a el <OBJETO> y regresa el <OBJETO>.

(SET 'REX 'PERRO) ---> PERRO

(SET 'REX 'BOBBY) ---> BOBBY

PERRO ---> BOBBY

REX ---> PERRO

SETQ [NOMBRE,OBJETO]

Reemplaza el valor (celda CAR) de <NOMBRE> con un apuntador a el <OBJETO> y regresa <OBJETO>, el primer argumento a una llamada a SETQ no se evalua.

(SETQ ALFA '(A B C D)) ---> A B C D

```
ALFA                          ---> A B C D
```

POP [NOMBRE]

Regresa la cima del "stack" llamado <NOMBRE> y actualiza el valor de <NOMBRE>.

```
(SETQ LISTA-SATCK '(R S T W X Y)) ---> (R S T W X Y)
(POP LISTA-STACK)                 ---> R
(POP LISTA-SATCK)                 ---> S
(POP LISTA-STACK)                 ---> T
LISTA-SATCK                       ---> (W X Y)
```

PUSH [OBJETO,NOMBRE]

Empuja <OBJETO> a la cima del "stack" llamado <NOMBRE> y actualiza el valor de <NOMBRE>.

```
(SETQ LISTA-STACK NIL) ---> NIL
(PUSH 'A LISTA-STACK)  ---> (A)
(PUSH 'B LISTA-STACK)  ---> (B A)
(PUSH 'C LISTA-STACK)  ---> (C B A)
LISTA-STACK            ---> (C B A)
```

2.2 FUNCIONES DE SELECCION

Los árboles binarios están hechos de ramas (nodos) y terminados en hojas (átomos).

CAR regresa la rama derecha del árbol, mientras que CDR regresa la rama izquierda. Para recorrer un árbol binario, solo se necesita aplicar sucesivamente estas dos funciones.

CAR [LISTA]

Regresa el primer elemento de la lista dada como argumento, donde la <LISTA> puede ser de uno ó más elementos.

(CAR '(M N O P Q R)) ---> M

(CAR 'COMPUTADORA) ---> COMPUTADORA

(CAR '(CARROS ESTANDAR)) ---> CARROS

CDR [LISTA]

Es el complemento de CAR, regresa todos los elementos de la <LISTA> excepto el primero. Cuando CDR es aplicado a una lista de un solo elemento, regresa una lista vacía, NIL.

(CDR '(J K L M N)) ---> (K L M N)

(CDR '((A B) C)) ---> C

(CDR '(EL ESCRITORIO AMPLIO))--> (ESCRITORIO AMPLIO)

CXXXR [LISTA]

Cuando se nesecitan muchos CDR's y CAR's, se pueden abreviar con el formato CXXXR, donde cada X puede ser una "A" representando a CAR o una "R" representando a CDR. Por ejemplo: (CAR (CDR [LISTA])) = (CADR [LISTA])

(CDDR '(A B C D E)) ---> (C D E)

(CADDR '(A B C D E)) ---> C

(CAADR '(LOS CARROS GRANDES))---> C

LAST [LISTA]

Regresa una lista que contiene solamente el último elemento de la lista dada como argumento.

(LAST '(W X Y Z)) ---> Z

(LAST '((A B)(C D))) ---> (C D)

NTH [n,LISTA]

Regresa el <n> ésimo elemento de la <LISTA> donde el CAR de la lista es el elemento dos. NIL es regresado si <LISTA> es mas chica que <n>.

(NTH 3 '(A B C D)) ---> D

(NTH 2 '(L M N O)) ---> M

NTH [LISTA,n]

Regresa la cola de la <LISTA> empezando con el <n> ésimo elemento, donde el CAR de la lista es el primer elemento.

(NTH '(L M N O P) 4) ---> (O P)

(NTH '(A B C D) 6) ---> NIL

ASSOC [LLAVE,UNA-LISTA]

Desarrolla una busqueda lineal de la lista de asociación <UNA-LISTA>, buscando un elemento no atómico donde la celda CAR es igual a la <LLAVE>. En este caso se utiliza la

notación de pares punteados.

(SETQ TABIQUE (FORMA.CUADRADA)(COLOR.BLANCO))

(ASSOC 'COLOR TABIQUE) ---> (COLOR.BLANCO)

(ASSOC 'FORMA TABIQUE) ---> (FORMA.CUADRADA)

2.3 FUNCIONES DE COMPARACION

Un predicado es una función que regresa un valor que puede ser : verdadero (indicado por "T") ó falso (indicado por NIL).

Las funciones de comparación requieren de dos argumentos y excepto MEMBER, regresan T ó NIL.

EQ [OBJETO1,OBJETO2]

Regresa "T" si el <OBJETO1> es idéntico a el <OBJETO2> (entendiendo por idéntico a que ocupan la misma localidad de memoria), si no regresa NIL.

```
(SETQ ALFA '(A B C))      ---> (A B C)
(EQ ALFA '(A B C))        ---> NIL
(SETQ LETRAS ALFA)        ---> (A B C)
(EQ ALFA LETRAS)          ---> T
```

EQUAL [OBJETO1,OBJETO2]

Regresa "T" si <OBJETO1> es igual a <OBJETO2> (entendiendo por iguales que tengan los mismos elementos), si no regresa NIL.

```
(EQUAL '(Z X Y)'(Z X Y))   ---> T
(EQUAL '(A B C)'(C B A))   ---> NIL
```

MEMBER [OBJETO,LISTA]

Prueba si un átomo llamado <OBJETO> es un elemento de una <LISTA>, regresando la cola de la lista, empezando con el primer elemento encontrado.

(MEMBER 'PERRO '(GATO PERRO GALLO)) ---> (PERRO GALLO)

(MEMBER 'M '(K L M N)) ---> (M N)

GREATERP [N1,N2,....,Nm]

Regresa "T" si N1>N2>....>Nm, si no regresa NIL. Los argumentos tienen que ser numéricos.

(GREATERP 12 7 3 1) ---> T

(GREATERP 12 7 9 1) ---> NIL

LESSP [N1,N2,.....,NM]

Regresa "T" si N1<N2<....<Nm, si no regresa NIL.

(LESSP 1 2 3 4) ---> T

(LESSP 1 -3 5) ---> NIL

ORDERP [ATOMO1,ATOMO2]

Regresa "T" si <ATOMO1> es ordenado antes de <ATOMO2>, si no regresa NIL. Los nombres son ordenados de acuerdo a su introducción.

(ORDERP 'GALLO 'LOBO) ---> T

(ORDERP 'LOBO 'GALLO) ---> NIL

2.4 FUNCIONES DE CONSTRUCCION

Utilizando las funciones de construccion los programas pueden generar dinamicamente nuevas estructuras de datos, que se neseciten para la representación del problema.

