CONVERT: Diseno de un lenguaje para manipulacion simbolica de datos y de su procesador correspondiente (CONVERT: Design of a language for symbolic manipulation of data and of the corresponding processor) by Adolfo Guzman Arenas. TESIS presentada para obtener el Titulo de Ingeniero en Comunicaciones y Electronica - 1965 (THESIS submitted to obtain the title of Communications and Electronics Engineer - 1965), Escuela Superior de Ingenieria Mecanica y Electrica (I.P.N.), partial translation (Introduction and Chapters I-IV). Translated from the Spanish (November 1965) by Leo Kanner.

****

INSTITUTO POLITECNICO NACIONAL

c o n v e r t

DESIGN OF A LANGUAGE FOR SYMBOLIC MANIPULATION OF DATA AND OF THE CORRESPONDING PROCESSOR

by

Adolfo Guzman Arenas

THESIS submitted to obtain the title
of Communications and Electronics Engineer

1965

COLLEGE OF MECHANICAL AND ELECTRICAL ENGINEERING

CHAPTER I

SHORT DESCRIPTION OF LISP

INTRODUCTION

CONVERT is a programing language suited for the symbolic manipulation of objects, recursive, operating with lists. It is based on the recognition of patterns, models or structures in an expression and in its corresponding change, modification or substitution; it is capable of carrying out arithmetic operations as well as information input and output commands.

The idea of CONVERT as a programing language is due to Dr. H.V. McIntosh, who began working on it some time ago; the language, and with it the programs, have undergone radical modifications, dictated primarily by the experience gained when working on specific examples. Finally, in February 1965, we presented a "stable" version of the processor, which was implemented in LISP 1.5 of Stanford University, California (Reference 8-4e; this article constituted, before this Thesis, the main source of information on this language), in the IBM 7090 system installed at the Computer Science Center of that university.

Since then, the language has been slowly absorbing modifications and additions which make it -- in my opinion -- more flexible and versatile, until it assumed its present form.

Currently, CONVERT operates in the IBM 709 system of the Instituto Politecnico Nacional de Mexico (National Polytechnic Institute of Mexico) and on the AN/FSQ-32 computer of System Development Corporation, Santa Monica, California. In the latter, it operates through a TELEX link to the Time Sharing System (which provides for the computer to carry out programs of a large number of users simultaneously) of the Q-32.

The CONVERT processor is written in LISP, by which its implementation becomes facilitated in systems already capable of processing this language.

Description of the Thesis

The Thesis consists of 8 chapters. The first chapter briefly describes LISP.

The CONVERT language is described in Chapter II: patterns and skeletons. Some examples will be found in Chapter V, while Chapter IV explains how to operate with the CONVERT system.

Chapter III is devoted to an explanation of the way

the processor operates, i.e. how it meets the linguistic requirements spelled out in detail in Chapter II.

A discussion, glossary and references appear in the last part of this Thesis.

oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo

Translation supplied by

The readers familiar with LISP may omit this chapter. Also described in some detail is MBLISP.

If more complete information is desired, the reader may resort to References 8-1, 8-2 and 8-3.

LISP, originally developed by McCarthy (Reference 8-1a), is a computer language with the following characteristics:

1. Symbolic manipulation of objects. It processes information which generally does not have any numerical meaning and in whose manipulation arithmetic operations are not essential.
2. Recursive. Applies recursive functions to lists. See Section 1-6.
3. Logic.
4. Operates with lists. All LISP notations consist of lists; a list is technically defined as a series of elements contained within parentheses. The elements contained in a list may in turn be lists or atomic symbols. An atomic symbol, the ATOM, is an uninterrupted succession (by parentheses or blank spaces) of characters available in a tape-punching machine. An example of list is ((AB) THREE (SI GA)).
   Examples of atomic symbols are 54321BANG DERIVATIVE
   One (or more) blank spaces separate one atom from another; between an atom and a list or between the latter no blank spaces are necessary.

   The lists are chains of elements with balanced parentheses.
   () is the zero list; it has no elements.

1.1 Expressions.

An expression is an atom or a list. Example: A, (B A (C)), but not C A M I, the latter being a fragment: the content of a list.

If the list L = (E S (I) M E), then the expressions E, (I), S, M, are -- individual -- elements of L, but not I, although I is the [only] element of (I).

1-2 Primitive Functions.

There are five primitive LISP functions as follows:

(CAR X). Its value is the first element of list X, if the value of X is a list. If X = (A B C D), (CAR X) = A.

(CDR X). Its value is the list X without its first element.

If X = (A B C D),
(CDR X) = (B C D). If N = (R), M = (A B C), then
(CDR M) = (B C); (CDR N) = ().

The car and the cdr of an atom are not defined; neither is the cdr of a vacant list. CDR always initiates a list as a value.

(CONS X Y) 1. Y list

Its value is a new list formed by the elements of Y added in front of X. If X = (A B), Y= (C D), then
(CONS X Y) = ((A B) C D) (CONS Y X) = ((C D) A B).

Predicates.

Functions whose value is T (true) or F (false).

(ATOM X) = T if X is an atom; otherwise F.
(EQ X Y) = T if X and Y are both equal atoms; i.e. they are in fact the same atom; otherwise F.

1-3 Other Functions.

Other functions are frequently useful, for which purpose they exist as machine functions, primarily to speed up the computation.

(NULL X) = T if X is the same null list.
otherwise F.

(QUOTE X) its value is its argument (without evaluating).
If X = (A B C D), then (QUOTE X) = X
(QUOTE (1 2 3)) = (1 2 3)

(LIST L1 L2 ...) Its value is a list containing the elements L1, L2, ... in that order. If A = (A B C), B= (C D E),
(LIST A B) = ((A B C) (C D E))
(LIST) = ()

(NOT P) P must be predicated.
= T if P is F
otherwise F.

Composition of Functions.

Assuming that each of the functions described above (except QUOTE) evaluates first its arguments, i.e. it associates with them their value, functions may be composed in

the usual manner; this means in order to evaluate a function, first one evaluates its arguments, and the function is applied to the result(s). Examples. X = (A B C); Y = ((1 2 3) 4)
(CONS (CAR X) (CAR Y)) = (A 1 2 3)
(CDR (CDR Y)) = ()
(CAR (CDR X)) = B
(CAR (CDR Y)) = 4
(CAR (QUOTE M)) is undefined (the car of an atom is not defined)
(NULL (CDDDR X)) = T
(EQ (CAAR Y) (CADR X)) = T
Those of the last examples show the use of the abbreviations (CAAR X), ... (CDDDDR X) for (CAR (CAR X)), ... (CDR (CDR (CDR (CDR X)))).

1-4 Conditional Expressions.

Conditional expressions constitute a means providing for bifurcations in the definition of functions.

(IF P Q R) P predicate; Q and R any functions.
Evalutate P. If P is T, evaluate Q, with the result being the value of the entire expression; if P is not T, evaluate R.
Note: P is always evaluated.
Of Q and R, only one is evaluated, depending on the value of P.

(COND (P1 E1) (P2 E2) ... (Pn En)) P_i predicates; E_i any functions.

If P1 is T, the value of E1 is the value of the entire expression.
If P1 is F, repeat the process on (P2 E2) etc.

The p_i are analyzed (evaluated) from the left to the right until the correct one is found. Then the E_i corresponding function is evaluated, it being the value of the entire expression. No further predicates or expressions are then evaluated.

If none of the P_i is T, the value of COND is undefined.

1-5 Boolean Functions.

(NOT P) = T if P is F
otherwise F.

(OR P1 P2 ...) = F if none of its elements is T
otherwise T

The evaluation of the P_i's is made from left to right; the first one that is T is accepted and no further evaluation is made.

(OR) is always F.

(AND P1 P2 ...) = T if none of its arguments P_i is F.
otherwise F.

It is necessary that all predicates be T for it to be T.

(AND) is always T.

The evaluation of the P_i's is made from left to right; it is assumed that the first is F, with F then being the value of the entire expression.

1-6 RECURSION.

A recursive function is that which is used itself in its own definition, although with different arguments. For example, for natural n, the elevation to the powers X^n may be defined as:

$$x^n = \begin{cases} x \text{ if } n = 1 \\ \text{otherwise } x \cdot x^{n-1} \end{cases}$$

or (EXP X N) = (IF (EQ N (QUOTE 1)) X (TIMES X (EXP X (DECR N)))).
Note that every recursive definition has a terminal condition or one or more final conditions, and one (or more) recursive conditions; hence, after a finite number of steps, the terminal condition must be arrived at.
Other examples: factorial

$$n! = \begin{cases} 1 \text{ if } n = 0 \\ \text{otherwise } n \cdot (n-1)! \end{cases}$$

If X is list the definition of EVEN, one predicate that asks if X has an even number of elements is

$$(\text{EVEN X}) = \begin{cases} \text{T if X = ();} \\ \text{F is X contains a single element;} \\ \text{otherwise (EVEN (CDDR X)).} \end{cases}$$

1-7 LAMBDA. .

An expression of the form x + 2y - 5 is considered a <u>form</u>, i.e. an expression capable of launching or having a value when its variables are identified and associated with values; for example, if

f(x, y) = x + 2y - 5, then [1]
f(1, 2) = 0; f(0, 0) = -5, etc. [2]

in LISP, [1] is expressed by the symbolism
(LAMBDA (X Y) form)

and [2] by ((LAMBDA (X Y) form) A1 A2)
which indicates that X = A1, and that Y = A2.

The value of a function of the type
((LAMBDA (V1 V2 ... Vn) form) A1 A2 ... An) is calculated as follows:

1. A1, A2, ... are evaluated.
2. To each V_i we associate the value of the corresponding A_i.
3. With these values assigned to the variables V_i we calculate the corresponding form.
4. The value of the entire expression is the value given by the form in e.

For example, if U = (S H I R T), V = ((1) (U))
the value of
((LAMBDA (A B) (CONS (CAR A) (CONS V B))) (LIST U V) (CAR V))
is ((S H I R T) ((1) (U)) 1), since

$$A = (\text{LIST U V})_{\text{evaluated}} = ((\text{S H I R T}) ((1) (U)))$$

$$B = (\text{CAR V})_{\text{evaluated}} = (1)$$

1-8 VARIANTS: LAMBDA*, LAMBDA L

((LAMBDA* (V1 V2 ... Vn) form) A1 A2 ... An) the difference of LAMBDA, does not evaluate its arguments, except that it associates V1 with A1, V2 with A2, ... and its form is evaluated.

((LAMBDA L form) A1 A2 ...)

Evaluates its arguments (of which there is an infinite number) and forms a list with the results, to which a value of L is assigned. It then evaluates form, which is a function of L.

For example, the definition of LIST is (LAMBDA L L) [see § 7-1].

((LAMBDA* L form) A1 A2 ...)

Does not evaluate its arguments (of which there is an infinite number) and forms a list with the results, to which a value of L is assigned. It then proceeds to evaluate form, which is a function of L.

Example: definition of VAL (3-53), CONVERTERROR (3-8) ..

1-9 DEFINITIONS

It is possible to associate a number to a function which will allow us, after having defined this number, to

use it (i.e., to use a single atom) instead of the entire list defining the function.

This is done with the postulate DEFINE, in the following manner:
(DEFINE (NUMBER DEFINITION) (NUMBER2 DEFINITION2) ...)
for example,

```
(DEFINE (EVEN (LAMBDA (X) (COND ((NULL X) (AND))
                                 ((NULL (CDR X)) (OR))
                                 ((AND) (EVEN (CDDR X))) )) )
        (EXP (LAMBDA (X N) (IF (EQ N (DEC (QUOTE 1))) X
                               (TIMES X (EXP X (DECR N))) )) )
        (POWER EXP)
                    )
```

defines the functions EVEN, EXP and POWER, with the latter being equivalent to EXP.

The postulate

```
(APPLY  EVEN (     (A B C D E F)       ))
```

apply — function number — list with arguments

"will apply" the even function to the argument (A B C D E F), and the result will be: T.

1-10 MBLISP CHARACTERISTICS

1. INTERPRETER. MBLISP is an interpreter written in machine language (709), of LISP in basic language; the interpreter is very small, occupying approximately 600 words; the entire processor occupies some 4,000 words, with its distribution being approximately as follows:

   600 words for the basic interpreter.
   1200 for atoms and permanent lists.
   600 for the input and output routine.
   1000 additional machine functions.
   600 Garbage Collector, primitives, AND, IF, COND, etc.

