QUANTUM THEORY PROJECT
FOR RESEARCH IN ATOMIC, MOLECULAR AND SOLID STATE
CHEMISTRY AND PHYSICS
UNIVERSITY OF FLORIDA, GAINESVILLE, FLORIDA

TWO LISP PATTERN RECOGNITION FUNCTIONS: SIMILAR AND SIMILAR*

by

Alberto Verjovsky*

#2.

February 28 1963

* Student, University of Mexico

PROGRAM NOTE # 2

ACKNOWLEDGEMENTS

This paper has been prepared as a portion of the exercises in an informal winter institute in symbolic programming held by Mr. H. V. McIntosh and The Quantum Theory Project at the University of Florida. The author wishes particularly to express his gratitude to the Physics Departament for its kind hospitality during his stay, and the University Computer Center for the time which was used to verify the functions described in this paper. This time was made available by a grant from the National Science Foundation, and the cooperation of the IBM Corporation.

Alberto Verjovsky.

Gainesville, Fla.
28 February, 1963.

GENERAL

A number of student projects have been subsumed under the University of Florida Computer Center account number 052101, which has generally been used to give students training in using the programming language LISP. It is generally expected that the participants in this account will prepare a report describing the LISP functions they have chosen to study, as well as possible applications. In this way it is possible to gradually build up a lore of tested functions which may be used by succeeding students or by other persons interested in computer languages.

Due to the ephemeral nature of any computer program, which will constantly undergo extension and revision, as well as the specialized nature of the possible audience, the reports of student projects will be generally retained as a file copy, and reproduced by Xerox process upon demand.

We gratefully acknowledge the interest and cooperation of both the direction and management of the University of Florida Computer Center, as well as the operating staff which has made this series of studies possible. Ultimately, of course, our support rests with the National Science Foundation and the IBM corporation, who made the computational facilities available, as well as the State of Florida and the officials of the University, all of whom have devoted considerable effort to establishing acomputer center.

The version of the LISP processor used was MBLISP, for the support in the development of which our thanks is due to RIAS and the Digital Computations Section of Martin Baltimore. Continued development of MBLISP has taken place with the support of the University of Florida center.

Harold V. McIntosh

ABSTRACT

In many applications of LISP it is necessary to discover whether an expression has a certain format, irrespective of its actual elements. For this purpose two pattern recognition functions are described.

The first, (SIMILAR X L), is a predicate of two variables. One, X, is a reference expression wich contains certain blanks. These blanks, depending on their nature, are to be matched in the other expression L, either against: 1) An arbitrary expression, 2) an arbitrary fragment of a list, or 3) a subexpression possessing a stated property.

The second function, (SIMILAR* X B L), differs in the respect that it also contains as an argument a list of bound variables appearing in the expression X. It is not a predicate; rather its value is a dictionary of the values of the bound variables which make the expressions X and L similar.

An auxiliary function, (TABLE G U V) is first described, which may be used to generate examples illustrating functions of two variables, and is applied to the case at hand.

"T A B L E"

Supongamos que tenemos una función "G", de dos argumentos, y queremos concer sus cualidades, el mejor camino para conseguir nuestro propósito, es calcular el valor de la función para los diferentes valores posibles de sus dos argumentos, es decir, calcularla para los rangos de éstos. Si el primer argumento puede tomar los valores $X_1, X_2, \ldots X_n$ y el segundo $Y_1, Y_2, \ldots Y_m$ o bien estos son los valores para los cuales nos interesa saber como se comporta la función, sería útil poder construir una tabla en la que aparezcan los valores de la función calculados para cualquier pareja de argumentos extraídos del conjunto de las X_i y las Y_j respectivamente, es decir:

G	X_1	X_2	---	X_n
Y_1	$G(X_1,Y_1)$	$G(X_2,Y_1)$	---	$G(X_n,Y_1)$
Y_2	$G(X_1,Y_2)$	$G(X_2,Y_2)$	---	$G(X_n,Y_2)$
⋮	⋮	⋮		
Y_n	$G(X_1,Y_n)$	$G(X_2,Y_n)$	---	$G(X_n,Y_n)$

Para la construcción de dicha tabla, usamos la función "TABLE". Esta función tiene tres argumentos:

1o. G, una función de dos argumentos.