CONS [OBJETO1,OBJETO2]

Toma una lista llamada <OBJETO2> y le inserta un nuevo primer elemento llamado <OBJETO1>, creando asi una nueva lista o nodo. CONS es un nemonico de CONStructor. Si la lista <OBJETO2> es un solo elemento se forma un par punteado.

(CONS 'A '(B C)) ---> (A B C)

(CONS 'J 'K) ---> (J.K)

LIST [OBJETO1,OBJETO2,...,OBJETOn]

CADA <OBJETO> se convierte en un elemento de la nueva lista, por lo que se pueden hacer listas de listas.

(LIST 1 2 3) ---> (1 2 3)

(LIST '(A B) '(C D))--> ((A B)(C D))

APPEND [LISTA1,LISTA2,....,LISTAn]

Regresa una lista con la agrupación de las <LISTAS> de la 1 a la n.

(APPEND '(A B) '(C D)) ---> (A B C D)

(SETQ ALFA '(J K L)) ---> (J K L)

(APPEND ALFA '(M N O)) ---> (J K L M N O)

ALFA ---> (J K L)

REVERSE [LISTA,OBJETO]

Regresa el elemento de la <LISTA> en orden inverso, sumado a <OBJETO>.

(REVERSE '(A B) 'C) ---> (B A.C)

(REVERSE '(W X Y Z)) ---> (Z Y X W)

2.5 FUNCIONES DE RECONOCIMIENTO

Las funciones de reconocimiento son predicados para identificar datos-objetos en lisp. Toman un simple argumento y regresan un valor "T" ó NIL.

NAME [OBJETO]

Regresa "T" si <OBJETO> es un nombre.

(NAME '(J K L)) ---> NIL

(NAME 'ZORRO) ---> T

NUMBERP [OBJETO]

Regresa "T" si <OBJETO> es un número.

(NUMBERP 3.1416) ---> T

(NUMBERP 200) ---> T

ATOM [OBJETO]

Prueba si <OBJETO> es un átomo.

(ATOM 100) ---> T

(ATOM 'JUAN)---> T

NULL [OBJETO]

Regresa "T" si <OBJETO> es una lista vacia (NIL).

(NULL '(A B C)) ---> NIL

(NULL NIL) ---> T

(NULL ()) ---> T

NONULL [OBJETO]

Regresa NIL si <OBJETO> es una lista vacia (), si no regresa el <OBJETO>.

(NONULL '(C D E)) ---> (C D E)

(NONULL NIL) ---> NIL

(NONULL 'GATO) ---> GATO

PLUSP [OBJETO]

Prueba si el <OBJETO> es un número positivo.

(PLUSP 343) ---> T

(PLUSP 0) ---> NIL

MINUSP [OBJETO]

Regresa "T" si <OBJETO> es un número negativo.

(MINUSP 0) ---> NIL

(MINUSP -3) --> T

ZEROP [OBJETO]

Regresa "T" si <OBJETO> es el número cero.

(ZEROP 0) ---> T

(ZEROP 5) ---> NIL

EVENP [OBJETO]

Prueba si el <OBJETO> es un número entero.

(EVENP 10) ---> T

(EVENP 0) ---> T

(EVENP -7) ---> NIL

2.6 FUNCIONES LOGICAS

Estas funciones nos permiten combinaciones booleanas de valores de verdad, esto es, falso (NIL) ó verdadero (T).

NOT [OBJET]

Regresa "T" si el <OBJETO> es NIL. Esta función es idéntica a la función NULL. Para un programa más legible, se utiliza NOT cuando se prueban valores de verdad y NULL cuando provamos una lista vacia.

(NOT '()) ---> T

(NOT NIL) ---> T

AND [OBJETO1,OBJETO2,....,OBJETOn]

Regresa NIL si cualquier argumento <OBJETOS> es NIL, en caso contrario regresa el <OBJETOn>.

(SETQ ALFA '(A B C)) ---> (A B C)

(AND T ALFA) ---> (A B C)

OR [OBJETO1,OBJETO2,....,OBJETOn]

Regresa NIL si todos los argumentos son NIL, si no regresa el primer argumento que evalua un valor que no es NIL.

(SETQ ANIMAL '(PERRO GATO)) ---> (PERRO GATO)

(OR (MEMBER 'ZORRO ANIMAL)(MEMBER 'GALLO ANIMAL)) ---> NIL

2.7 FUNCIONES DE PROPIEDADES

Se utilizan para manejar propiedades asociadas con nombres.

La celda CDR de un nombre apunta a una lista de propiedades conteniendo propiedades y banderas.

Las propiedades y la funciones banderas facilitan la construcción de bases de datos extendibles, de las cuales la información es facilmente extraida.

PUT [NOMBRE,LLAVE,OBJETO]

Coloca en la lista de propiedades llamada <NOMBRE>, bajo la indicación de la <LLAVE> el valor <OBJETO>.

(PUT 'LEON 'ANIMAL 'MAMIFERO) ---> MAMIFERO

(PUT 'ITALIA 'CAPITAL 'ROMA) ---> ROMA

GET [NOMBRE,LLAVE,OBJETO]

Regresa el valor de la propiedad asociada con <NOMBRE>, bajo la indicación de la <LLAVE>.

(GET 'ITALIA 'CAPITAL) ---> ROMA

(GET 'LEON 'ANIMAL) ---> MAMIFERO

REMPROP [NOMBRE,LLAVE]

Remueve de la lista de propiedades llamada <NOMBRE>, la propiedad asociada con la <LLAVE>. Regresando el viejo valor de la propiedad.

(REMPROP 'ITALIA 'CAPITAL) ---> ROMA

(GET 'ITALIA 'CAPITAL) ---> NIL

2.8 FUNCIONES DE MODIFICACION

Las funciones de modificación re-dirigen los apuntadores en la estructura de datos, por lo que alteran destructivamente el contenido de las celdas de memoria.

Son usadas por sus efectos, más que por el valor que regresan.

RPLACA [OBJETO1,OBJETO2]

(Replace CAR). Toma dos argumentos el primero de los cuales <OBJETO1> puede ser una lista, cuyo primer elemento (CAR) es sustituido con el valor de <OBJETO2>.

(SETQ VOL '(A B C)) ---> (A B C)

(RPLACA VOL 'M) ---> (M B C)

RPLACD [OBJETO1,OBJETO2]

(Replace CDR). Toma dos argumentos en donde <OBJETO2> sustituye el CDR de <OBJETO1>.

(SETQ V1 'AMISTAD ES VIRTUD) ---> (AMISTAD ES VIRTUD)

(REPLACD V1 '(DIVINO TESORO))---> (AMISTAD DIVINO TESORO)

NCONC [LISTA1,LISTA2,...,LISTAn]

Altera el valor de cualquier variable, cuyo valor es representado por un apuntador en la primera lista. NCONC como APPEND pueden regresar listas iguales solo que NCONC puede alterar las primeras listas.