   Its main attraction resides in its small size and in the fact of having predicate operators, value cells, LAMBDA*, LAMBDA L; its disadvantage is primarily its speed when compared with a compiler; nevertheless, the A-list of MBLISP is built by a rule -- it is really a rule --, to increase speed to some extent (see § 1-8).

2. ARITHMETIC AND LOGIC OPERATIONS. There is no floating point arithmetic; only integers of 15 bits are

processed (less than 32,768), and with them normal operations can be performed, which will be dealt with as binary numbers (to be used with mascs, etc.) or octals.

Arithmetic operations -- and logics -- are applied to numerals, non-printable atoms; (DEC A), where A is an atom formed by digits, will produce the corresponding numeral; for example, (DEC (QUOTE 1)) will produce the numeral 1; (UNDEC N), where N is a numeral, will produce the corresponding atom capable of being printed.

3. ARRAY ELEMENTS. Very rudimentary; static. Divided into two classes: those which contain lists (or atoms, but not mixed) and those which contain data (octal digits, BDC characters, etc.), and both types of information cannot be mixed within the same array. In this Thesis additional reference to rules will be made in Section 2-14, entitled "Operators in CONVERT."

4. OPERATORS-PREDICATES. Many of these produce T as a value so that they can be manipulated through AND-OR chains; they alter permanently the structure of lists; this means they modify their arguments. Iterative operations may be performed through the ITER function, which continues evaluating its argument until it assumes the value of T (original, page 3-56), and/or by the use of sequential variables (original, page 3-52).

1-11 REFERENCES.

MBLISP is described in greater detail in Reference 8-2b, where many definitions of functions are shown.

The operators are described in OPERATORS FOR MBLISP, Program Note No. 9, University of Florida, 1963.

Arithmetic operations are dealt with in detail in INTEGER ARITHMETIC FUNCTIONS IN MBLISP, Program Note No. 6, University of Florida, 1963.

1-12 PERMANENT ATOMS PRESENT IN MBLISP.

(	Hollerith character left-hand parenthesis.
,	Hollerith character comma.
Z	Alphabetic character.

ZER	(ZER N) is T if its argument is the numeral 0 (zero).
ZERARR	(ZERARR N), where N is a decimal number, creates a rule of N (DEC (QUOTE 0))'s.
Y	Alphabetic character.
X	Alphabetic character.
XDR	(XDR L M) assumes as its value (CDR M), but at the same time originates a transformation in M; its CDR now is L.
XAR	(XAR A M) has the value (CAR M), and at the same time changes the (CAR M) to A.
W	Alphabetic character.
VAL	(VAL X), presently (CAIDR (QUOTE X)) is the VALue (see original, pages 3-52, 3-53) of X, which is not evaluated.
V	Alphabetic character.
U	Alphabetic character.
UNOCT	(UNOCT N) is the octal equivalent of N and is an atom.
UNHOL	Is the hollerith equivalent of its numerical argument, probably an octal number less than 77 and probably a "legal" hollerith character. It has the form (UNHOL N).
UNDEC	(UNDEC N) is the decimal equivalent of its numerical argument and is an atom.
T	Alphabetic character.
S	Alphabetic character.
SYMTAB	(SYMTAB N) gives us certain internal directions (locations), inherent in the MBLISP processor, affecting work in the system.
SWT	(SWT N) examines the switch or interrupter N; it is T if depressed (on).
STOSET	(STOSET Z) initializes the primitive STORE operator to the rule whose title is Z.
STORE	(STORE C) is an operator which stores the octal digit O in the nearest sequential location in the rule to which it has been initialized. The digits are kept sequentially from left to right in a word, and then in words of increasing direction. Their value is T if space remains to store another digit, or F.
STOP	An atom to control the system which causes 3 end-of-file being written out on SYSPOT and PCHTAP, and calls -- selects -- the card reader for the next program.
SL	(SL M N) is T if M is strictly less than N. If M and N are numerics, it questions by algebraic inequality; if one is numeric, it is used as a test for the relative direction of the location and the nucleus with respect to the other, and if both are non-numerics,

	it compares their present locations in the memory, and serves as a list of arbitrary order which may or may not be trusty.
SKIP	Causes the APPLY present to be abandoned, that the temporary registers and the pushdown list be restored and the execution of the next APPLY or system command.
SEQ	(SEQ X) causes the X atom to assume as its value the next sequential y in accordance with the sequencial mode defined. See original, pages 3-52, 3-53.
SEK	(SEK K) is similar to SEQ, however, the sequential value is taken immediately, while for SEQ its value was the old value, and VAL must be used to obtain the new value.
SDR	(SDR L M) causes (CDR M) to become now L; it has a value of T.
SAR	(SAR A M) causes that (CAR M) now become L; it has a value of T.
/	Hollerith character / (diagonal, divide).
(blank)	Hollerith character blank -- blank space --.
*	Hollerith character asterisk.
$	Hollerith character sign of pesos.
$XOR	($XOR M N) calculates the exclusive OR of M and N, probably defined as (OCT X).
$WRITE	($WRITE Z) causes the "buffer" whose title is Z -- this is an array element -- to be entered on the SYSPOT (A-3) tape.
$TIMES	($TIMES X Y) is a function of two values, with first being the more significant part of the product X·Y, and X and Y being numerals.
$READ	($READ Z) reads a SYSPIT (A-2) record in the "buffer" array element whose title is Z.
$PUNCH	($PUNCH Z) causes the content of the "buffer" array element entitled Z to be placed on PCHTAP (B-4).
$PLUS	($PLUS X Y) is a function of two values: the first is a predicate, T if there has been no overflow. The second is a sum module 2**15.
$OR	($OR M N) is OR of M and N, treated as 15-bit binary numbers.
$MINUS	($MINUS X Y) is a function of two values. The first is a predicate, T if the difference x-y is positive; the second is the absolute value of this difference.
$COM	($COM N) is the complement of number N of 15-binary bits.
$AND	($AND M N) is the logic end of the two numbers M and N, 15-binary bits.
R	Alphabetic character.
RPAREN	(RPAREN) produces the hollerith character) value.

RAR	(RAR X L) causes (CAR L) to be converted into X; its value is X, the new (CAR L).
RDR	(RDR X L) causes (CDR L) to be converted into X; its value is X, the new (CDR L).
RANSET	(RANSET) adjusts the RAND operator to its initial value.
RANDOM	(RANDOM) is a predicate which assumes the value of T or F randomly each time it is consulted. It is a pseudo-random operator, which may be re-initialized by RANSET.
Q	Alphabetic character.
QUOTE	(QUOTE E) assumes as its value the argument E without evaluation.
QDR	(QDR X L) causes (CDR L) to be converted into X; its value is L, the new list.
QAR	(QAR X L) causes (CAR L) to be converted into X; its value is L, the new list.
P	Alphabetic character.
PRINT	(PRINT X) is an operator whose value is X, however, which causes the argument X to appear also in SYSPOT (tape A-3).
PAUSE	(PAUSE) causes the computer to perform an HPR; it is a predicate whose value is T.
PACSET	(PACSET Z) is an operator-predicate which assumes the value T and which adjusts the operator PACK to the array element whose title is Z.
PACK	(PACK C) is an operator-predicate which stores the argument C, probably a hollerith character, in the next sequential position in the array element to which it has been initialized. If more space remains, it assumes the value of T.
O	The letter O; alphabetic character.
OR	(OR P1 P2 ... Pn) is a predicate of a variable number of arguments evaluated until a T is found. Those that follow the argument T may be it undefined. If no correct (T) argument is found, its value is F -- false --.
ONLINE	(ONLINE P) is an operator which causes the messages printed destined for SYSPOT to print their last line on the printer connected to the 709.
OCT	(OCT N) is the octal equivalent as a numeral of the atom it has as an argument. Probably this atom is a chain of digits which does not include 8 or 9's.
N	Alphabetic character.
NUM	(NUM X) is a predicate which is T if its argument is a numeral. It is false if its argument is a list or another type of atom.
NULL	(NULL X) is a predicate which assumes the value of T if its argument is an empty list. An

	empty list is not an atom. It is false for any other list or for atomic arguments.
NOT	(NOT X) is a predicate which is false if its argument is true, and it is true if its argument is false. Any other argument remains without change.
M	Alphabetic character.
L	Alphabetic character.
LPAREN	(LPAREN) is a function whose value is the hollerith character "(." Like blanks and right-hand parentheses, they cannot be quoted.
LAMBDA	LAMBDA is a sign used to introduce a function definition in terms of a form.
LABEL	is another signal used to introduce a function definition within the program.
LABEL*	LABEL* is an alternative form of LABEL used when the definition of the function is to be computed, rather than to assume it as quoted.
K	Alphabetic character.
J	Alphabetic character.
-	Alphabetic character, minus sign.
)	Hollerith character, right side parenthesis, which is the value of (RPAREN).
.	Hollerith character period.
I	Hollerith character.
ITER	(ITER P) continues evaluating its argument P, probably a predicated operator, until it assumes the value T. The value of ITER is then T.
INCR	(INCR N) adds 1 to its argument, probably numeric.
IF	(IF P Q R) evaluates the argument P, and if it is T, evaluates the argument Q, otherwise the argument R. The non-evaluated argument may be indefinite.
H G F E	Alphabetic characters.
EVAL	(EVAL E ALIST) evaluates the argument E with respect to the ALIST designated. Gives the programer access to the interpreter.
EQ	(EQ X Y) is T if X and Y are at least one atom and are the same atom. If X and Y are both lists, a T value is trusty but an answer F is not. (EQ X Y) is T if both X and Y have the same representation in the machine (kept in the same location).
D	Alphabetic character.
DO	(DO F A1 A2 ... An) calculates (F A1 A2 ... An), but its value is T. Used to convert an operator into a predicated operator.

DISSOC	(DISSOC) produces the next three bits in the form of an atom and in the range 0-7 of the array element to which it has been initialized. If no octal digits remain, its value is an empty list.
DSCSET	(DSCSET Z) initiates DISSOC with the array element whose title is Z.
DISSET	(DISSET Z) initiates DISINT under the array element whose title is Z.
DISINT	(DISINT) produces the next character of the array element to which it has been initialized. If no characters remain, whether the array element has been terminated or because it has found 77_o, its value is ().
DEFINE	(DEFINE (N E) (N' E') ...) causes the atomic symbols N, N', ... to be considered functions in accordance with definitions E, E', ...
DEC	(DEC N) transforms an atomic argument into a numeral which is its decimal equivalent. Probably N is a chain of characters representing arabic numbers: 1 2 3 4 5 6 7 8 9 0
DECR	(DECR N) subtracts 1 from N, assumed to be a numeral.
C	Alphabetic character.
CONS	(CONS X L) forms a new list adding X in front of L.
COND	(COND P1 F1) (P2 F2) ...) examines sequentially each argument in turn. For the first correct P_i, the associate function F_i is evaluated. If no correct predicate exists, it is an error. Both predicates and functions following the selected pair may be indefinite.
CAR-CDR	combined, up to a length of 4. Also included are the combinations CIDR CIIDR CAIDR CAIIDR; I signifies INCR.
B	Alphabetic character.
BLANK	(BLANK) assumes the value of the blank hollerith character.
A	Alphabetic character.
ATOM	(ATOM X) is a predicate which is correct if its argument is an atom, including numerals as a special case. It is false when applied to any list, including the empty list.
ARRAY	(ARRAY A E) has an argument which generates an array element (e.g., ZERARR) whose title is then considered the value of A. This title is valid with respect to the name A while expression E is evaluated whose value is that of the postulate ARRAY
APPLY	(APPLY F L) is the universal LISP function that applies the function F to its list L of arguments.