2o. U, lista de valores del primer argumento de G.

3o. V, lista de valores del segundo argumento de G.

Nota: Se puntualiza que los argumentos segundo y tercero de "TABLE", se ponen en el orden en que estén los argumentos de G, si ésta es asimétrica.

"TABLE" procede de la siguiente manera:

Inicialmente, aplica la función a cada elemento de U, con el primerode V y después continúa el proceso recurrentemente hasta que V se vacía; su definición sería entonces:

```
(TABLE (LAMBDA (G U V) (COND
((NULL V) (LIST))

((T) (CONS (FOREACH (FUNCTION (LAMBDA (X Y) (G X Y)))
                                          U (CAR V))
                    (TABLE G U (CDR V)))))))
```

O usando "MAPCAR":

```
(TABLE (LAMBDA (G U V) (MAPCAR (FUNCTION (LAMBDA (W)
(FOREACH (FUNCTION (LAMBDA (X Y) (G X Y))) U W)))) V)))
```

Para aclarar la definición anterior, conviene hacer un - esquema en el que la función G, queda representada por un segmento de recta, y en los puntos inicial y final de éste, se ponen sus - argumentos, esto es:

G(X,Y), se representa por:

X———Y

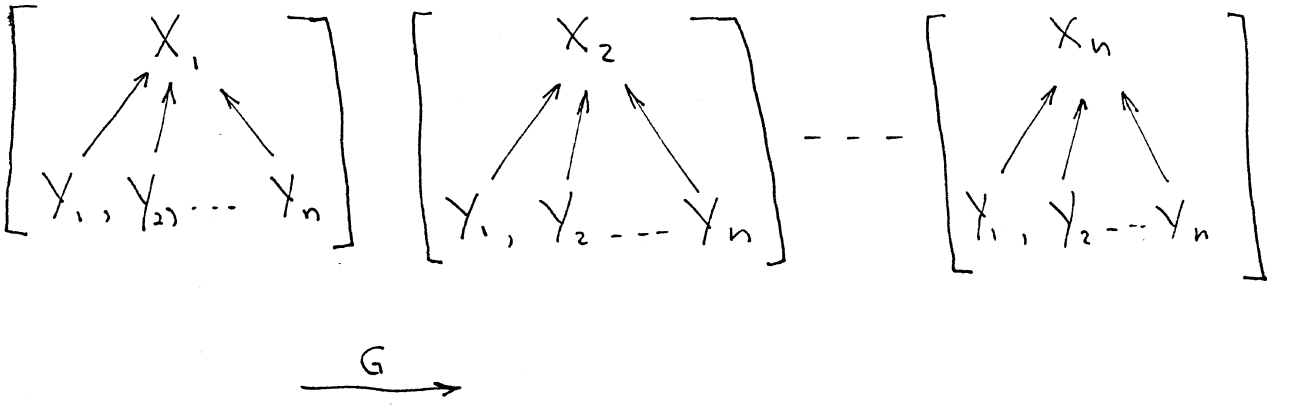
Si la función es asimétrica o sea $G(X,Y) \neq G(Y,X)$, entonces representamos a G por un segmento dirigido de tal modo, que el orden de sus argumentos lo indique la dirección del segmento, o sea:

G(X,Y), será:

X———→Y

De acuerdo con lo anterior, el esquema queda en la forma siguiente:

TABLE:



Como el resultado de esta función va a ser una lista y - no un arreglo rectangular, se puede usar la función "PRINT" para - cada X_i e imprima cada vez que usamos FOREACH, con lo que la definición sería:

```
(TABLE (LAMBDA (G U V) (MAPCAR (FUNCTION (LAMBDA (W)
(PRINT (LIST (APPEND (QUOTE (U =)) (LIST W))
(FOREACH (FUNCTION (LAMBDA (X Y) (G X Y))) U W))))) V)))).
```

S I M I L A R

El proceso de listas exige, en muchas ocasiones, el conocimiento de ciertas propiedades en éllas y esto se debe en gran parte, al hecho de que la valuación de una función, depende en gran — cantidad de las propiedades de sus argumentos. Si quisieramos, por ejemplo, efectuar una operación únicamente a un tipo específico de expresión , necesitaríamos una forma de detectarla, ya sea por contener ciertos elementos, por tener cierta estructura o cumplirse en élla un predicado.

Como consecuencia de lo antes establecido, es necesario agrupar las expresiones en clases, correspondiendo a cada clase, - un número de elementos y la propiedad que las incluye en ésta. Esto da lugar a la función "SIMILAR", que es la que hace distinciones entre elementos que tengan la misma propiedad, siendo ésta cualquiera atribuída a expresiones, (pues las características de éllas, pueden reducirse a elementos y forma, la disposición de los primeros - genera la segunda) y "SIMILAR" calcula cualquier característica.

Esta función es un predicado y tiene dos argumentos, una función modelo hecha de tal modo, que toda expresión similar a élla, esto es si la función toma el valor "T", pertenecerá a la misma - clase; se tiene asi un conjunto de elementos similares y dada una - expresión sabemos si pertenece o no al conjunto al tomar la función los valores "T" y "F" cuando se compara con la lista modelo; esto a su vez nos indica que "SIMILAR" es una función asimétrica, ya que - el primer argumento es fijo para cada propósito, es decir, se tiene que usar "QUOTE".

Al hacer la comparación de entidades con la estructura - modelo, es conveniente introducir en ésta un símbolo que represente a una expresión cualquiera, pues con esto tendremos la propiedad de unicidad y existencia de elementos, aqui se usara el apóstrofe con este fin.

También es importante asociarle un símbolo que aqui será ''', a una sucesión de cualquier número (ninguno inclusive) y elementos de lista, es decir, algo parecido a los tres puntos en matemáticas.

Si se quiere usar un símbolo atómico como el nombre de un predicado, se forma la lista modelo:

(' ' P)

En la que los dos apóstrofes son un símbolo que nos dice que su sucesor, es decir, P es un predicado; todo elemento para que sea verdadero el predicado, será similar con este modelo.

También se pueden usar modelos sin contener símbolos - que tengan una misión previa o símbolos atómicos como modelos — (entonces, la similitud será cierta si el modelo es un apóstrofe o si es igual al segundo argumento de "SIMILAR") y queda así - como caso particular la igualdad entre expresiones. Se usa el - método recurrente tomando "SIMILAR" elemento a elemento entre — dos listas. (Se hace la observación de que una lista modelo no puede contener sus dos primeros elementos iguales a '''). De este modo podemos hacer una lista modelo a voluntad para cada caso particular.

La definición de "SIMILAR" queda formalmente hecha, como sigue:

```
(1).-  (SIMILAR (LAMBDA (X L) (OR

(A).-  (EQ X (QUOTE ''))

(B).-  (EQ X L)

(C).-  (AND (NULL X) (NULL L))

(D).-  (AND (NOT (NULL X)) (NOT (ATOM X)) (OR

(D').- (AND (EQ (CAR X) (QUOTE ''')) (NOT (ATOM L)) (OR

(1').- (NULL (CDR X))

(2').- (AND (NOT (NULL (CDR L))) (OR

(2'-1).- (SIMILAR (CDR X) (CDR L))

(2'-2).- (SIMILAR X (CDR L))))

(3').- (AND (NOT (NULL L)) (SIMILAR (CDR X) L))))

(D'').- (AND (EQ (CAR X) (QUOTE '')) ((CADR X) L))
        (AND (NOT (ATOM L)) (NOT (NULL L))

(D''').- (SIMILAR (CAR X) (CAR L))
         (SIMILAR (CDR X) (CDR L)))))))
```