(SETQ ALFA '(J K L)) ---> (J K L)

(NCONC ALFA '(M N O))---> (J K L M N O)

ALFA ---> *(J K L M N O)*

TCONC [APUNTADOR,OBJETO]

Suma el <OBJETO> a el final de la lista apuntandolo por el CAR del <APUNTADOR> par modificación de la lista. De esta manera se hace una eficiente suma de elementos a la cola de la lista.

```
(SETQ ALFA 'NIL)
(TCONC ALFA 'A)
(TCONC ALFA 'B)
(TCONC ALFA 'C)
(CAR ALFA) ---> (A B C)
```

LCONC [APUNTADOR,LISTA]

Suma <LISTA> a el final de la lista, apuntado por el CAR del <APUNTADOR> para modificación de la lista.

```
(SETQ ALFA 'NIL)
(LCONC ALFA '(A B))
(LCONC ALFA '(C D))
(LCONC ALFA '(E F))
(CAR ALFA) ---> (A B C D E F)
```

2.9 FUNCIONES STRING

Son principalmente usadas para textos y aplicaciones de procesamiento de lenguaje.

SUBSTRING [ATOMO,n,m]

Regresa del <n> ésimo caracter hasta el caracter <m> del string del <ATOMO>.

(SUBSTRING 'A B C D E F G 2 5) ---> B C D E

(SUBSTRING 'A B C D E F G 3) ---> C D E F G

(SUBSTRING 1000 1) ---> "1000"

FINDSTRING [ATOMO1,ATOMO2,n]

Regresa la posición de la primera ocurrencia del <ATOMO1> en el <ATOMO2>, después de saltar <n> caracteres. <n> es opcional, por default es cero.

(FINDSTRING 'X Y Z 'A B C X Y Z D E F X Y Z 5) ---> 10

(FINDSTRING 'X Y Z 'A B C X Y Z D E F X Y Z) ---> 4

PACK [LISTA]

Es una función que toma una <LISTA> y regresa un átomo.

(PACK '(4 B 6)) ---> "4B6"

(PACK '(MONO)) ---> MONO

UNPACK [ATOMO]

Esta función recibe como argumento un <ATOMO> y regresa una lista de caracteres.

(UNPACK 'ABCD) ---> (A B C D)

(SETQ NUM 312) ---> 312

(UNPACK NUM) ---> ("3" "1" "2")

LENGTH [OBJETO]

Regresa la longitud del <OBJETO>.

(LENGTH -430) ---> 4

(LENGTH 'MAMUT) ---> 5

(LENGTH '(R S T U V))--> 5

ASCII [ATOMO]

Regresa el equivalente ASCII del <ATOMO>.

(ASCII 'A) ---> 65

(ASCII 65) ---> A

2.10 FUNCIONES BANDERAS

Como las funciones de propiedad pueden ser usadas para almacenar información en la lista de propiedades de un nombre.

FLAG [NOMBRE,ATRIBUTO]

Abandera <NOMBRE> con el <ATRIBUTO>, haciendo <ATRIBUTO> el primer elemento de la lista de propiedades llamada <NOMBRE>. <ATRIBUTO> es sumado solo si no esta presente.

FLAGP [NOMBRE,ATRIBUTO]

Regresa un valor que no es NIL, si el <ATRIBUTO> es un miembro de la lista de propiedades llamada <NOMBRE>.

(FLAGP 'PEDRO 'HOMBRE) ---> (HOMBRE)

(FLAGP 'MARIA 'HOMBRE) ---> NIL

REMFLAG [NOMBRE,ATRIBUTO]

Remueve el <ATRIBUTO> de la lista de propiedades llamada <NOMBRE> y regresa "T". Si el <ATRIBUTO> no es encontrado regresa NIL.

(REMFLAG 'PEDRO 'HOMBRE) ---> T

(FLAGP 'PEDRO 'HOMBRE) ---> NIL

2.11 FUNCIONES NUMERICAS

Se utilizan para implementar las operaciones matemáticas, usando exactitud, presición infinita y enteros aritmeticos.

MINUS [n]

Regresa menos <n>.

(MINUS -7) ---> 7

(MINUS 3) ---> -3

PLUS [N1,N2,...,Nm]

Regresa la suma de <N1> hasta <Nm>.

(PLUS 3 43 4) ---> 50

(PLUS -12 6) ---> -6

DIFFERENCE [N1,N2,....,Nm]

Regresa la diferencia entre <N1> y la suma de <N2> hasta <Nm>. Si la diferencia esta dada como un simple argumento, regresa menos el argumento.

(DIFFERENCE 15 5) ---> 5

(DIFFERENCE 7 2) ---> 5

TIME [N1,N2,....,Nm]

Regresa el producto de <N1> hasta <Nm>.

(TIMES 3 3 2) ---> 18

(TIMES 4 10) ---> 40

QUOTIENT [n,m]

Regresa el entero truncado "QUOTIENT" de <n> dividido por <m>. Si <n> (el dividendo) es negativo, y -<n> no es divisible por <m> (divisor), el valor absoluto truncado es incrementado en orden para preservar la identidad fundamental de la división.

(QUOTIENT 7 2) ---> 3

(QUOTIENT -7 2)---> -4

(QUOTIENT -7 -2)--> 4

REMAINDER [n,m]

Regresa el residuo de <n> dividido por <m>. Esta definición conserva la relación fundamental de la división:

n = mQUOTIENT [n,m]+REMAINDER [n,m]

(REMAINDER 7 2) ---> 1

(REMAINDER -7 2)---> 2

(REMAINDER -7 -2)--> 2

DIVIDE [n,m]

Regresa un nodo donde la celda CAR apunta a QUOTIENT [n,m] y CDR apunta a el REMAINDER [n,m].

(DIVIDE 7 2) ---> (3.1)

(DIVIDE -7 2)---> (-4.2)

GCD [N1,N2,....,Nm]

Regresa el máximo común divisor de los argumentos enteros.

(GCD 8 20 12) ---> 4

2.12 FUNCIONES DE DEFINICION

Significan el acceso del nombre que se le da a la definición de función.

PUTD [NOMBRE,DEFINICION]

Modifica la celda de definición llamada <NOMBRE> apuntando a la compilación de la <DEFINICION>.

Se utiliza para encadenar lo definido por el usuario.

```
(PUTD 'CUADRADO '(LAMBDA (N)    ;DEF. EL CUADRADO DE 2 #
     (TIMES N N) ))             ;MULTIPLICA NxN
```

MOVD [NOMBRE1,NOMBRE2]

Mueve la función de definición llamada <NOMBRE1> a <NOMBRE2> y regresa <NOMBRE2>.

```
(MOVD 'CUADRADO 'CUAD)
(CUAD 5)  ---> 25
```

2.13 FUNCIONES DE EVALUACION

Son utilizadas por las expresiones y funciones para la evaluación del cuerpo del programa.