AND	(AND A1 A2 ... An) is a predicate which is T if all arguments are correct, including none. The arguments are evaluated sequentially until the first wrong one is found, and then the value is F with the subsequent arguments possibly being indefinite.
ALIST	(ALIST) produces the present ALIST of the system in use. It is primarily used together with EVAL, however, it may also be used as a diagnostic and for analytic purposes.
+	Hollerith character plus.
'	Hollerith character apostrophe.
=	Hollerith character equal sign.
9 8 7 6 5 4 3 2 1 0	The Hollerith characters corresponding to Arabic numbers.
2NDVAL	Operator which destroys the first value of a function of two values.
1STVAL	Operator which destroys the second value of a function of two values.
SEVENTYSEVEN	(SEVENTYSEVEN) has the hollerith character value 77 (non-existent), used to terminate print-names of atoms.
$GARLIS	($GARLIS) is a function of the system which gives us the direction of the garbage-list; this means the list of items salvaged during a garbage-collection. It is used to protect the VALues of atoms.
$DIVIDE	($DIVIDE X Y) has two values, the first being the quotient X/Y and the second the remainder.
REINTEGRATE	(REINTEGRATE L) transforms the argument L which is a list of hollerith characters into the atom formed by the union of all. This atom is inserted or is put in its corresponding spot on the list of all atoms.
NEXTJOB	(NEXTJOB) is a system function which writes an end-of-file onto SYSPOT and onto PCHTAP (A-3 and B-4), passes over the next end-of-file in SYSPIT (A-2), rewinds and reads the SYSTAP (B-7) tape expecting to carry out another LISP operation.
LISPTAPE	(LISPTAPE) is a function of the system which generates a new tape of the system as LSPTAP (B-3).
LAMBDA*	A variation of LAMBDA which previously did not evaluate its arguments.
ENTITLE	The function of the system used to create an appropriate title for a special array element class.
COMMENT	(COMMENT : : :) is not taken into account by the system, thus allowing for the insertion of commentaries in the program.

SYSTEM FUNCTIONS which may be used only at a higher level and which may be re-defined at lower levels as functions, or connected to values such as atoms:

APPLY
COMMENT
DEFINE
LISPTAPE
NEXTJOB
STOP

All the other names which appear in the position of the function name act as a STOP.

NAMES OF FORMS THAT ARE NOT FUNCTIONS

LAMBDA
LAMBDA*
LABEL
LABEL*

CHAPTER II

DESIGN AND DESCRIPTION OF THE CONVERT LANGUAGE

Presently the CONVERT language is being implemented by LISP functions which carry out the necessary operations, by way of an interpretation process, to complete the corresponding transformation.

The principal function is (CONVERT M I E R), which produces a new expression as a value which is the result of modifying expression E according to the changes indicated by the arguments M, I, R.

2-1 FORM OF THE ARGUMENTS

The argument R has the form of a collection -- list -- of sets of rules and their corresponding names; it is a cycle 2 dictionary of the form

(C1 (= = = =) C2 (= = = =) . . . .)
set C1 set C2

where the atoms C_i are the names of the sets to the right; these sets contain rules.

A RULE is a list that contains elements, with the left-hand portion being the pattern (P) and the right-hand portion the skeleton (S).
A PATTERN is an expression used for comparison; it contains certain symbols which are interpreted in a special form; in general, when comparing a pattern with any expression, we obtain T if there is similarity or F if not. The comparison is carried out by the function RESEMBLE.
A SKELETON is an expression used to be filled or substituted; certain symbols here represent "blanks" which will be replaced by expressions.

So that a more accurate view of R is:

R = (C1 ((P S) (P S) (P S) ...) C2 ((P S) (P S) ...) . . .)
first set
number of the second set
first rule of the second set
skeleton of the first rule of C2
pattern of the second rule of C2

The argument M is the first of CONVERT. It has the form of a cycle 3 dictionary; each cycle consists of a NAME, a MODE and a MEMBER.

MODES are atoms indicating the relationship -- the type of relationship -- that exists between the NAME and the MEMBER.
NAMES are atoms or singlets (lists containing one atom) which specify that this atom will be connected

to two quantities, two expressions: MODE and PARTNER.
PARTNER is an expression tied to a NAME.

Consequently, the dictionary M is
M = ($NAME_1$ $MODE_1$ $PARTNER_1$ N_2 M_2 S_2 ...), for example,
M = (A PAV (=OR= 1 2 4 8) (XXX) SKEL (B A C) (YYY) CNT 0).

The argument I is a simple list (cycle 1 dictionary) of NAMES, such as (X Y (ZZZ) V).

The argument E, the third of CONVERT, is the expression we will convert.

2-2 OPERATION

CONVERT applies the first set of rules of R to the expression E, examining sequentially its rules: each pattern is compared with E by way of the RESEMBLE function which determines the way in which this comparison is made and makes use of the information provided by the M and I dictionaries; this information consists of attributing to certain atoms associate properties (PARTNERS), by way of MODES, or by declaring them connectable, i.e. capable of comparing with any expression.

If there is a resemblance, the RESEMBLE value is a dictionary which contains information on the identified sub-expressions of E; then this information is substituted in the skeleton of the rule that was successful by way of the REPLACE function which specifies the form in which the substitutions are made.

If there has been no resemblance between a pattern and the expression E, in which case the RESEMBLE value is F (FALSE), the next rule is applied, and so on until a rule is found whose pattern matches with E; it is then substituted in the corresponding skeleton.

The value of (CONVERT M I E R) is the skeleton, properly substituted, of the rule whose pattern matches with E.

If we arrive at the end of the set of rules, and none of them has been successful, the CONVERT value is the expression E without modification.

2-3 WHAT ABOUT THE OTHER SETS?

R is a collection of sets (see § 2-1). Initially, the

first rule of the first set is examined. The rules belonging to the other sets are only used if they are called upon during an intermediate phase in the conversion process, if an additional conversion of a partial result requires its use. For this purpose the skeletons =REPT=, *CONT*, etc. are used (original, page 2-32).

Furthermore, other skeletons allow us to construct an expression with the data obtained in the resemblance, and to apply the same set of rules where these skeletons are located by which the recursion becomes possible (original, page 2-32).

2-4 INTERNAL CHANGES

At the start of operations, the M and I dictionaries are merged into a single one of cycle 3 by means of ITLZI (original, page 3-5), which is more adequate for being used by RESEMBLE.

Another function, (EDIT L) [original page 3-90], slightly modifies the list which launches RESEMBLE as a value, whenever there exists similarity, into a more useful form to be used by REPLACE.

We shall now describe RESEMBLE and REPLACE, which are the functions which manipulate the left-hand and the right-hand half of the rules.

2-5 (RESEMBLE X L E)

(RESEMBLE X L E) is a function used to recognize patterns by means of which two expressions X and E may be compared. In general, these expressions are simply compared sub-expression by sub-expression, and it is hoped that their constituent atoms are equal and that they are arranged in the same manner so that these expressions would be compared by the EQUAL function of LISP. Nevertheless, certain special symbols may occur in expression X, which indicate that we must interpret the corresponding portions of E in a special way. Furthermore, these symbols or special sub-expressions frequently indicate that a certain expression must occur at various points in the expression E, and we are able to learn the identity of these sub-expressions (of E), which will provide for a similarity between X and E. For this reason, RESEMBLE uses the argument L, which is a dictionary of these sub-expressions that have been identified, as well other special terms. To facilitate the recursive definition of RESEMBLE, this same dictionary, properly modified or increased, assumes its value, whenever there is similarity, or the value F if not.

The pattern X consists of basic patterns which, in turn, may constitute more complex expressions upon being written in the form of lists within lists.

L is a dictionary of cycle 3:
$L = (NAME_1\ MODE_1\ PARTNER_1\ NAM_2\ MOD_2\ PARTNER_2\ \ldots\ldots)$. L defines the names N1, N2, ... , representing the partner S_1, S_2, ... , under the mode M_1, M_2, ... Each time an N_i appears in X, its partner S_i is compared with the corresponding part of E; the way in which this comparison is made depends on the mode M_i.

The definitions made in L are used during the entire process of comparison; there are also ways of making temporary definitions (see =DEF=, original, page 2-14).

The most important modes in RESEMBLE are

UAR Indefinite variable (connectable variable), which compares with anything, after which the mode changes to VAR, with the PARTNER now being the compared quantity.

VAR Defined variable. The expression with which it is compared must be equal to its partner.

PAT The partner is a pattern which is compared against E instead of the atom.

PAV Combines the PAT and UAR modes in the sense that its PARTNER is a pattern compared with E, and if there is a similarity, PAV changes to VAR, and as a partner it retains the E compared.

SUB The partner is a pattern that defines a sub-set of the expression E compared. The mode then changes to SEQ with the pattern ahead of the sub-set it determines.

MSU Same as SUB, except that the mode changes to MSS.

SEQ The CAR of the partner is a pattern which defines a sub-set of the expression E being compared. This sub-set must be the same (the order of the elements is immaterial) as the CDR of the partner.

MSS Same as SEQ, except that the intersection of the sub-set of E and the CDR of the partner replace the CDR of the partner.

These modes are self-explanatory except possibly the last two. The need of these patterns as well as the modes operating with sub-sets was pointed out by Prof. McCarthy, who gave us the example of a program of algebraic simplification, where it is desired to work with a set of coefficients of a term in a polynomial, for example, and assuming that the multiplication is commutative, we would not take into account if these coefficients appear later on in the same order or

rearranged. MSU is a variation on the same theme, where we seek the maximum sub-set of coefficients common to a certain number of terms.

Presently, the dictionary may contain atomic NAMES as well as combinations of lists containing a single atom. These are the ones that give CONVERT its particular character of a processor of chains instead of a processor of lists.

If an atomic NAME is found in the dictionary, it is assumed that it corresponds to a single expression E to which it may or may not be similar. A singlet (list of a single atom) cannot occur alone as a pattern, but only as part of a greater pattern. This greater pattern must be compared with a list where the listed atom -- singlet -- may represent a connected segment of this list, in other words, a FRAGMENT. Fragment variables may occur in the UAR, VAR and PAT modes, among others. In this case [PAT], the symbol represents a fragment of the original pattern, which is inserted as a fragment, rather than as an element. VAR means that the symbol represents a fragment whose elements must be equal to the corresponding elements of E, while UAR means that we are searching for an unknown fragment of expression E.

A fragment variable in the UAR mode forces RESEMBLE to carry out an enormous inspection since, if the same variable occurs more than once, its original estimated value will be continuously revised for all the possible fragments, beginning with the empty fragment up to the first which matches is found (and consequently the shortest one).

In addition to the variables that correspond to fragments or expressions, there exists a number of other fundamental patterns, based on which a more general pattern may be constructed.

2-6 PATTERNS MATCHING EXPRESSIONS

We refer to patterns that compare with a complete expression as differentiated from others which have similarity with parts of fragments of lists. These patterns (which match expressions) may occupy location X in (RESEMBLE X L E), while those which match fragments must appear within X; X must be a list, being unable to have an "independent" existence.

2-6.1 TERMINAL PATTERNS

These are simple patterns that do not require a later

comparison to determine if the expression they are being compared is similar to them.

=*=	Compares with any expression, atom or list.
=ATO=	Compares with any atom (including numeral).
=NUM=	Compares with any numeral
an ATOM not defined in L	Compares with another identical atom.
an ATOM whose mode is VAR	There exists a similarity if its partner and the expression E are equal to each other (under the LISP EQUAL function; original, page 3-17).
( )	Compares only with another empty list.
(=QUO= P)	Similarity is found if P and expression E are equal (under the LISP EQUAL function; orignal, page 3-17). It does not recognize in P symbols with any special significance.
(=DEC= P)	P is transformed into a numeral and then compared by way of EQ and E. P is an atom. It will match E if E is the numeral corresponding to the atom P.

ARRANGEMENT PATTERNS

These are used to discover the size or "order" of a numeral or, if it is a list or an atom, of its corresponding numeral.

=ORD=	Matches with a list whose elements are in an ascending order, or beter, in a non-descending order. For example, (1 2 3) or (1 2 2).
STL L mode = (... A STL 8 ...):	A will match with any atom strictly smaller than [its partner] 8.
STG mode	For example, L = (B STG 9); then A -- in X-- will match with any atom strictly larger than [its partner] 9.

In the STL and the STG modes, the partners are numerals or atoms formed by digits.

The arrangement patterns have their main use in the joint interrogation of numerals or atoms.

EXAMPLES

(RESEMBLE (QUOTE =ATO=) L E) is equivalent to saying (IF (ATOM E) L (OR)).

2. X = (=ATO= A =*=) represents a list with three elements: any atom, the atom A and any expression. For example: (B A (N)) but not ((B) A (N)).