Para aclarar la definición anterior, se explicarán paso - a paso los puntos de la misma, a continuación:

(1).- "SIMILAR", es una función de dos argumentos, el primero es una lista fija, y sirve de modelo, al aplicarse —— "SIMILAR" se pone:

(SIMILAR (QUOTE ——) ——)

El segundo argumento es una expresión cualquiera, "SIMILAR" toma el valor T, si alguna de las condiciones siguientes es cierta:

(A).- Si X es un apóstrofe entonces es similar con cualquier expresión.

(B).- Atomos iguales son similares.

(C).-La lista vacía es similar con ella misma.

(D).- X tiene que ser una lista con al menos un elemento (pues ya se probó si era átomo o lista vacía) y tiene que ser verdadero cualquiera de lo que sigue:

(D').- Si (CAR X) es el símbolo : ''' y L no es un átomo, - entonces "SIMILAR" toma el valor T si se cumple:

(1').- La lista (''') es similar con cualquier lista (este — símbolo no se puede usar en sucesión más de una vez ya que (''' ''') y (''') son modelos equivalentes).

(2').- L tiene más de un elemento y se satisface:

(2'-1).- Los "CDR" son iguales.

(2'-2).- X es similar a (CDR L)

(3').- L no es vacía y (CDR X) es similar a L.

(D'').- X es de la forma ('' P) y el predicado aplicado a L es cierto, es decir:
((CADR X) L) ES T

(D'''.- Similitud elemento a elemento.

Todo lo dicho anteriormente se puede esquematizar en el - siguiente diagrama, en el que se dibujan las funciones lógicas "AND" y "OR", como un arreglo vertical de asteriscos o signos más, respectivamente, de los cuales salen ramas horizontales hacia cada uno de sus argumentos, una serie de guiones para argumentos afirmativos y otra de signos de igualdad para las negaciones.

```
          ---(SIMILAR X L)---
                                                              2/22/63
                                                              VERJOVSKY


+-----(EQ X (QUOTE -))
+
+-----(EQ X L)
+
+----------------*-----(NULL X)
+                *
+                *-----(NULL L)
+
+-----*=====(NULL X)
      *
      *=====(ATOM X)
      *
      *-----+-----*-----(EQ (CAR X) (QUOTE ---))
            +     *
            +     *=====(ATOM L)
            +     *
            +     *-----+-----(NULL (CDR X))
            +           +
            +           +
            +           +-----*=====(NULL (CDR L))
            +           +     *
            +           +     *-----+-----(SIMILAR (CDR X) (CDR L))
            +           +           +
            +           +           +-----(SIMILAR X (CDR L))
            +           +
            +           +
            +           +-----*=====(NULL L)
            +                 *
            +                 *-----(SIMILAR (CDR X) L)
            +
            +
            +-----*-----(EQ (CAR X) (QUOTE --))
            +     *
            +     *-----((CADR X) L)
            +
            +
            +-----*=====(ATOM L)
                  *
                  *=====(NULL L)
                  *
                  *-----(SIMILAR (CAR X) (CAR L))
                  *
                  *-----(SIMILAR (CDR X) (CDR L))

 LEGEND ...

      *   AND
      +   OR
      =   NOT

)00054 *
```

```
(SIMILAR (LAMBDA (X L) (OR
(EQ X (QUOTE -))
(EQ X L)
(AND (NULL X) (NULL L))
(AND (NOT (NULL X)) (NOT (ATOM X)) (OR
(AND (EQ (CAR X) (QUOTE ---)) (NOT (ATOM L)) (OR
(NULL (CDR X))
(AND (NOT (NULL (CDR L))) (OR
(SIMILAR (CDR X) (CDR L))
(SIMILAR X (CDR L))))
(AND (NOT (NULL L)) (SIMILAR (CDR X) L))))
(AND (EQ (CAR X) (QUOTE --)) ((CADR X) L))
(AND (NOT (ATOM L)) (NOT (NULL L)) (SIMILAR (CAR X) (CAR L))
(SIMILAR (CDR X) (CDR L)))))))))
```