QUOTE [OBJETO]

Es una función que regresa los argumentos <OBJETO> sin evaluarlos, esta función se puede abreviar por medio del caracter "'".

```
(SETQ ALFA 320) ---> 320
ALFA            ---> 320
'(ALFA)         ---> ALFA
QUOTE (ALFA)    ---> ALFA
```

APPLY [FUNCION,LISTA-DE-ARGUMENTOS]

Aplica la <FUNCION> a la <LISTA-DE-ARGUMENTOS>, esto es, hace que la función evalue cada uno de los elementos de la lista.

```
(SETQ M1 '(3 2 5)) ---> 3 2 5
(APPLY 'PLUS M1)   ---> 10
```

LOOP [TAREA1,TAREA2,....,TAREAn]

Iterativamente evalua las tareas hasta que un predicado que no es NIL es encontrado, y regresa el resultado de la última tarea evaluada.

(Ver segundo ejemplo de la seccion 1.6)

COND [CONDICIONAL1,CONDICIONAL2,...,CONDICIONALn]

Sucesivamente evalua el CAR de cada condición (un predicado), hasta que un valor que no es NIL es encontrado, ó todos los predicados han sido evaluados.

En el caso que se encuentre un valor que no sea NIL, el CDR de el condicional es evaluado como un cuerpo de función.

```
(COND (ATOM (MEM) PRINT 'ATOMO)
      (NAME (MEM) PRINT 'NOMBRE)
      (NUMBERP (MEM) PRINT 'NUMERO)
      (NULL (MEM) PRINT 'LISTA-VACIA)
```

PROG1 [TAREA1,TAREA2,...,TAREAn]

Evalua sucesivamente las <TAREAS 1-n> y regresa el resultado de la evaluación de la primera tarea <TAREA1>.

```
(SETQ ALFA '(M N O P Q)) ---> (M N O P Q)
(PROG1 (CDR ALFA)(SETQ ALFA (CAR ALFA))) ---> (N O P)
ALFA                          ---> M
```

2.14 FUNCIONES DE LECTURA

Las funciones de lectura habilitan los programas para leer e interpretar datos-caracteres de varias fuentes.

RDS [NOMBRE-ARCHIVO,TIPO-ARCHIVO,DRIVE]

Lee un archivo del disco llamado <NOMBRE-ARCHIVO>, con su extención <TIPO-ARCHIVO> que se encuentra en la unidad de disco <DRIVE>.

```
(RDS INVENTARIO LIB A)        ; LEE DE A: INVENTARIO.LIB
```

RATOM [ ]

Lee un token y regresa el correspondiente átomo, un token es un string de caracteres, delimitados por un separador ó caracteres de ruptura.

```
(RATOM) LUIS ----> LUIS
(RATOM) 35ABC---> 35
```

READ [ ]

Lee una expresión simbólica y regresa la lista encadenada equivalente.

```
(READ)
(COMPUTADORAS EFICIENTES)  --->   (COMPUTADORAS EFICIENTES)
```

READCH [bandera]

Si la bandera no es NIL, lee un caracter simple.

READP [ ]

Regresa "T" si hay un caracter disponible para lectura, en caso contrario regresa NIL.

Esta función puede ser usada para probar si un caracter ha sido tecleado por el usuario.

2.15 FUNCIONES DE ESCRITURA

Las funciones de impresión habilitan a los programas para una salida directa de caracteres a varios destinos.

PRINT [OBJETO]

Evalua un argumento simple <OBJETO> y lo imprime en una linea.

```
(SETQ MEM 'COMPUTACION)
(PRINT MEM)
COMPUTACION
COMPUTACION
```

PRIN1 [OBJETO]

Evalua un argumento simple <OBJETO>, pero no empieza en una nueva linea.

```
(PRIN1 MEM) COMPUTACION
COMPUTACION
```

TERPRI [n]

Envia <n> cambios de linea. Se utiliza para comenzar nuevas lineas.

SPACES [n]

Envia <n> espacios (caracteres en blanco). Si <n> no es un numero positivo, no se envian espacios.

LINELENGTH [n]

Envia la longitud a las lineas de salida y automaticamente sera terminado en (n) caracteres y regresa la longitud de la linea previa.

RADIX [n]

Envia la base (RADIX) en la cual los nameros son expresados para entrada o salida. La base (radix) por default es 10 ó decimal.

```
(RADIX 16) ---> 0A              ; CAMBIA A HEXADECIMAL
(PUUS 3F6 0A4E) ---> 0E44       ; SUMA EN HEXADECIMAL
(RADIX 0A) ---> 16              ; VUELVE A DECIMAL
```

CLRSCRN [ ]

Limpia la pantalla y mueve el cursor a la esquina superior izquierda de la pantalla (posición home).

2.16 FUNCIONES DE MEDIO AMBIENTE

Medio ambiente lisp consiste de todos los valores de variables, definición de funciones y valores de propiedades presentes en el sistema.

SAVE [NOMBRE-ARCHIVO,DRIVE,BANDERA]

Si <BANDERA> es NIL, guarda el medio ambiente lisp como <NOMBRE-ARCHIVO>.SYS en la unidad de disco <DRIVE>.

Si <BANDERA> no es NIL, lo guarda el interprete como <NOMBRE-ARCHIVO>.COM en la unidad de disco <DRIVE>.

```
(SAVE 'MEX 'B)          ; GUARDA MEX.SYS EN B:
(SAVE 'MEX NIL T)       ; GUARDA MEX.COM EN EL "DEFAULT-DRIVE"
```

LOAD [NOMBRE-ARCHIVO,DRIVE]

Restaura el medio ambiente presente cuando <NOMBRE-ARCHIVO>.SYS fue creado.

```
(LOAD 'MEX 'B)          ; CARGA MEX.SYS DEL DRIVE B:
```

SYSTEM [ ]

Cierra todos los archivos abiertos, termina la ejecución lisp y regresa el control al sistema operativo.

Pulsando <Ctrl-C> mientras entra una linea de texto, tambien termina lisp y regresa el control al sistema operativo, pero los archivos no se cierran automaticamente.

CAPITULO III

APLICACION DE LISP (PROGRAMA)

3.1 ENTENDIMIENTO DE EL LENGUAJE NATURAL

Como se dijo anteriormente, el entendimiento de el lenguaje natural es una de las áreas de investigación de la Inteligencia Artificial.

Una de las caracteristicas de la raza humana es su habilidad para poderse comunicar en algun tipo de lenguaje natural, entendiendose por lenguaje natural, el que se utiliza para expresar las ideas, ya sea el alemán, el ingles ó cualquier otro.

La palabra entender se refiere a el hecho de transformar un tipo de representación en otra, siendo esta última seleccionada para que por cada evento, una acción apropiada sea desarrollada.