3. X = (=ATO= B (() ==)) represents a list with three elements: the first may be any atom, the second may be atom B, and the third must be a list of elements: the empty list and any expression -- atom or list --.
4. If X = (A B C == A) and L = (A VAR (G O A) B VAR BB), then X will match with ((G O A) BB C D (G O A))
   and with ((G O A) BB C (E) (G O A))
   but not with ((G O A) BB C (E) (F) (G O A))
   neither with (A B C == A).
5. X = (=QUO= (A == C)) will match with E = (A == C) but not with E = (A B C) nor with E = (=QUO= (A == C)).

In items 4. and 5., the result obtained by resemble when similarity is discovered is the dictionary L such as it was originally.

A table of results is shown below.

TABLE OF EXAMPLES OF TERMINAL PATTERNS

in these examples, the dictionary L is (B VAR TANGENT)

X \ E	( )	B	C	(B(A C))	TANGENT	(B ==)	(TANGENT PI)	5	(=QUO= C)
=ATO=	F	T	T	F	T	F	F	T	F
==	T	T	T	T	T	T	T	T	T
=NUM=	F	F	F	F	F	F	F	F	F
B	F	F	F	F	T	F	F	F	F
C	F	F	T	F	F	F	F	F	F
( )	T	F	F	F	F	F	F	F	F
(== ==)	F	F	F	T	F	T	T	F	T
(B ==)	F	F	F	F	F	F	T	F	F
(=QUO(B ==))	F	F	F	F	F	T	F	F	F

For convenience, we marked with T those cases where similarity exists, while in reality the result is (B VAR TANGENT); i.e., the dictionary L. For example, the first line [of =ATO=] would be read:

```
(RESEMBLE (QUOTE =ATO=) (QUOTE (B VAR TANGENT)) (QUOTE ())) = F
(RESEMBLE (QUOTE =ATO=) (QUOTE (B VAR TANGENT)) (QUOTE B))) = (B VAR TANGENT);
(RESEMBLE (QUOTE =ATO=) (QUOTE (B VAR TANGENT)) (QUOTE C))) = (B VAR TANGENT), etc.
```

2-6.2 TRANSITIVE PATTERNS

Atoms whose mode indicates the type of comparison between the partner and expression E. They are used to represent in the source program more complex expressions and/or to give us information regarding sub-expressions of expression E.

To determine their similarity or disparity with expression E, the partner must be examined, and for this reason we can say that they are recursive patterns that compare the partner and the expression under RESEMBLE.

ATOMS REPRESENTING INDEFINITE EXPRESSIONS

When the pattern in which they are contained matches, they change to the VAR mode -- in the dictionary -- and thus provide useful information which will subsequently be used by REPLACE in the skeleton.

Atom whose mode is UAR. If L = (... Y UAR * ...), then Y represents a blank, a space which may be filled arbitrarily by any expression -- atom or list -- but which must be filled systematically by the same expression in the sense that if various Y's appear in the pattern X, the same expression must appear at the corresponding points of E.

In UAR, the PARTNER is unimportant, since Y has no initial associated value; nevertheless, it must be present to conserve the period 3 of the dictionary.

We say that Y is a connectable variable, capable of matching with any expression; once it is matched, it becomes a connected variable changing L into (... Y VAR EE ...), where the partner EE is the expression with which it made connection.

EXAMPLE 1. X = (N == N A S), L = (N UAR *) will compare with E = (P A P A S), where the value of RESEMBLE is (N VAR P), and with E = ((T) (O R) (T) A S), where the value of RESEMBLE is (N VAR (T)), but not with E = (T A B A S), where the value of RESEMBLE is F.

We can say that a connectable variable behaves like ==, except that it "remembers" the value with which it matched.

EXAMPLE 2. If L = (Z UAR ()),
E = (A B), it will match with
X = (== ==) [a list of elements]
but not with X = (Z Z) [a list of equal elements].

Note that (RESEMBLE X L E) does not imply (RESEMBLE E L X); i.e. pattern and expression are not commutative; RESEMBLE only recognizes special symbols in its first argument.

ATOMS REPRESENTING PATTERNS

They allow that an atom represent an entire pattern, lending simplicity to the argument X and providing recursive

definitions (see original, page 2-13, recursive patterns). For example, instead of X = (M (== == ==) N (== == ==) (== == ==)), L = (), we can write X = (M 3 N 3 3), L = (3 PAT (== == ==)).

Atom whose mode is PAT. This atom causes its PARTNER to be compared with E, in search of similarity. This partner is, therefore, a pattern as general as desired.

Example: X = (A C), L = (A PAT (B == B) B UAR **) will match with E = ((N U N) C), having the value (A PAT (B == B) B VAR N), but not with E = ((N U M) C), having the value F, nor with E = (A C).

Atom whose mode is PAV. As in PAT, its partner is a pattern which is compared with E, but, like UAR, if there is similarity, it assumes the value of the expression compared and "remembers" it for future use, changing into a connected variable (VAR mode).

Example: X = (O S O), L = (O PAV (== ==)), E = ((M I) S (M I)), then (RESEMBLE X L E) = (O VAR (M I)).

Example 2: X = (B B), E = ((M A) (M E)), will have similarity under L = (B PAT (== ==)), yielding (B PAT (== ==)) as the value of RESEMBLE, but will not compare under L = (B PAV (== ==)), yielding F as a value.

This means that PAV requires for the first time similarity with its partner, and later equality with the expression with which it was first matched.

PAV requires that the expression with which it is matched be the same in all cases and that, furthermore, it match with the pattern it has as its PARTNER.

EXAMPLES

1. The definition in L of . as (. PAT ==) allows us to use . instead of ==, i.e.,
   X = (A . . . B) is equivalent to (A == == == B), therefore, it will match with C = (A 1 2 3 B), and
   E = (A (A) (B) (C D) B), but not with E = (A (B) (C) E), giving F as a result.
2. L = (X PAV ==) is equivalent to (X UAR *); a connectable variable. Nevertheless, X UAR * is more rapid. See Chapter III.
   For example, it will match with E = (AR AR), if X = (X X), yielding the result (X VAR AR).

Note: Example 1 above will not be correctly realized by CONVERT on a Q-32, since in LISP 1.5 the period (.) has a

special meaning. This is the reason why . was not used instead of ==; furthermore, we prefer to use double symbols, for example, ==, instead of simple symbols (=), since thus we are able to analyze a text consisting of characters.

TABLE OF EXAMPLES OF ATOMS REPRESENTING PATTERNS

X = (P E P A)

	L=(P PAT(= ==))	L=(PAV (== ==))	L=(P PAT(== Y)Y UAR *0
E=((1 2)E(3 4) A)	(P PAT(== ==))	F	F
E=((1 2)E(3 2)A)	(P PAT(== ==))	F	(P PAT(= Y)Y VAR 2)
E=((1 2)E(1 2)A)	(P PAT(== ==))	(P VAR(1 2))	(P PAT(= Y)Y VAR 2)
E=(1 2 E 1 2 A)	F	F	F

	L=(P PAV(== Y)Y UAR *)
E=((1 2)E(3 4)A)	F
E=((1 2)E(3 2)A)	F
E=((1 2)E(1 2)A)	(P VAR(1 2)Y VAR 2)
E=(1 2 E 1 2 A)	F

ATOMS REPRESENTING SUB-SETS

These atoms match with lists, from which they extract the elements similar with their partner; this is a pattern which gives us the criterion to accept or reject an element of the list as a member of the sub-set.

These atoms consider the list with which they match as an unordered set; furthermore, we obtain F if we try to compare it with an atom.

Atom whose mode is SUB. The first occurrence of this atom extracts from the list with which it compares the elements with similarity with its partner; it "recalls" this information and requires that the same elements be obtained in later comparisons (regardless of order); is converted to VAR, whenever this occurs [that the various comparisons produce the same subset]; otherwise a false answer results.

with PARTNER the subset thus obtained

Example: X = (S == == S), L = (S SUB =ATO=) is a pattern which compares with a list whose extremes are sets containing the same atoms.

Applied to
E = (((A) U (B) 2 (C) 3 3) (() 1) RR (3 (M) (A) U 3 (C) 2))
we obtain the result (S VAR (U 2 3 3)).

Atom whose mode is MSU. It operates similar to the previous mode, however, each time it compares, it forms a list with the elements selected, yielding as a value the intersection of these lists. It does not require that the same elements be obtained by different comparisons, except that it accepts the maximum sub-set common to all.

EXAMPLES

ATOMS REPRESENTING SUB-SETS. TABLE OF EXAMPLES.

X = (S O S)

E \ L	(S SUB =ATO=)	(S MSU=ATO=)
(B O B)	F	F
((A B C)O(B A C))	(S VAR(A B C))	(S VAR (A B C))
((A (G)B C(T)C)O((M)A C(G)(T) C B))	(S VAR(A B C C))	(S VAR (A B C C))
((A (G)B C) O(B (N) A D))	F	(S VAR (A B))

2-6.3 COMPOUND PATTERNS

These patterns allow boolean combinations of patterns.

(=OR= P1 P2 ...) For this pattern to have similarity with E, when at least one of the patterns P1, P2, ... must compare with E. The first similarity, together with its identification of sub-expression, is accepted.

(=AND= P1 P2 ...), where P1, P2, ... are patterns. This pattern will compare with E if none of the P_i fails to compare with E: E must be similar to each one of the patterns P1, P2, ... If any variables are identified with expressions, the same definitions must satisfy all the patterns P_i: the dictionary accumulates. Example 1. (=AND=) is equivalent to == [any expression].

(=NOT= P) The pattern P compares with E, and we assume as the value of RESEMBLE the ~~opposite value~~ converse of the resulting value.

Converse
[illegible] value of L = F
Converse [illegible] value of F = present L (dictionary)
Example 1. X = (=== (=NOT= A) ===) will [illegible] compare with a list which has some element different from A.
X' = (=NOT= (=== A ===)) will [illegible] compare with a list which does not contain the element A.

It is assumed that complex patterns can be "simplified" and read more easily if additional patterns are defined in L; furthermore, we can apply the common rules of boolean algebra, as well as the De Morgan theorem.

EXAMPLES.

The pattern which compares with the list containing A or B, but not both, is

```
X = '(=AND= (=OR= (=NOT= (=== A ===)) (=NOT= (=== B ===)))
            (=== (=OR= A B) ===))
```

which is reduced by the De Morgan theorm to

```
X = (=AND= (=NOT= (=AND= (=== A ===) (=== B ===)))
           (=== (=OR= A B) ===) )
```

If we define atoms NA and NB in the L dictionary as NA PAT (=== A ===); NB PAT (=== B ===), the previous pattern converts into

```
X = (=AND= (=NOT= (=AND= NA NB)) (=== (=OR= A B) ===))
```

which may also be written in the form
X = (=AND= NAB (=== (=OR= A B) ===)) with the additional definition NAB PAT (=NOT= (=AND= NA NB)).

Example 2. The use of a connectable variable as the last pattern of an =AND= allows us to recall the expression E which had similarity; e. g.:

```
(RESEMBLE (QUOTE (=AND= (== =ATO= R) Y))
          (QUOTE (Y UAR *))
          (QUOTE (B A R )))  = (Y VAR (B A R)); the same
```

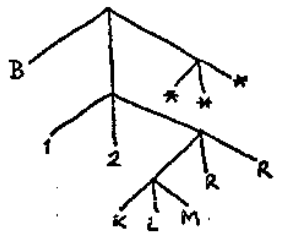
example applied to E = ((B) (A) R) yields F as a result.

TABLE OF EXAMPLES OF COMPOUND PATTERNS

L = (Y UAR *)

X \ E	(G O R D A)	B	( )	(C)
(=AND=(=NOT= =ATO=)Y)	(Y VAR(G O R D A))	F	(Y VAR ( ))	(Y VAR C))
(=OR=(=NOT= =ATO=)Y)	(Y UAR *)	(Y VAR B)	(Y UAR *)	(Y UAR *)
(=AND=(=NOT= =ATO=) (=NOT= C))	(Y UAR *)	F	F	(Y UAR *)

Example. L= (A PAT (=OR= =ATO= (A A A))) defines A as a ternary tree, in the sense that it is a list with three atoms or ternary trees; see illustration.
A will match with E = (B A C)
with E = (B (1 2 ((K L M) R R)) (* * *))
with E = YYYYYYY
with E = (YYYYYY (YY YY YY) POLITECNICO)
[polytechnic]
but not with E = (DOS TACOS) [two wads]
nor with E = (() () ())
The list (B (1 2 ((K L M) R R)) (* * *)) used in this example has the form shown in the figure.