00000015 *

20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3

S I M I L A R*

Una extensión de "SIMILAR" es la función "SIMILAR*", en - cuyos argumentos se incluyen variables, es una función de tres argumentos:

X.- Que tiene el mismo significado que en "SIMILAR".

B.- Que representa una lista, cuyo primer elemento es la - lista vacía y los elementos restantes, son una colección de variables.

L.- Una expresión cualquiera.

"SIMILAR*", toma dos valores, uno fijo igual a F, en caso de queno existan valores posibles de las variables que hagan similares las expresiones X y L. El otro valor, es una lista cuyo primer elemento es otra lista, en forma de diccionario, es decir, las variables seguidas de los valores que hacen que X y L sean similares. Los siguientes elementos, son variablesque no afectan la similitud, esta lista se obtiene con modificaciones a B.

Si X es vacía hay dos casos:

a).- Que L también lo sea y en tal caso el valor es B.

b).- Que L no sea vacía, tomando "SIMILAR*" el valor F.

Si X es un apóstrofe entonces es similar a cualquier expresión, y "SIMILAR*" toma el valor B.

Si x es un átomo que pertenece a las variables, entonces "SIMILAR*" tiene como valor la lista que resulta de introducir a (CAR B), X proseguida de L, y remover de (CDR B) a X.

Si X es un átomo que no sea una variable, ni tampoco - igual a un apóstrofe , entonces X es similar sólo a un átomo igual a él.

Con estas condiciones finales, y las propias de "SIMILAR", se genera la definición de "SIMILAR*" de la siguiente manera:

```
(REMOVE (LAMBDA (X L) (COND
((EQ X (CAR L)) (CDR L))
((T) (CONS (CAR L) (REMOVE X (CDR L)))))))

(ASSOC* (LAMBDA (X L) (COND
((NULL L) X)
((EQ X (CAR L)) (CADR L))
((T) (ASSOC* X (CDDR L))))))

(SIMILAR* (LAMBDA (X B L) (COND
((NULL X) (COND
((NULL L) B) ((T) (F))))
((ATOM X) (COND
((EQ X (QUOTE -)) B)
((ELEMENT X (CDR B))
(CONS (CONS X (CONS L (CAR B)))
(REMOVE X (CDR B))))
((EQUAL (ASSOC* X (CAR B)) L) B)
((T) (F))))
((AND (EQ (CAR X) (QUOTE --)) ((CADR X) L)) B)
((AND (EQ (CAR X) (QUOTE ---)) (NOT (ATOM L)))
(COND ((NULL (CDR X)) B)
((NULL L) (F))
((T) ((LAMBDA (Z) (COND
((EQ Z (F)) (COND ((NULL (CDR L)) (F))
((T) ((LAMBDA (W) (COND
((EQ W (F)) (SIMILAR* (CDR X) B (CDR L)))
((T) W))) (SIMILAR* X B (CDR L))))))
((T) Z))) (SIMILAR* (CDR X) B L))))))
((AND (NOT (ATOM L)) (NOT (NULL L)))
((LAMBDA (Z) (COND
((EQ Z (F)) (F))
((T) (SIMILAR* (CDR X) Z (CDR L)))))
(SIMILAR* (CAR X) B (CAR L))))
((T) (F)))))
```