El programa desarrollado toma el papel de un psicoanalista en un diálogo con el usuario, por medio de una terminal.

Para su implementación se utilizó la técnica empleada por Joseph Weinzenbaum en 1966, conocida como Emparejamiento Aproximado (Approximate Matching) [Rich, E.].

La estructura del programa se basa, en el emparejamiento de las oraciones de entrada dadas por el usuario con patrones o modelos ya definidos. Dichos patrones, preguntan por palabras especificas (palabras-llaves) en las ora-

ciones que el usuario elabora, de esta manera las palabras conocidas por el programa serán identificadas y el resto de la oración será ignorada, por lo que no se necesita igualar la oración entera. Esto nos permite igualar (emparejar) una gran variedad de oraciones en un mismo patrón, pues la complejidad gramatical de el lenguaje es ignorada.

En el programa aunque el diálogo parezca sorprendentemente real e inteligente, realmente no entiende las oraciones de entrada (usuario) en el sentido de mapearlas en las estructuras que representan su significado (representación del conocimiento), en lugar de esto, son mapeadas directamente en una respuesta apropiada, dependiendo del patrón emparejado [Angulo y del Moral].

Se realizará el entendimiento de una pieza del lenguaje, de tal manera que para una situación particular desarrolle una representación apropiada, por lo tanto el programa representa un analisis superficial del lenguaje.

En resumen, el objetivo del programa es simular un diálogo inteligente, teniendo como enfoque la ingeniería, pues trata de simular inteligencia sin referirse al modo en que el hombre utiliza sus facultades mentales para entender el lenguaje natural.

3.2 EMPAREJANDO EXPRESIONES SIMBOLICAS

El proceso de comparar expresiones simbolicas, para ver si una es similar a la otra es conocido como emparejamiento (matching) [Winston y Horn].

Se desarrolló una función llamada COMP que compara un patrón con un dato, ambos listas. El patrón es un modelo que se establece, que puede contener caracteres especiales como: + ?. El dato es la oración de entrada tecleada por el usuario.

Cuando un patrón es comparado con un dato y el patrón no contiene símbolos especiales la salida será "T" (verdadero), solo si son exactamente iguales, esto es, cada posición correspondiente es ocupada por el mismo átomo.

```
(COMP '(ESTA BIEN) '(ESTA BIEN))
T
```

El símbolo "+" expande la flexibilidad de la función COMP, comparando cero ó más átomos. Un patrón con un caracter "+" puede ser igual a un dato que contenga más átomos que el patrón.

```
(COMP '(+ ESTA BIEN +) '(HOY ESTA BIEN LA SITUACION))
T
```

Como se puede apreciar la función no presta atención en verificar, si el dato es un hecho real o imaginario, solo prueba la forma ignorando su significado.

Para su implementación se adoptó una estrategia de mover el patrón y el dato, átomo por átomo, de tal manera que el átomo del patrón y el átomo del dato, se comparen en cada

posición [Winston y Horn].

Traducido a lisp la función completa quedaría así:

```
(PUTD 'COMP '(LAMBDA (PAT DAT ASIG)
     ((AND (NULL PAT)(NULL DAT))
      (COND
           ((NULL ASG) T)
           (T ASIG)))
     ((OR (NULL PAT)(NULL DAT)) NIL)
     ((EQUAL (CAR PAT)(CAR DAT))
      (OR (COMP (CDR PAT)(CDR DAT) ASIG)
          (COMP (CDR PAT)((NULL (CDR DAT)) U) ASIG)))
     ((EQUAL (CAR PAT) '+)
      (OR (COMP (CAR PAT)(CAR DAT) ASIG)
          (COMP PAT (CDR DAT) ASIG)
          (COMP (CDR PAT) DAT ASIG)))
     ((ATOM (CAR PAT)) NIL)
     ((EQUAL (P-I (CAR PAT)) '+)
      (SETQ N-ASIG (S-P (P-V (CAR PAT))
                        (CAR DAT)
                        ASIG))
      (OR (COMP (CDR PAT)(CDR DAT) N-ASIG)
          (COMP PAT (CDR DAT) N-ASIG)))
     ((AND (EQUAL (P-I (CAR PAT)) 'REST)
           (EQUAL (R-I (CAR PAT)) '?)
           (PREA (R-P (CAR PAT))(CAR DAT)))
      (OR (COMP (CDR PAT)(CDR DAT) ASIG)
          (COMP (CDR PAT)((NULL (CDR DAT)) U) ASIG))) ))
```

Se puede observar que la expresión LAMBDA contiene tres argumentos: el patron (PAT), el dato (DAT) y un tercero que es asignación (ASIG). Este último es una lista de asociación de pares de valores variables. Dicha lista de asociación, es construida por la función COMP en la recursividad si existen en el patrón patrones variables, los cuales serán explicados más adelante.

La primera cláusula checa si el fin de ambas listas es encontrado, una vez que esto sucede y ASIG es una lista vacía, la salida es "T", en caso contrario, la salida es la lista de asociación ASIG.

La siguiente cláusula checa por si una de las listas es más corta que la otra, en cuyo caso, la salida es NIL.

La siguiente checa el primer elemento (CAR) de las dos listas, y si es satisfecho, hay dos opciones de recursividad: una se llamará así mismo con el CDR de las listas, y la otra en el caso de que el dato sea una lista vacía, esta última se cumple cuando el patrón tenga un símbolo "+" a la derecha y el dato ya no contenga átomos.

Luego se checa si es el símbolo "+", en cuyo caso, inicia la recursividad llamando a COMP para ver si una de las tres posibilidades se trabaja: la primera es en el caso que solo iguale un átomo, la segunda si sigue igualando átomos y la tercera en caso que el patró tenga un símbolo "+" a la izquierda y el dato no contenga átomo inicial (recuerdese que el símbolo "+" compara cero o más átomos).

Después se checa si el patrón es un átomo, si lo es, la

salida es NIL, de esta manera se evita la busqueda en las dos clausulas siguientes, que prueban elementos y no átomos.

La siguiente dimensión generaliza COMP, para que ciertos símbolos de patrones sean asociados con valores. Estos elementos patrones, que son listas, empiezan con el símbolo "+" por ejemplo: (+ L). Los símbolos vistos, en los elementos patrones, junto con el símbolo "+" son llamados patrones variables, en este caso sería "L". Cuando COMP sucede, los patrones variables regresan asociados con la parte del dato que los elementos del patrón variable igualan. La asociación es envuelta en una lista de asociación de pares variables, regresada en ASIG despues de la recursión. Al iniciar, se utiliza la lista vacia "NIL" como tercer argumento, pues todavia no hay pares de valores variables.

```
(COMP '(EL COCHE (+ L)) '(EL COCHE ES VELOZ) NIL)
((L (ES VELOZ)))
```

Para generalizar COMP, se usaron dos funciones de acceso: P-I (patrón indicador) y P-V (patrón variable).

```
(PUTD 'P-I ' (LAMBDA (Q)
     (CAR Q) ))

(PUTD 'P-V '(LAMBDA (Q)
     (CADR Q) ))
```

Si el elemento del patrón, es una lista que comienza con el símbolo "+", sumaremos un par de valores variables a la lista de asociación en la recursividad.