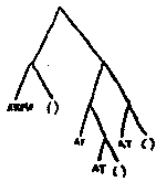


Form of the list (B (1 2 ((K L M) R R)) (* * *)) used in one of the examples.

Example. The cycle K PAT (=OR= =ATO= (=ATO= K =ATO=)) defines K as a tree like that of the figure below; with two lateral branches and a trunk; see illustration, will match with (A (B (C D E) G) N)

Tree represented by the pattern K.

Example. The dictionary
L = (A PAT (=OR= =ATO= (A L)) L PAT (=OR= () (A L)))
defines A (and L) as binary trees in whose left-hand branches we find suspended atoms and on whose right-hand branches we have empty lists. Its shape is shown in the figure. If X = (A L), and E = (A ()), F = (() ()), G = (A B), then
(RESEMBLE X L E) = (A PAT (=OR= =ATO= (A L))
L PAT (=OR= () (A L)))
(RESEMBLE X L F) = F -- false --
(RESEMBLE X L G) = F -- false --



Tree represented by A or L.

With the same L, if X is simply A, this pattern will match with BB, ((A ()) ((K ((M ()) (N ())))) ())), but not with (), ((A B) (C D)). This subject of trees and recursive patterns will be dealt with in greater detail in Reference 8-4c.

2-6.4 PATTERNS DEFINING PATTERNS

(=DEF= R P Q) means that Q must compare with E, but that during the comparison R PAT P is added to L, so that within the Q pattern R represents the pattern P.

If Q is omitted, P is used instead, while, if several definitions are carried out, the last element is compared with E, which will have all the other inputs connected as patterns in L. R may remain in parentheses indicating that it is a fragment which is being defined.

These definitions are temporary in the sense that they are only valid in P'; nevertheless, they have priority, therefore =DEF= may be used to re-define temporarily variables in the dictionary.

=DEF= only defines NAMES in the PAT mode.

2-7 PATTERNS COMPARING WITH FRAGMENTS

These patterns, since they match fragments, cannot exist independently, but must appear as elements of a list.

In the L dictionary, the atoms represent expressions or fragments according to whether their names are alone or contained in parentheses -- singlets --. For example, A VAR (1 2) represents list (1 2); (A) VAR (1 2) represents the fragment 1 2. See NOTE 1 in the "VAR mode." (p 2-15)

It is customary to give the fragment a name consisting

of 3 equal letters: XXX, ZZZ, although, in reality, the important thing is that the fragments appear in L as a singlet and the expressions as simple atoms.

2-7.1 TERMINAL PATTERNS

Terminal patterns do not require later comparisons to determine if the fragment with which they are compared has similarity.

===	Compares with any portion of a list.
VAR mode	An atom that appears in L as a singlet in the form (XXX) VAR (A B C) represents the fragment A B C, with which it will match via EQUAL. Example: X = (XXX T XXX), L = ((XXX) VAR (A (B) C)) will match with E = (A (B) C T A (B) C). NOTE 1. Just as list (A B C) may be represented by X VAR (A B C), "technically" it can be conceived that the fragment A B C can be represented in the dictionary L as XXX VAR A B C, however, due to the fact that the latter representation would destroy period 3 of the dictionary, we have agreed to enclose between parentheses the name and partner: (XXX) VAR (A B C) under the assumption that ~~in the intelligence,~~ if XXX = A B C then (XXX) = (A B C) As we see, the partner of a fragment in the VAR mode must be a list.
(*QUO* P)	Means that P will be dealt with as a fragment of X instead of an expression; comparing the content of P by equality. *QUO*, like VAR, is used to quote ATOMS which otherwise would have a different meaning, whether because they are special or because they appear in L. P must be a list. Example. (*QUO* (XXX XXX)), if L = ((XXX) VAR (C Z)), means XXX XXX, and not C Z C Z. *QUO* is of little use.

EXAMPLES

[See next page for first example.]

Example 2. The definition in
L = (... FIRST PAT (=OR= A (FIRST ===)) A PAV =ATO=).
FIRST extracts the first atom from a list at any depth; the result will appear as A VAR ABC
↳first atom found.

TABLE OF EXAMPLES OF TERMINAL FRAGMENTS

L = ((YYY) VAR (G ==N)

X \ E	(B G R M)	(A G K N K O B)	(A YYY K O B)	(A(YYY K O)B)	A	( )
(=== A ===)	F	T*	T*	T*	F	F
(=== YYY ===)	F	F	F	F	F	F
(A(#QUOM(YYY K O))B)	F	F	T	F	F	F
(===)	T*	T*	T*	T*	F	T*
(===(===)===)	F	F	F	T*	F	F

Note: T* = ((YYY) VAR (G == N))

2-7.2 SUBSTITUTIONAL PATTERNS

Substitutional patterns are atoms that occur as singlets in the L dictionary; their partner is a list; the mode indicates the form in which these atoms are connected with the content -- i.e. with a fragment -- of that list.

ASSOCIATED FRAGMENT

The phrase "associated fragment" indicates "the content of the partner," assuming it to be a list; for example, if L = ((XXX) PAV (A M T)), the associated fragment with XXX is A M T.

ATOMS REPRESENTING INDEFINITE FRAGMENTS

These atoms may be compared initially with any fragment; if successful, they "stabilize," and their mode changes to VAR.

They are used to extract information from E: the final length of the fragment of E which compares with one of these atoms is indefinite, being determined only by the matching or similarity of the other parts of the pattern. At least this similarity requires that the lengths of the compared fragments be modified.

UAR mode. They appear in the dictionary as (XXX) UAR (). Like ===, they may match with any fragment, but once one of them has been found, they retain this value, and VAR changes; this means that the appearance of various XXX

in a pattern signifies that there will be several equal fragments in E located in the corresponding places.

Tentatively, if we assign to a fragment UAR a length zero (empty fragment), and if this value fails, i.e., if this selection does not cause the remainder of X to match with the remainder of E, the length increases, and the comparison continues, etc., until a match is found. By this procedure, the solution found is the minimum solution (the shortest fragment).

If the partner is not an empty list, but one that contains some elements, for example, (XXX) UAR (A B), then XXX will match with a fragment whose initial elements are those of the associated fragment.

CNT mode. Has the form (XXX) CNT ATOM; like ===, it will match with an arbitrary number of any elements such that the remainder of the pattern and of the expression match; if the same atom occurs several times, it is required that the number of matched elements be the same in each case, although in general the elements will differ from one case to the next.

The partner is an atom (formed by digits), which indicates to us the minimum length the fragment must have.

CNT "counts" the elements with which it matched. After a match has been successful, CNT changes to REP and then [to be used by REPLACE] to EXPR.

NOTE: Actually, the transformation continues to (XXX) REC num instead of (XXX) REP num; rec is a mode of internal use. For all practical effects it is equal to REP, the difference being that the REP modes do not produce an output in REPLACE (they do not retain the value of the compared fragments) and the REC modes appear as XXX EXPR ATOM, where (1) XXX is no longer a singlet but an atom and (2) its partner is an atom -- formed by digits -- which indicate the length of the fragment with which it matched.

CNT is one of these rare cases where a fragment has as a partner an atom and

which, when it is used, is employed as an expression and not as a fragment.

TABLE OF EXAMPLES OF INDEFINITE FRAGMENTS

L=((XXX) UAR ( ) (YYY) UAR ( ) (ZZZ) UAR (A) (CCC) CNT 0)

X \ E	(A R)	((A)R)	(G A M)	(B O A)	(A R B O L)	((G)(O)(M))
(XXX)	XXX = A R YYY = vacío (ZZZ) UAR(A) (CCC) CNT 0	XXX = (A) R YYY = vacío ZZZ UAR (A) (CCC) CNT 0	XXX = G.A M YYY = vacío (ZZZ) UAR (A) (CCC) CNT 0	XXX = B O A YYY = vacío (ZZZ) UAR (A) (CCC) CNT 0	XXX = A R B O L YYY = vacío (ZZZ) UAR (A) (CCC) CNT 0	XXX = ((G)(O)(M)) YYY = vacío (ZZZ) UAR (A) (CCC) CNT 0
(XXX YYY)	XXX = vacío YYY = A R (ZZZ) UAR (A) (CCC) CNT 0	XXX = vacío YYY = (A) R (ZZZ) UAR (A) (CCC) CNT 0	XXX = vacío YYY = G A M (ZZZ) UAR (A) (CCC) CNT 0	XXX = vacío YYY = B O A (ZZZ UAR (A) (CCC) CNT 0	XXX = vacío YYY = A R B O L (ZZZ) UAR (A) (CCC) CNT 0	XXX = vacío YYY = (G)(O)(M) (ZZZ) UAR (A) (CCC) CNT 0
(XXX A YYY)	XXX = vacío YYY = R (ZZZ) UAR (A) (CCC) CNT 0	F	XXX = G YYY = M (ZZZ) UAR (A) (CCC) CNT 0	XXX = B O YYY = vacío (ZZZ) UAR (A) (CCC) CNT 0	XXX = vacío YYY = R B O L (ZZZ) UAR (A) (CCC) CNT 0	F
(ZZZ)	XXX = vacío YYY = vacío ZZZ = A R (CCC) CNT 0	F	F	F	XXX = vacío YYY = vacío ZZZ = A R B O L (CCC) CNT 0	F
(ZZZ B YYY)	F	F	F	F	XXX = vacío YYY= O L ZZZ = A R (CCC) CNT 0	F
(CCC)	XXX = vacío YYY = vacío (ZZZ) UAR (A) CCC VAR 2	XXX = vacío YYY = vacío (ZZZ) UAR (A) CCC VAR 2	XXX = vacío YYY = vacío (ZZZ) UAR (A) CCC VAR 3	XXX = vacío YYY = vacío (ZZZ) UAR (A) CCC VAR 3	XXX = vacío YYY = vacío (ZZZ) UAR (A) CCC VAR 5	XXX = vacío YYY = vacío (ZZZ) UAR (A) CCC VAR 3

Legend -- translated: (1) Tree; (2) Empty.

The table is highly schematic; for example, the first square must be interpreted as follows:

(PESEMBLE (QUOTE (XXX)) (QUOTE ((XXX) UAR ( ) (YYY) UAR ( ) (ZZZ) UAR (A) (CCC) CNT 0))
(QUOTE (A R))) = ((XXX) VAR (A R) (YYY) UAR ( ) (ZZZ) UAR (A) (CCC) CNT 0)

Example 2. If XXX is an indefinite fragment, X = (=== XXX B XXX), E = (O B A B O B B A B O), then the process of comparing X with E is as follows:

1. X (=== XXX B XXX)
   E (O B A B O B B A B O)

2. X (=== XXX B XXX)
   E (O B A B O B B A B O)

The arrow ↕ indicates the part that does not match.

```
*. X  (---   XXX   B   XXX )
   E  (0  B A B 0  B  B A B 0)
```

with the result ((XXX) VAR (B A B 0)). See Chapter III on notation of observers.

FRAGMENTS REPRESENTING PATTERNS

Their partner is a list whose content is treated like a [fragment] pattern. They match with a fragment of E which has similarity with the fragment partner.

PAT mode. The associated fragment compares with the corresponding fragment of E under the rules of RESEMBLE; it may be said that, in X, the atom whose mode is PAT is replaced by the associated fragment, and this new X is compared with E.

REP mode. Its partner is a list whose CAR is a pattern (an expression, not a fragment) which repeats as many times as indicated by the CADR of the partner. Example. (XXX) REP (A 8) is equivalent to the fragment A A A A A A A A.

Furthermore, it is assumed that (YYY) REP 5 means == == == == ==.

PAV mode. Like PAT, its associated fragment must match -- under resemble -- with the corresponding fragment of E; if similarity is found, PAV "recalls" the matched fragment, changing to VAR, so that various occurrences of the same atom will compare with the same fragment. This information is then used by REPLACE.