00000036 *

```
(APPLY SIMILAR* ((+ (* (COS X) (COS Y)) (* (SIN X) (SIN Y))) (() X Y)
(+ (* (COS (/ PI 2)) (COS (/ PI 3))) (* (SIN (/ PI 2)) (SIN (/ PI 3))))))
VALOR
((Y (/ PI 3) X (/ PI 2)))
(APPLY SIMILAR* ((+ (** (COS X) 2) (** (SIN X) 2)) (() X)
(+ (** (COS U) 2) (** (SIN U) 2))))
VALOR
((X U))
(APPLY SIMILAR* ((+ (** (COS U) 2) (** (SIN U) 2)) (() X)
(+ (** (COS X) 2) (** (SIN Y) 2))))
VALOR
F
APPLY TABLE ((LAMBDA (U V) (SIMILAR* U (QUOTE (() X Y Z)) V))
((X) (- X) (--- X) (--- X - Y) (X Y)) ((0) (0 1) (0 1 2) (0 1 2 3) (0 1 2 3 4
(0 1 2 3 4 5))))
VALOR
U = (0)  ..........  (((X 0) Y Z) F ((X 0) Y Z) F F))
U = (0 1)  ..........  (F ((X 1) Y Z) ((X 1) Y Z) F ((Y 1 X 0) Z)))
U = (0 1 2)  ..........  (F F ((X 2) Y Z) ((Y 2 X 0) F))
U = (0 1 2 3)  ..........  (F F ((X 3) Y Z) ((Y 3 X 1) Z) F))
U = (0 1 2 3 4)  ..........  (F F ((X 4) Y Z) ((Y 4 X 2) Z) F))
U = (0 1 2 3 4 5)  ..........  (F F ((X 5) Y Z) ((Y 5 X 3) Z) F))
(APPLY TABLE (SIMILAR (- (-) ((-)) (- -) (- - -) (- (-)---) (---)
((-- ATOM) - -)) (A () (A) ((A)) (A B) ((A) B) ((A B) C)
((A) B C) (A (B C)) (A (B C) D) ((A B) C D) ((A) B (C D))
(A (B) C D))
))
VALOR
U = A  ..........  (T F F F F F F F)
U = ()  ..........  (T F F F F F T F)
U = (A)  ..........  (T T F F F F T F)
U = ((A))  ..........  (T T T F F F T F)
U = (A B)  ..........  (T F F T F F T F)
U = ((A) B)  ..........  (T F F T F F T F)
U = ((A B) C)  ..........  (T F F T F F T F)
U = ((A) B C)  ..........  (T F F F T F T F)
U = (A (B C))  ..........  (T F F T F F T F)
U = (A (B C) D)  ..........  (T F F F T F T T)
U = ((A B) C D)  ..........  (T F F F T F T F)
U = ((A) B (C D))  ..........  (T F F F T F T F)
U =  (A (B) C D)  ..........  (T F F F F T T F)
(APPLY TABLE (
(LAMBDA (U V) (SIMILAR* U (QUOTE (() X Y Z)) V))
(( X - X) (X --- X) (- X -)
(--- X - X ---) (--- X --- X ---))
((0 1 0) (0 1 2 0) (0 1 2) (3 0 2 0 1 3) (0 1 0 0) (0 1 2 0 2))))
VALOR
U = (0 1 0)  ..... (((X 0) Y Z) ((X 0) Y Z) ((X 1) Y Z) ((X 0) Y Z)((X 0) Y Z
U = (0 1 2 0)  .... (F ((X 0) Y Z) F F ((X 0) Y Z))
U = (0 1 2)  ..... (F F ((X 1) Y Z) F F)
U = (3 0 2 0 1 3)  (F ((X 3) Y Z) F ((X 0) Y Z) ((X 3) Y Z))
U = (0 1 0 0)  .... (F ((X 0) Y Z) F ((X 0) Y Z) ((X 0) Y Z))
U = (0 1 2 0 2)  .. (F F F ((X 2) Y Z) ((X 0) Y Z))
```