El cambio necesitado para hacer que la lista de asociación de valores variables sea regresada cuando COMP ha ocurrido, esta dada por la función S-P y es asignada temporalmente ala variable N-ASIG (nueva asignación). Esta función pone al día la lista de asociación, pues checa si la lista de asociación de un patrón variable determinado, ya ha sido empezada.

```
(PUTD 'S-P '(LAMBDA (VA1 VA2 U-LI)
     (COND
          ((NULL U-LI)(LIST (LIST VA1 (LIST VA2))))
          ((EQUAL VA1 (CAAR U-LI))
           (CONS (LIST VA1 (APPEND (CADAR U-LI)(LIST VA2)))
                 (CDR U-LI)))
          (T (CONS (CAR U-LI)
                   (S-P VA1 VA2 (CDR U-LI))))) ))
```

Se generaliza aún más la función COMP, haciendo que una posición especifica sea llenada con el miembro de algun grupo de datos. Al usar la caracteristica REST (restricción), una lista descriptiva es sustituida en el patrón donde previamente solo átomos son esperados.

En la última cláusula, se utilizan dos funciones auxiliares R-I y R-P:

```
(PUTD 'R-I '(LAMBDA (Q)
     (CADR Q) ))
```

```
(PUTD 'R-P '(LAMBDA (Q)
     (CDDR Q) ))
```

Dicha clausula, checa que se cumplan tres condiciones: la primera es que el primer átomo sea la palabra REST, la segunda es que el segundo átomo sea el símbolo "?" y la tercera es que se satisfaga la función PRBA, la cual prueba el predicado, en este caso, que el átomo pertenezca a la lista definida previamente.

```
(PUTD 'PRBA '(LAMBDA (PRED ARG)
     (COND
          ((APPLY (CAR PRED)(LIST ARG)) T)
          (T NIL)) ))
```

Un ejemplo sería:

```
(PUTD 'FAMILIA '(LAMBDA (W)
     (MEMBER W '(MADRE PADRE HIJO)) ))

(COMP '(+ (REST ? FAMILIA) +) '(MI PADRE ES BUENO) NIL)
T
```

3.3 FUNCION PRINCIPAL

La función DIALOG interactua con el usuario en una terminal como lo haría un sicólogo.

Dicha función está escrita como un loop atraves de condiciones que contienen palabras-llaves y frases junto con preguntas apropiadas. Si la oración de entrada (usuario), contiene dos palabras-llaves diferentes, como se sigue un mapeo secuencial, el patrón que este primero en la función es el que genera la pregunta, esto quiere decir que la prioridad de la palabras-llaves depende de su posición en la función.

El primer patrón sería (+ ESTOY PREOCUPADO (+ L)). El símbolo "+" empareja cualquier numero de palabras incluyendo ninguna. En (+ L), el patrón variable "L" va a almacenar lo que está escrito en la oración de entrada (usuario) después de la palabra PREOCUPADO, para desplegar la pregunta con lo contenido en "L".

Cuando el patrón sea (+ (REST ? PRED) +), y la oración de entrada contenga alguna de las palabras definidas en el predicado, se genera una pregunta, esto significa, que un grupo de palabras-llaves serán tratadas identicamente, generando la misma pregunta.

Se puede dar el caso que la oración de entrada, no contenga ninguna palabra-llave, entonces se despliega una pregunta de proposito general (DEF), y cuando ya no puede generar otra pregunta la computadora se despide.

Cuando algunas de las condiciones se cumplen, se levanta una bandera, por ejemplo: (SETQ PRE T), para que después,

si la oración de entrada no es emparejada por ninguna condición, se despliega la pregunta que contiene la bandera, en este caso, (PRE), donde es puesta en NIL.

Para simplificar el proceso de generación de salida, algunas palabras son inmediatamente traducidas en nuevas formas, de esta manera se elaborará la pregunta dirigiendose al usuario, este es el caso de MI --> TU, CONMIGO --> CONTIGO, etc.

Esto se lleva a cabo en la función P-P, cuya tarea es la de desplegar lo almacenado en los patrones variables.

```
(PUTD 'P-P '(LAMBDA (U-L)
     ((NULL U-L))
     (SETQ B (POP U-L))
     (COND
          ((OR (EQUAL B 'YO)
               (EQUAL B 'MI))
           (PRIN1 "TU"))
          ((EQUAL B 'ME )
           (PRIN1 "TE"))
          ((EQUAL B 'CONMIGO)
           (PRIN1 "CONTIGO"))
          ((EQUAL B 'MIO)
           (PRIN1 "TUYO"))
          (T (PRIN1 B)))
     (SPACES 1)
     (P-P U-L) ))
```

La función M-V, nos da el valor requerido de la lista de asociación de valores variables, permitiendonos así el manejo de la informnación almacenada en la lista de asociación generada por COMP.

```
(PUTD 'M-V '(LAMBDA (K LI)
     (CADR (ASSOC K LI))  ))
```

La función R-L al igual que PRINT y PRIN1, establecen comunicación con el usuario.

R-L nos permite leer la oración de entrada del usuario, hasta que <RETURN> es pulsado, como si fuera un READLINE, sin necesidad de encerrar la oración entre paréntesis. Esta función, regresa una lista de tokens leidos de una linea de entrada, los espacios son ignorados excepto para el proposito de separar los tokens [Rich, A.].

```
(PUTD 'R-L '(LAMBDA (C)
     (SETQ C (READCH T))
     ((EQ C (ASCII 13)) NIL)
     ((EQ C '" ")
      (R-L))
     (CONS (RATOM)(R-L)) ))
```