EXAMPLES

1. L = (... (XXX) PAT (== B ===) ...),
   X = (XXX A XXX). Then X is equivalent or similar to
   X = (== B === A == B ===), and therefore will compare with
   E = (P B 1 2 3 A G B 7 5).

2. L = ((YYY) REP ((===) 3)).
   Then X = (A YYY B) will compare with
   E = (A (1) (D D) (3 (E)) B) but not with
   E = (A 1 D D 3 (E) B), the result being
   ((YYY) REP ((===) 3)).

3. L = ((XXX) PAV (== B ===))
   X = (XXX A XXX) will compare with (P B 1 2 3 A P B 1 2 3) but not with E = (P B 1 2 3 A G B 7 5) [see example 1], nor with (1 2 3 A 1 2 3).
4. (...) PAT (===) is another way of saying ===.
5. (XXX) PAV (===) is equivalent to (XXX) UAR (). Nevertheless, this cycle provides for greater speed -- see PAV mode, original, page 3-39.

GENERAL COMMENTARY ON SUBSTITUTIONAL-FRAGMENT PATTERNS

RESEMBLE manipulates a pattern which contains fragment patterns, substituting in the pattern its name for the content of its partner, i.e. for its associate fragment, and then making the comparison.

For example, if L = ((XXX) PAT (A == YYY) (YYY) PAT (===))
X = (XXX B XXX)
then X is changed to (A == YYY B A == YYY), hence to
(A == === B A == ===)
and then compared. It can be said, therefore, that the singlets represent fragments in the source program.

This is generally acceptable; nevertheless, certain precautions must be made. For example, if
L = ((XXX) PAT (=DEF= A B C))
X = (XXX D F)
then X is considered (=DEF= A B C D F)
wherefore A PAT B
C PAT D
is added to L, and F is compared with E.

Another equally logical interpretation, but which resemble does not carry out, is that (XXX) PAT (=DEF= A B C) represents the content of expression C during whose evaluation A is the pattern B; a content which is added to X as a value or replacement of XXX. Resemble does not work this way.

2-7.3 COMPOUND PATTERNS

These patterns provide for boolean combinations of fragment patterns.

Note that: I. -- Although their form is a list, they are compared with fragments, for which reason they cannot appear alone as a pattern except within a greater list which is pattern X.

II. -- P_i are expressions, not fragments, whose content will be compared.

(*OR* P1 P2 ... Pn) Compares with a fragment if this fragment compares at least with one of the contents of P1, P2, ...

(*NOT* P1) Represents a fragment which has no similarity with the content of P.

(AND P1 P2 ... Pn) This expression will compare with a fragment if the same fragment compares simultaneously with content of (all) the P_i's.

(*AND* P1 P2 ... Pn) X will be compared with E. Content within X will be compared if there exist in E fragments which match with the content of P1, the content of P2, ... , so that the remainder of X matches the rest of E.

The difference of AND, which is a "strong" AND, the *AND* does not require that the fragment against which it is compared be the same for all the P_i's.

EXAMPLES

1. X = (=== (*AND* (A) (B)) ===), a list containing one A and one B, compares with (T A B A N O S) [translation: horseflies], but would not do so if it were AND; nevertheless, X = (XXX (*AND* (A) (B)) ===), with (XXX) UAR (), will not compare, since XXX fixes the length of the fragment to the left of the (*AND* ...).

2. (=DEF= (EVEN) ((*OR*() (== == EVEN))))
   note the ((

   defines a fragment of even length; in this form recursive patterns may be defined which represent or match with fragments; the same result would be obtained if we add the cycle (EVEN) PAT ((*OR* () (== == EVEN))) to the dictionary L.

3. The statement (EVEN) PAV ((*OR* () (== == EVEN))) in L will also define a fragment of even length, and will retain the value of the final fragment (called the major), since it is a recursive pattern.

TABLE OF EXAMPLES FOR THE EVEN PATTERN FRAGMENT

X = (EVEN R EVEN)

E	(ejemplo 2) ① EVEN definido como PAT	(ejemplo 3) ② EVEN definido como PAV
(U N O R U N O) ③	F	F
(T R E S R T R E S) ④	(EVEN) PAT ((*.....))	(EVEN) VAR (T R E S) ⑥
(T R E S R S E I S) ⑤	(EVEN) PAT ((*.....))	F

Legend -- translated: (1) (example 2) EVEN defined as PAT: (2) (example 3) EVEN defined as PAV; (3) (O N E R O N E); (4) (T H R E E R T H R E E); (5) (T H R E E R S I X); (6) (T H R E E).

RECURSIVE PART OF RESEMBLE

More complex expressions are compared requiring that the (CAR X) present similarity with (CAR E), and at the same time (CDR X) present similarity with (CDR E). Nevertheless, bear in mind the comment regarding substitutional fragment patterns, original, page 2-19.

Next we will present two tables, one of mnemonics for the modes used in RESEMBLE, and another of the approximate values of RESEMBLE and REPLACE. On page 2-22 of the original we find a table containing all the existing patterns. Furthermore, on pages 3-47A and 3-47B of the original we find the details of the interconversion of the modes in RESEMBLE; page 3-90 of the original gives the definition of EDIT which converts the dictionary headed by REPLACE whenever there exists similarity to the D used by REPLACE. See also, in this regard, section 2-8, page 2-23 of the original.

APPROXIMATE VALUES

FUNCTIONS	Value at first approximation
(RESEMBLE X L E)	(EQUAL X E)
(REPLACE D S)	S
special symbols are recognized	

2-7.4

MODES IN RESEMBLE

TABLE OF MNEMONICS

UAR	indefinite variable
VAR	definite variable
PAT	pattern
PAV	pattern and variable
REP	repeat
CNT	count
SUB	sub-set
MSU	minimum sub-set
STL	strictly less
STG	strictly greater
SEQ	internal use
MSS	internal use
WAR	internal use
REC	internal use

TABLE OF PATTERNS IN RESEMBLE

PATTERNS COMPARING WITH EXPRESSIONS	PATTERNS COMPARING WITH FRAGMENTS
Terminal Patterns	
==	===
=ATO=	VAR mode -- singlet --
=NUM=	(*QUO* P1)
Atoms not in L	
()	
(=QUO= Ps)	
(=DEC= A+)	
=ORD=	
STL mode	
STG mode	
VAR mode	

TABLE OF PATTERNS IN RESEMBLE (cont.)

PATTERNS COMPARING WITH EXPRESSIONS		PATTERNS COMPARING WITH FRAGMENTS	
Substitutional Patterns			
UAP mode	represent blanks	UAR mode	CNT mode
PAT mode, PAV mode	represent patterns	PAT mode, REP mode	PAV mode
SUB mode, MSU mode	represent sub-sets		

Compound Patterns

(=OR= P1 P2 ...)	(*OR* P1 P2 ...)
(=AND= P1 P2 ...)	(*AND* P1 Ps ...)
(=NOT= P1)	(*NOT* P1)
	(AND P1 P2 ...)

Patterns Defining Patterns

(=DEF= N1 P1 N2 P2 ... P')

2-8 USE OF R E S E M B L E AND R E P L A C E IN CONVERT

Each one of the rules forming the sets of the argument R of CONVERT is compared by way of RESEMBLE with expression E -- the third argument of the CONVERT function -- in search of similarity; during this comparison, RESEMBLE uses as a dictionary L a mixture -- prepared by ITLZI, original, page 3-5 -- of M and I, where the latter is modified by adding to its elements the UAR mode and the partner ().

The similarity of the pattern and the expression cause RESEMBLE to yield as a value a dictionary [L*] enriched with information originating from the comparison. This dictionary is changed into another D, renaming the modes in L* to others suitable for REPLACE; with this new dictionary, we substitute in the second part the skeleton of the rule which matched, to form the value of CONVERT.

In general, the VAR modes -- those which originally were declared VAR by the programer and those which were converted to VAR, for example, UAR -- are converted to the EXPR mode to be used by REPLACE. The function which makes this change is EDIT, described on page 3-90 of the original.

EXAMPLE

The numbers refer to the schematic of the following page. Let us assume that we want to "rotate" a list about its first A; i.e., (C A B A L L O) [horse] will be converted into (B A L L O A C); (G O R D A) [fat] into (A G O R D), and in general (--- A ...) → (... A ---).

It is precisely this rule which we use ① in our program, where we declared ② to XXX and YYY as indefinite fragments.

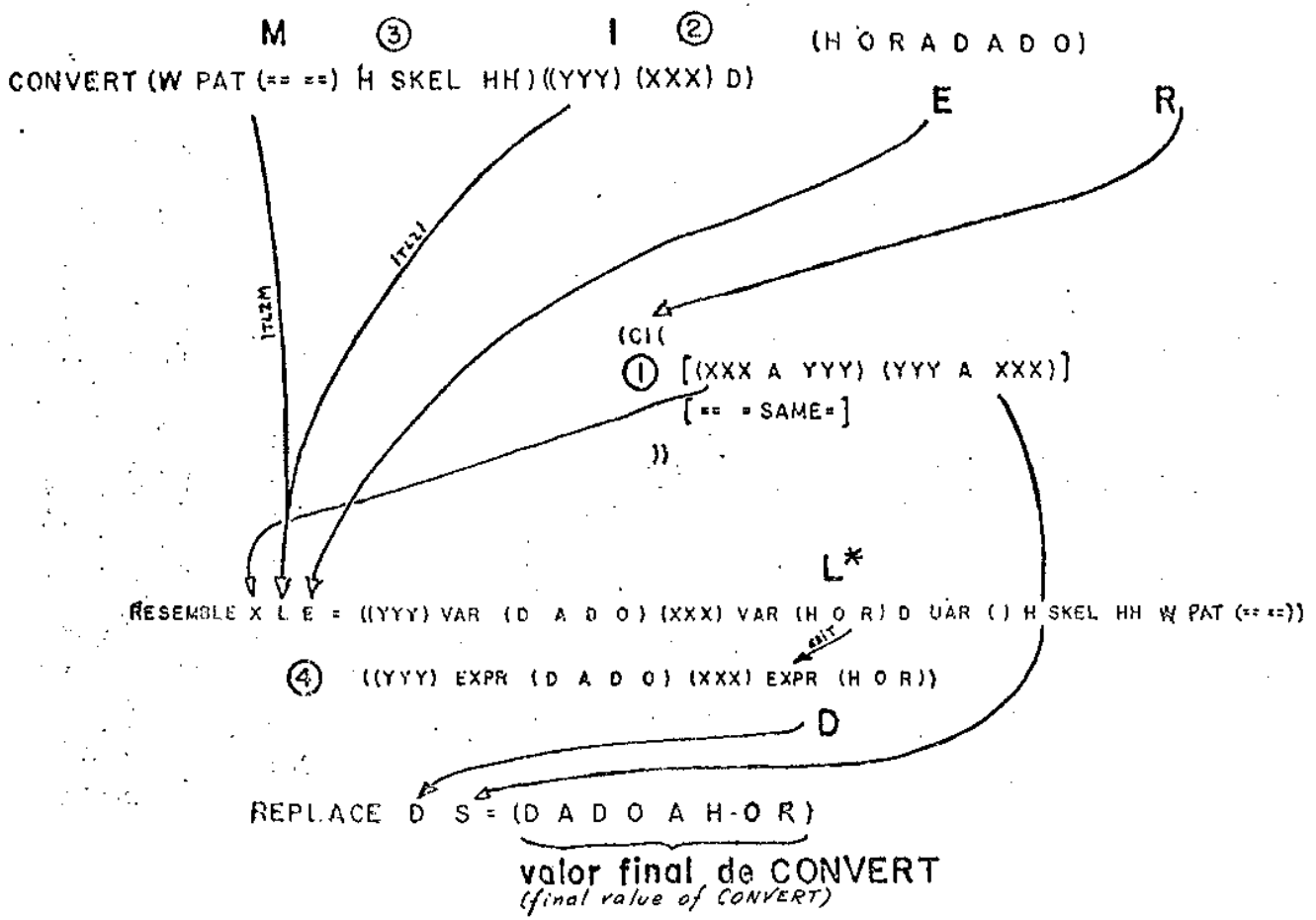
The variables D ② , H and W ③ of M are not used in the program, their presence being merely illustrative.

The transformation is carried out as follows:

1. ITLZI converts to I and to M in the dictionary
   '((YYY) UAR () (XXX) UAR () D UAR () W PAT (== ==) H SKEL HH)
2. This dictionary is used by RESEMBLE, which, in order to compare it with E = (H O R A D A D O) [given hour], uses the pattern (XXX A YYY), of the first rule of the set C1.

Assuming that there is similarity between (XXX A YYY) and (H O R A D A D O), RESEMBLE yields as the value of the list
((YYY) VAR (D A D O) (XXX) VAR (H O R) D UAR () W PAT (== ==)
H SKEL HH)
of which EDIT removes the "garbage collection" and transforms to ④
((YYY) EXPR (D A D O) (XXX) EXPR (H O R) H SKEL HH)
This list is used by REPLACE to make the substitution in (YYY A XXX), the skeleton of the rule which was successful, obtaining (D A D O A H O R) as a skeleton already "filled" and consequently as the final value of CONVERT.

(== =SAME=), the rule two of the single set of R, says: "if anything else is seen in E it returns it as a value," however, this is unnecessary, because CONVERT, after exhausting all the rules of C1 without finding similarity, would return as a value the expression E without change. In fact, therefore, only the rules indicating change are worth mentioning.


M ③ I ② (H O R A D A D O)
CONVERT (W PAT (== ==) H SKEL HH) ((YYY) (XXX) D)
E
R
(CI(
① [(XXX A YYY) (YYY A XXX)]
[== =SAME=]
))
L*
RESEMBLE X L E = ((YYY) VAR (D A D O) (XXX) VAR (H O R) D UAR ( ) H SKEL HH W PAT (== ==))
④ ((YYY) EXPR (D A D O) (XXX) EXPR (H O R))
D
REPLACE D S = (D A D O A H-O R)
valor final de CONVERT
(final value of CONVERT)

2-9 (REPLACE D S)

(REPLACE D S) is a substitution function which modifies its second argument, S, according to the information provided by dictionary D.

In general, the value of REPLACE is S, with the same atoms, except those that appear in the dictionary, which are replaced by their partner, and by some special ones recognized by REPLACE.

If an atom is found which is the name of a fragment, the associate fragment is inserted in S in a similar manner.

Consequently, if we have
D = (X EXPR 1 Y EXPR 2 (XXX) EXPR (A B C)), S = (X XXX Y R), we would have (REPLACE D S) = (1 A B C 2 R) and with S = '(A X I O M A XXX Y R E G L A), the result would become (A 1 I O M A A B C 2 R E G L A).

Nevertheless, it may happen that it is not convenient to describe a transformation in a single step, and that we would want to use different parts of an expression, or perhaps to build an expression, to analyze it and to form another whose form would be much more difficult to describe or predict directly.

For this reason, the skeleton may contain code expressions indicating how these variations are to be made in the construction.

2-10 SKELETONS PRODUCING EXPRESSIONS

These skeletons are the ones which, upon being properly substituted, give an expression as a value. These skeletons may occupy the place of S in (REPLACE D S); in contrast with skeleton fragments, they need not go into a greater skeleton.

2-10.1 TERMINAL SKELETONS

These change expressions without using REPLACE recursively.

=SAME= By means of this symbol, we are able to refer to the expression being transformed without the need to assign to it a name as a variable.

Atoms that do not appear in L. They are substituted by themselves; this means they are not substituted. They represent themselves.

Atoms under the EXPR mode. An atom with EXPR mode is substituted by its partner.

() The empty list is substituted by () -- another empty list --.

(=QUOT= S1) Its value is S1, as it stands. No later substitutions within S1 are made.

Skeletons producing Symbols.

=LPAR= whose value is the character (

=RPAR= whose value is the character)

=BLNK= whose value is the character blank -- blank space --

=GNSY= whose value whenever consulted is a new atom (on the 709 it is an atom numeral)

(=INTG= C1 C2 ...) whose value is the atom obtained when joining the chain of hollerith characters C1, C2, ... replaced

=DISI= whose value whenever consulted is the next sequential hollerith character taken from the array element to which it has been previously fixed (or initialized) [with *DIST*].

=DISO= whose value, whenever consulted, is the next and sequential octal digit taken from the array element previously initialized -- with *DOST*, Section 2-14 -- .

EXAMPLES

D = (A EXPR (B O N) B EXPR BB),
S = (A R B O () (L (E (S))) (=QUOT= A)) then
(REPLACE D S) = ((B O N) R BB O () (L (E (S))) A). Note that the value is not ((BB O N) R BB O () (L (E (S))) (B O N)); i.e. there are no later substitutions in the partner of an atom-EXPR, nor within =QUOT=.

Example 2. The substitution of D = (R EXPR I T EXPR L) in S = (=INTG= C A R T 5) yields as a result the atom CAIL5.

2-10.2 SUBSTITUTIONAL SKELETONS

The final value substitutional skeletons acquire upon being replaced requires the recursive use of REPLACE.

Atom in the SKEL mode. An atom with the SKEL mode in L represents a skeleton: its partner. Consequently,

the replacement will be carried on its partner, as if it had been written instead of the atom in the S skeleton.
Example: D = (A SKEL (B B) B SKEL N M EXPR (A N A))
S = (A M A B A N)
will yield ((N N) (A N A) (N N) N (N N) N);
meanwhile, if in D A would have the EXPR mode, the result would be
((B B) (A N A) (B B) N (B B) N).
Note: We must avoid confusion between an expression [i.e. an S expression: an atom or a list] and an atom with the EXPR mode as well as between a skeleton (an expression to be "filled" or substituted by REPLACE) and an atom under the SKEL mode.

2-10.3 SKELETONS DEFINING SKELETONS

They provide temporary skeleton definitions; in contrast with atoms appearing in D, representing valid definitions at a distance from the entire S skeleton, the definitions made with these modes are valid locally.

(=QUOT= N1 S1 N2 S2 ... Nm Sm S') Substituted in skeleton S', but with the new definitions N_i EXPR S_i added to the dictionary D.

(=EXPR= N1 S1 N2 S2 ... Nm Sm S') Substituted in skeleton S', but with the new definitions N_i EXPR S_i replaced added to the dictionary D.
We use the present dictionary D to substitute in the S_i; we match the resultant expressions with the corresponding N_i and the EXPR mode to increase our dictionary D, which is the one we use to substitute in S'.

(=SKEL= N1 S1 N2 S2 ... Nm Sm S') Substituted in skeleton S', but with the new definitions N_i SKEL S_i added to the dictionary D. These definitions are valid only during the evaluation of S', but have precedence over those of dictionary D, if by chance any of the names N_i becomes repetitive. We can thus change locally the meaning of the names.

The names may be atoms or singlets; in the latter case, we will be defining skeleton fragments.

S' may be omitted; in this case, the replacement will occur on Sm. Note that S' may be any skeleton; as a particular case, S' may be another (=EXPR= ...).

A summary of what has been explained above appears in the following table.

CASE	INSTANT WHEN SKELETONS S1, S2, ... ARE REPLACED	DICTIONARY USED FOR REPLACEMENT
=QUOT=	never	none
=EXPR=	at the moment of replacing the skeleton (=EXPR= ...)	the one in force upon replacing the skeleton (=EXPR=...)
=SKEL=	at the moment of using N_i since they pass into the dictionary as SKEL	the new; the one enriched with the temporary definitions

EXAMPLES

D = (H EXPR (1 2 3)), S = (H H (=EXPR= H (A H)) H)
results: ((1 2 3) (1 2 3) (A (1 2 3)) (1 2 3))
if instead of =EXPR= it were =QUOT=, the result would be
((1 2 3) (1 2 3) (A H) (1 2 3))
and if were =SKEL=, ((1 2 3) (1 2 3) (A (A (A ...))) (1 2 3))
infinite list

the result would have as its third element an infinite list, since H has been defined as the skeleton (A H).
"PUSH DOWN LIST EXHAUSTED".

Example 2. D = (J SKEL (A B C))
S = (J (=SKEL= A B B C (B A J O)) J)
results: ((A B C) (C C (A B C)O) (A B C))
if instead of =SKEL= it were =EXPR=, the result would be
((A B C) (C B (A B C) O)) (A B C)).

2-10.4 ARITHMETIC OPERATIONS

REPLACE has a set of skeletons which produce arithmetic operations and take the value of the result; they are of the form (=PLUS= S_1 S_2 ... S_n). Their value is always a numeral. In all of them, we first replace the S_i's, and then the operation is performed. In general, the S_i (already replaced) must be numerals; on the 709, this is satisfied if the S_i's are atoms in their last value to which has been applied a =DECM=; in Q-32 LISP, they must be atoms beginning with a digit, ., +, -. The numbers take the base 10.

On the 709, the operations are on whole numbers, mod 2^{35}; in Q-32 there are whole numbers and floating point, depending on the form of the S_i's.

It must be recalled that, in MB LISP, a numeral lacks written representation; it is "unprintable," wherefore, in order to see its value (i.e., if the final value of a computation contains numerals) it is necessary to convert them to ordinary atoms by means of the =UDEC= skeleton.

On the Q-32, in turn, the atoms which have the form of numbers (see Reference 8-3b, page 87) are converted by the input routine, wherefore it is not necessary to make a =DECM= and a =UDEC= on them; they are recognized by the printing routine and converted to their corresponding written representation; therefore, in Q-32 CONVERT, a numeral is an atom number, for example, 8.0, -32, etc., and the atom corresponding to a numeral or the numeral corresponding to an atom coincide if the atom in question has a numeric format (formed by figures, signs, etc). The following table shows a summary of what has been discussed.

CASE	709	Q-32
ATOM ordinary NUMERAL	(=DECM= ATOM) = = NUMERAL	=DECM= and =UDEC= are treated in the Q-32 as no-operation; their value is the argument without change.
NUMERAL ordinary ATOM	(=UDEC= numeral) = = ATOM	

In Q-32 there is coincidence between S, (=DECM= S) and (=UDEC= S); this distinction is necessary on the 709.

(=DECM= A) where A is an atom. The value is the numeral corresponding to A. In general, A has the form of an atom formed by digits (whole). In -Q-32, real numbers are also permissible. Example: D = (BORR EXPR 45); S = (=DECM= BORR). Result: $45_{numeral}$

(=UDEC= N) where N replaced is a numeral. The value of this skeleton is the atom corresponding to N. Example: D = (BORR EXPR 23), S = (=UDEC= (=DECM= BORR)); Result: $45_{(atom)}$

(=PLUS= X1 X2 ...) The sum of all the replaced arguments is formed. After the replacement, it is assumed that all the arguments are numerical. On the 709, the sum is calculated mod 2^{15}. In Q-32, the sum is whole or real, depending on the form of the addends. Its value is a numeral.

(=MINS= X Y) The difference X-Y is calculated, after first replacing X and Y. Its value is a numeral. On the 709, the difference is always calculated in absolute values.

(=TIMS= X1 X2 ...) The product of the replaced values of the arguments X1, X2, ... is formed. The overflow is neglected. The result is a numeral.

(=POWR= Y X) We calculate X raised to the power of Y; after first replacing X and Y. Its value is a numeral.

(=DIVD= X Y) The quotient X/Y is formed. First X, Y are replaced. The result is a numeral.

(=REMN= X Y) The residual of the quotient X/Y is formed. First X and Y are replaced. The result is a numeral.

There are also predicates in RESEMBLE which ask for the presence of a numeral, its sign, asking for 0, etc. For example, the pattern =NUM= is used to match with a numeral, in the same way as (===) compares with a list, or == compares with any expression, or =ATO= matches with an atom, or 0 matches with the atom 0, or (=DEC= 0) matches with the numeral 0, etc.

We also have =ORD= (page 2-6 of the original), which will compare with a set whose members -- assumed to be numerals -- are in an increasing order, and the modes STL (strictly less) and STG (strictly greater), which compare to the expression E with the partner, both supposed to be numerals.

2-10.5 OPERATIONS ON SETS

Although not logically necessary, since they can be synthesized with the aid of the appropriate CONVERT program, it is convenient to have available the functions common in the theory of sets. They are, together with their definition, the following. In each case their arguments are replaced before the execution. They all yield lists as their value.

(=COMP= A B) Forms a list of the elements of A not belonging to B.