Antes de definir la función DIALOG, definamos los predicados que ocuparemos para los patrones (REST ? PRED):

```
(PUTD 'SAL '(LAMBDA (W)
     (MEMBER W '(SALUD ENFERMEDAD ENFERMO DEBIL SALUDABLE
                SANO)) ))

(PUTD 'TRA '(LAMBDA (W)
     (MEMEBER W '(TRABAJO EMPLEO NEGOCIO INFLACION)) ))

(PUTD 'DESP '(LAMBDA (W)
     (MEMBER W '(ADIOS BYE BASTA CALLATE CALLAR CALLAS
                CALLARAS)) ))

(PUTD 'SEN '(LAMBDA (W)
     (MEMBER W '(LLORO LLORAR RIO REIR ENOJO ENOJAR ODIO
                ODIAR OLVIDO OLVIDAR ARREPIENTO ARREPENTIDO
                AMO AMAR QUIERO NECESITO DESEO)) ))

(PUTD 'IN '(LAMBDA (W)
     (MEMBER W '(QUIZA POSIBLEMENTE)) ))

(PUTD 'PER '(LAMBDA (W)
     (MEMBER W '(ELLA ELLOS ELLAS AMIGO AMIGA)) ))

(PUTD 'FAM '(LAMBDA (W)
     (MEMBER W '(PADRE MADRE PAPA MAMA HERMANO HERMANA PADRES
                HERMANOS ABUELO ABUELA ABUELOS HIJO HIJA
                HIJOS)) ))
```

```
(PUTD 'COM '(LAMBDA (W)
     (MEMBER W '(COMPUTADOR COMPUTADORA COMPUTADORAS
                ORDENADOR ORDENADORES)) ))

(PUTD 'DEP '(LAMBDA (W)
     (MEMBER W '(DEPRIMIDO DEPRIME DEPRIMO DEPRESION)) ))

(PUTD 'DIN '(LAMBDA (W)
     (MEMBER W '(DINERO DOLAR PESOS DOLARES PRECIOS
                ECONOMIA)) ))

(PUTD 'TR '(LAMBDA (W)
     (MEMBER W '(TRISTE TRISTEZA)) ))

(PUTD 'IGU '(LAMBDA (W)
     (MEMBER W '(IGUAL IGUALES SIMILAR SIMILARES)) ))

(PUTD 'TIE '(LAMBDA (W)
     (MEMBER W '(HOY MA#ANA AYER AHORA TIEMPO HORAS MINUTOS
                SEGUNDOS TARDE DIAS DESPUES)) ))
```

De esta manera, la función DIALOG la definiremos así:

```
(PUTD 'DIALOG '(LAMBDA ()
     (CLRSCRN)
     (PRINT "HOLA !   BIENVENIDO A DIALOG")
     (SETQ N 1) (SETQ CO NIL) (SETQ SE NIL)
```

```
(SETQ PRE NIL) (SETQ I NIL) (SETQ DE NIL)
(SETQ RE NIL) (SETQ FA NIL) (SETQ TRI NIL)
(SETQ IG NIL) (SETQ DI NIL) (SETQ SA NIL)
(SETQ EC NIL) (SETQ TI NIL) (SETQ PE NIL)
(SETQ FE NIL) (SETQ DEF T) (SETQ START T)
(LOOP
     ((ZEROP N)
      (TERPRI)
      (PRINT "FUE UN PLACER PLATICAR CONTIGO")
      (TERPRI 3)
      (PRINT "      ----  ADIOS  ----"))
      (TERPRI)
      (SETQ O (R-L (READP)))
      (TERPRI)
      (COND
           ((SETQ R (OR (COMP '(+ ESTOY PREOCUPADO
                                  (+ L)) O NIL)
                        (COMP '(+ ME PROCUPO (+ L))
                              O NIL)))
            (SETQ PRE T)
            (PRIN1 "DESDE CUANDO TE PREOCUPAS ")
            (P-P (M-V 'L R))
            (PRINT "?"))
           ((SETQ R (COMP '(+ MI (+ J)) O NIL))
            (PRIN1 "TU ")
            (P-P (M-V 'J R))
            (PRINT "?"))
           ((SETQ R (OR (COMP '(+ RECUERDO (+ H))
```

```
                    O NIL)
                (COMP '(+ ME ACUERDO (+ H)))
 (SETQ RE T)
 (PRIN1 "PORQUE RECUERDAS ")
 (P-P (M-V 'H R))
 (PRINT "JUSTAMENTE AHORA ?"))
((COMP '(+ (REST ? IN) +) O NIL)
 (SETQ I T)
 (PRINT "TU RESPUESTA ES INSEGURA, PORQUE
        ?"))
((COMP '(+ (REST ? PER) +) O NIL)
 (SETQ PE T)
 (PRINT "QUE RELACION EXISTE ENTRE USTEDES
        ?"))
((COMP '(+ (REST ? FAM) +) O NIL)
 (SETQ FA T)
 (PRINT "QUE OPINION TIENES SOBRE LA FAMILIA
        ?"))
((COMP '(+ (REST ? COM) +) O NIL)
 (SETQ CO T)
 (PRINT "PORQUE MENCIONAS A LAS COMPUTADORAS
        ?"))
((OR (COMP '(+ NO SOY FELIZ +) O NIL)
     (COMP '(+ INFELIZ +) O NIL))
 (SETQ FE T)
 (PRINT "PIENSAS QUE VENIR AQUI TE AYUDARA
        A SER FELIZ ?"))
```

```
((COMP '(+ (REST ? DEP) +) 0 NIL)
 (SETQ DE T)
 (PRINT "TU CREES QUE LA DEPRESION
         PROPORCIONE ALGUN BENEFICIO ?"))
((COMP '(+ (REST ? DIN) +) 0 NIL)
 (SETQ DI T)
 (PRINT "SI TUVIERAS MUCHO DINERO, SERIAS
         MAS FELIZ ?"))
((OR (COMP '(+ NO QUIERO PLATICAR +) 0 NIL)
     (COMP '(+ EN NADA +) 0 NIL))
 (SETQ N 0))
((COMP '(+ (REST ? TR) +) 0 NIL)
 (SETQ TRI T)
 (PRINT "ALGUNA VEZ HAS SOLUCIONADO TUS
         PROBLEMAS ESTANDO TRISTE ?"))
((COMP '(+ (REST ? IGU) +) 0 NIL)
 (SETQ IG T)
 (PRINT "EN QUE SON IGUALES ?"))
((COMP '(+ (REST ? SAL) +) 0 NIL)
 (SETQ SA T)
 (PRINT "QUE PIENSAS DE LA FRASE:  MENTE
         SANA CUERPO SANO"))
((COMP '(+ (REST ? TIE) +) 0 NIL)
 (SETQ TI T)
 (PRINT "SIEMPRE PLANIFICAS TUS ACTIVIDADES
         ?"))
((COMP '(+ (REST ? TRA) +) 0 NIL)
 (SETQ EC T)
```

```
 (PRINT "TIENES PROBLEMAS ECONOMICOS ?"))
((OR (COMP '(PORQUE) 0 NIL)
     (COMP '(PORQUE ?) 0 NIL))
 (PRINT "EL QUE HACE LAS PREGUNTAS SOY YO,
         TU RESPONDE"))
((COMP '(+ (REST ? SEN) +) 0 NIL)
 (SETQ SE T)
 (PRINT "CREES QUE TUS SENTIMIENTOS
         PROVENGAN DE TUS PENSAMIENTOS ?"))
((COMP '(+ (REST ? DESP) +) 0 NIL)
 (SETQ N 0))
((COMP '(SI) 0 NIL)
 (PRINT "ME GUSTARIA QUE ME DIERAS MAS
         DETALLES ?"))
((COMP '(NO) 0 NIL)
 (PRINT "PORQUE ?"))
(START (SETQ START NIL)
       (PRINT "EN QUE PUEDO AYUDARTE ?"))
(PRE (SETQ PRE NIL)
     (PRINT "EL ESTAR PREOCUPADO
             SOLUCIONARA TUS PROBLEMAS ?"))
(RE (SETQ RE NIL)
    (PRINT "CREES QUE ES MEJOR VIVIR EN EL
            PASADO QUE EN EL PRESENTE ?"))
(SE (SETQ SE NIL)
    (PRINT "ERES CAPAZ DE CONTROLAR TUS
            SENTIMIENTOS ?"))
```

```
(I (SETQ I NIL)
   (PRINT "CREES QUE PIENSAS POR TI
          MISMO ?"))
(PE (SETQ PE NIL)
    (PRINT "ERES UN HACEDOR O UN CRITICO
           ?"))
(FA (SETQ FA NIL)
    (PRINT "CUENTAME MAS ACERCA DE TU
           FAMILIA ?"))
(CO (SETQ CO NIL)
    (PRINT "ME HABLASTE DE COMPUTADORAS,
           REALMENTE TE PREOCUPAN ?"))
(PE (SETQ PE NIL)
    (PRINT "RIGES TU VIDA POR LO QUE
           PIENSAN LOS DEMAS ?"))
(DE (SETQ DE NIL)
    (PRINT "PUEDES ACEPTARTE TAL COMO ERES
           Y EVITAR LOS REPROCHES ?"))
(DI (SETQ DI NIL)
    (PRINT "EL DINERO TE PROPORCIONA
           SEGURIDAD EN TI MISMO ?"))
(TRI (SETQ TRI NIL)
     (PRINT "TE RESPETAS A TI MISMO ?"))
(IG (SETQ IG NIL)
    (PRINT "PUEDES DARME UN EJEMPLO
           ESPECIFICO ?"))
(SA (SETQ SA NIL)
    (PRINT "TE ENOJAS MUY FRECUENTEMENTE ?"))
```

```
(T1 (SETQ T1 NIL)
    (PRINT "ERES DE LOS QUE POSTERGA SUS
            ACTIVIDADES ?"))
(EC (SETQ EC NIL)
    (PRINT "PIENSAS QUE EL DINERO LO ES
            TODO ?"))
(DEF (SETQ DEF NIL)
     (PRINT "ERES TU QUIEN ESTABLECE TUS
             PROPIAS REGLAS DE CONDUCTA ?"))
(T (PRINT "LO SIENTO, NO DISPONGO DE MAS
           TIEMPO")
   (SETQ N 0)))) ))
```

3.4 MANEJO DEL SISTEMA

Una vez cargado el sistema operativo se procede a cargar el sistema, verificando que el diskette contenga una copia de MULISP.COM y de DIALOG.SYS. Cuando aparece el prompt del sistema operativo se teclea: MULISP DIALOG <RETURN>, donde además de cargar muLISP, se carga el archivo DIALOG.SYS desde el nivel de sistema operativo. Después, aparece el prompt de muLISP, que consiste en el signo de pesos "$", caracter seguido por un espacio y el cursor de la computadora, indicando que el sistema está listo para aceptar la entrada.

Para ejecutar el programa se teclea la función dialog entre parèntesis: (DIALOG) <RETURN>, como se puede apreciar muLISP solo acepta mayúsculas, por lo que se recomienda que antes de empezar el sistema se pulse <CAPS LOCK>.

En caso que se cometa algun error al teclear, se pulsa <BACK SPACE> como normalmente se utiliza, borrando el ó los caracteres equivocados.

Para salirse de la ejecución, solo basta teclear alguna palabra de despedida como: ADIOS, BYE, etc.. Una vez hecho esto vuelve aparecer el prompt de muLISP, y para regresar al sistema operativo se teclea (SYSTEM) <RETURN>.

CAPITULO IV

RECOMENDACIONES Y CONCLUSIONES

El lenguaje de programación muLISP proporcionó el soporte y las estructuras necesarias para el desarrollo e implementación del programa de Entendimiento del Lenguaje Natural, en este caso el Español.

El programa desarrollado intenta sucitar los sentimientos de su interlocutor mediante preguntas poco comprometidas, que dan pie al usuario a continuar la comunicación, por lo tanto no da una solución al problema expuesto por el usuario, sino que lo hace dialogar, en cuyo caso le serviría como desahogo emocional.

Descomponer el programa en funciones, facilitó su implementación y aumentó su legibilidad, además las funciones creadas pueden utilizarse para cualquier otra aplicación, con muy poca o ninguna modificación, de esta manera se hace posible la reutilización de las funciones.

El emparejamiento de expresiones simbólicas, es una herramienta importante en algunos sistemas de Inteligencia Artificial, por lo que la función COMP puede ser utilizada para manejar una gran base de conocimientos ó en el desarrollo de un Sistema Experto.

La ventaja de usar la técnica de emparejamiento aproximado, es que la gramatica es inusual, siendo su desventaja que al incrementar la tolerancia permitida en el emparejamie-

ndo, aumenta el número de patrones que hay que comparar, por lo tanto se incrementa el proceso principal de busqueda, aumentando así el tiempo de respuesta.

Otra aplicación practica de esta técnica seria la implementación de un sistema de reservación de lineas aéreas, de manera que la computadora hiciera el papel de un empleado de una agencia de viajes, para lo cual se recomienda modificar la estructura principal de la función DIALOG, junto con los predicados, de manera de que los patrones a emparejar esten relacionados con las preguntas más usadas en el ambiente turistico, además, habria que anexar una base de datos que incluyera: horario de vuelos, destinos, aereolineas, capacidad del avión, reservaciones confirmadas, precios, etc..

Angulo, J. y del Moral, A. INTELIGENCIA ARTIFICIAL. Paraninfo: Madrid, 1986.

Charniack, E. y McDermott, D. INTRODUCTION TO ARTIFICIAL INTELLIGENCE. Addison-Wesley: Reading, Mass., 1985.

Crosson, F. INTELIGENCIA HUMANA E INTELIGENCIA ARTIFICIAL. Fondo de Cultura Economica: México, D.F., 1975.

Presser, L., Cárdenas, A. y Marin, M. CIENCIAS DE LA COMPUTACION, Vol. II. Limusa-Wiley: México, 1972.

Rich, A. muLISP-83 REFERENCE MANUAL. The Soft Warehouse: Honolulu, Hawaii, 1983.

Rich, E. ARTIFICIAL INTELLIGENCE. McGraw-Hill: Singapore, 1983.

Winston, P. H. y Horn, B. LISP. Addison-Wesley: Reading, Mass., 1984.

Winston, P. H. ARTIFICIAL INTELLIGENCE. Addison-Wesley: Reading, Mass., 1984.