(=INTS= A1 A2 ... An) A list of all those elements common to A1, A2, ... is formed. If an element is found repeated k times at the intersection, this means that it occurred k times, but no more in each one of the sets A1, A2, ...

(=UNON= A1 A2 ...) Forms a list of different elements that appear in the sets A1, A2, ... In particular, (=UNON= A) will contain the same elements as A, but each element will appear only once, regardless of its multiplicity in A.

(=CONC= A1 A2 ...) A list of all the elements which appear in the sets A1, A2, ... is formed. =CONC= is an APPEND multiple; in other words, the sets are joined simply in the order in which they are listed, so that the elements can be repeated an arbitrary number of times according to their appearance in one or more of the sets A1, A2, ...

(=CART= A1 A2 ...) The Cartesian product of the arguments A1, A2, ... is formed so that the result is a list of x items, where the first element belongs to A1, the second to A2, etc.

2-10.6 RANDOM SKELETON

There has been a certain number of requests to introduce a random variable in LISP. For purposes of program testing, error tracing, etc. it is preferable to use a pseudo-random variable, and a convenient way of achieving this is the process of raising to the square a large number, causing the selection to depend on one of the central bits, and to store the central bits of the square and then repeat the process. In the absence of something more sophisticated, we have implemented this technique in the function (RANDOM) of MBLISP, which produces the values T or F according to the above. Consequently, it is a predicate and, when used as an argument of an IF, it will produce a "random" selection among the two final arguments of IF. It may be initialized by the predicate operator (RANSET), which is always T if it is desired to repeat the same sequence of [its] values.

With the aid of this function, we can also have a binary random selection in CONVERT.

(=RAND= A B) A pseudo-random selection is made of its two arguments A and B, replacing them the one chosen.

This random skeleton may be used to generate examples of patterns which depend on an =OR= or an *OR*. For example, a binary tree satisfies the pattern (=DEF= B (=OR= =ATO= (B B))). To obtain an example of a binary tree, the skeleton (=SKEL= B (=RAND = * (B B))) may be used. At any instant, the same probability of writing the atom * or constructing a pair of elements which obey the same construction rule exists.

If we desire a random selection other than purely binary, various binary selections may be superimposed so that the distribution of the frequency desired is obtained or an approximation of it for its present values.

Hence, if we write (=RAND= (=RAND= A B) (=RAND= C D)), we expect to obtain the variables A, B, C, D, each with a probability of 1/4.

2-10.7 INPUT AND OUTPUT SKELETONS

(=PRNT= S1) If we write (=PRNT= S1) we obtain the substitute skeleton, but also its value is printed on the output tape. It is useful if we want to obtain any type of intermediate results.

2-10.8 RECURSIVE SKELETONS

This set of skeletons deals with the possibility of constructing an intermediate result which, however, is transformed by another set of conversion rules.

(=BEGN= S1) Having replaced skeleton S1 to form a new expression, the entire conversion applies to it. This means that we return to the first set of rules of argument R of CONVERT, and we begin to apply its rules to the superseded S1. All the variables return to their original state of definition or non-definition.

(=REPT= S1) Having evaluated S1 by means of REPLACE, the present set of rules -- in use -- is applied to this new expression. All the variables return to their initial state. Nevertheless, the definitions introduced by =QUOT=, =EXPR= or =SKEL= are retained.

(=CONT= S1) Having replaced S to form a new expression, the set presently in use is applied to it, however, the variables presently defined are preserved. This means that in the rules of S1, in patterns and skeletons, the values of variables found by RESEMBLE remain valid: it means that the definitions of dictionary D of REPLACE continue valid; we do not return to the variables in their original state given by M and I -- first and second arguments of CONVERT --, as we do with =BEGN= and =REPT=.

(=REPT= S1 K) K is the number of a set of rules used instead of the present set; otherwise its behavior is the same as without its argument K.

(=CONT= S1 K) By the same token, the set K is used instead of the set presently in use.

(=REPT= S1 K1 R1 K2 R2 ... Kn Rn) In this case, the dictionary (K1 R1 K2 R2 ...) is APPENDed (added) to the dictionary of sets of rules, and the conversion continues with the set K1, i.e. R1.

(=CONT= S1 K1 R1 K2 R2 ... Kn Rn) is the same version for =CONT=.

Note that in all cases the skeleton S1 is always substituted by the present dictionary (dictionary D), and to the resulting value (*) we apply the corresponding set of rules, preserving dictionary D or returning to the original dictionary, depending on whether =CONT= or =REPT=-=BEGN= is involved.

(*) The value of a skeleton is the expression (or fragment) which results from replacing this skeleton in accordance with dictionary D in the rules of REPLACE. A summary of the above is found in the table below.

TABLE OF RECURSIVE SKELETONS, DESCRIBING THEIR PROPERTIES

SKELETON	SET TO BE USED	DICTIONARY TO BE USED	AFFECTED BY =EXPR= *SKEL* etc.	AFFECTED BY =ITER=
(=BEGN= S1 ...)	first set of dictionary R	initial	NO	NO
(=REPT= S1 ...)	the set presently being used or the one whose name it invokes		YES	
(=CONT= S1 ...)		present		YES

Note: The forms shown with * instead of = obey the same rules.

The last column corresponds to the comment on page 6-4 of the original.

2-10.9 ITERATIVE SKELETONS

They allow us to substitute a skeleton expression E several times, and where E contains variables whose range and domain are specific sets.

(=ITER= V1 S1 V2 S2 ... E) S1, S2, ... are skeletons whose value is a set (a list), let
(A1, A2, A3, ...) be the value of S1
(B1, B2, B3, ...) be the value of S2, etc.
V1, V2, ... are skeletons whose value is an atom, such as R1, R2, ...
E is a skeleton (not a fragment) to be substituted which contains [in general] the atoms R1, R2, ...
The value of (=ITER= ...) is a list of all the elements resulting from the substitution in E,
R1 by A1, A2, A3, ...
R2 by B1, B2, B3, ...
.....
without omitting nor repeating any.

The order (from left to right) in which they appear in this list is obtained by varying more rapidly the most internal variable (closest to the expression E), while the most external variables undergo slower changes.

Each variable varies about (takes values of) its set to the right.

Assuming that E is any skeleton, other symbols that form it and which are not R1, R2, ... will be replaced according to the normal rules of REPLACE, using dictionary D; for example, (=QUOT= R1) will not be replaced.

=ITER= carries out an evaluation (i.e. a replacement) of all the expressions to its right; therefore, it is incorrect to assume that every occurrence of V1 in E will be replaced by members of S1; what really occurs is that every occurrence in E of the atom [R1] which results from the evaluation of V1 will be replaced by elements of the set which will doubtlessly result upon replacing S1.

In fact, =ITER= temporarily adds to dictionary D the cycles R_i EXPR A_i, R_2 EXPR B_i, etc., hence these definitions have precedence in the case of R1, R2, etc. existing in the dictionary.

EXAMPLES

```
D= (A EXPR J S EXPR (B A A C B D) )
S= (=ITER= A (1 2 3) K (=INTS= S (G A B)) (J * K * A) )
result: ( (1 * A * (1 2 3)) (1 * B * (1 2 3)) (2 * A * (1 2 3))
          (2 * B * (1 2 3)) (3 * A * (1 2 3)) (3 * B * (1 2 3)) )
```

Note that it is not A which assumes the values 1, 2, 3, but J; J is the value of A.

Interactions on Whole Numbers.

If the values of the S_i's are numerals -- instead of lists --, this numeral will be taken as the upper mark of the range of the corresponding R_i, causing R_i = 1, 2, 3, ... S_i (inclusive) to vary. This means that S_i is a numeral which tells us how many times the iteration is carried out. The values assumed by R_i are numerals.

```
Example: D = (R SKEL (=DECM= 3) 0 SKEL J PRO SKEL (=TIMS J K))
S = (=ITER= 0 (=DECM= 2) K R ((=UDEC= K)  BY (=UDEC= J)  IS
                                          (=UDEC= PRO)))
result: ((1 BY 1 IS 1) (2 BY 1 IS 2) (3 BY 1 IS 3)
         (1 BY 2 IS 2) (2 BY 2 IS 4) (3 BY 2 IS 6) )
```

Note that the most internal variable varies very rapidly; K in this case.

Although the previous example would also run in Q-32 CONVERT, there we could simplify
D = (R SKEL 3 0 SKEL J PRO SKEL (=TIMS= J K))
S = (=ITER= 0 2 K R (R BY J IS PRO)) presumably with the same result.

Linear variations of another type are obtained easily by a skeleton.

```
Example 1.
       D = (L SKEL (=UDEC= (=PLUS= (=DECM= 105 (=TIMES=
                                               (=DECM= 3) J))))
       S = (=ITER= J (=DECM= 4) L)
       result: (108 111 114 117)

Example 2. [Only for Q-32]
       D = (L SKEL (=TIMS= J J))
       S = (=ITER= J 5 L); result = (1 4 9 16 25)
```

2-10.10 OPERATORS OF DATA MOVEMENT

Used to manipulate array elements.

(=STOR= C)
(=PACK= O) Described in Section 2-14, "Operators in CONVERT."

2-11 SKELETONS PRODUCING FRAGMENTS

The value of these skeletons is a fragment, wherefore they must appear as elements of a list, a more general pattern.

Replace, as we have seen, has atoms with a special

significance, such as =CONC= or =ITER=. When these appear in a skeleton of the form (=NAME= . . .) and where =NAME= is one of these symbols, this means that the previous list [the previous skeleton] will produce as a value an expression, while if the skeleton is of the form (*NAME* . . .), then the previous list is a fragment skeleton, and will produce as a value a fragment, the content of the expression that would be produced by =NAME=.

For example, (=UNON= (A B C) (D E A)) = (A B C D E)
and (*UNON* (A B C) (D E A)) = A B C D E

Consequently, to learn the fragment skeletons, it is sufficient to remember the expression skeletons and to exchange = by *.

In general, the following identity is realized
((*NAME* ....)) ≡ (=NAME= ....)

2-11.1 TERMINAL SKELETONS

Substituted by a fragment without using REPLACE recursively.

SAME Its value is the content of expression E which matched the pattern of the rule of which *SAME* is a part.
E must be list; otherwise an error message will result.

(*QUOT* S1) Its value is the content, without replacement, of S1, which must be a list.

SKELETONS PRODUCING SYMBOLS

(*DESN* A) whose value is the fragment consisting of the chain of hollerith characters which form the name (the print-name) of the atom A, after being replaced. A, after being replaced, must not be a numeral or a list.
Example: S = (A (*DESN* ELECTR) (*QUOT* (O N I C)) A) will be converted into (A E L E C T R O N I C A) [A electronics].

EXPR mode (fragments). If an atom figures as a singlet in the dictionary D, under the EXPR mode, it is then substituted by its associate fragment without being followed by a later replacement in it.
Example. D = ((XXX) EXPR (A R G) G SKEL GGGG)
S = (S A S E (S) I) → (A R G A A R G E (A R G) I)
The partner of an EXPR mode is considered as quoted material; no special symbols are recognized there.

2-11.2 SUBSTITUTIONAL SKELETONS

These are atoms found with singlets in the dictionary D under the

SKEL mode. The atom is replaced in the skeleton by its associate fragment to form a new skeleton, which is replaced by REPLACE. In the majority of cases, the value of an atom such as SKEL is the value of its corresponding associate fragment; this means, the result of substituting D in the associate fragment. There are "rare" cases where CAR affects CDR. See examples.
In EXPR as well as in SKEL, the partners must be lists.

EXAMPLES

1. D = ((XXX) EXPR (A R C) (YYY) SKEL (B XXX O))
   S = (YYY YYY) has as a value (B A R C O B A R C O).
   [shipship]

2. D = ((XXX) SKEL (A B C) A EXPR AA)
   S = (B XXX A C) will convert into (B AA B C AA C)

3. D = (R EXPR J (XXX) EXPR (=ITER= R (B A C)))
   S = (XXX (R J)) has the value of (=ITER= R (B A C) (J J))

   however, if we had defined XXX as SKEL, the result would be ((J B) (J A) (J C), since a fragment in the SKEL mode must first be added to the skeleton S and then the substitution must be made; this means that it substitutes for the intermediate skeleton
   S = (=ITER= R (B A C) (R J))

   Conclusion: as in RESEMBLE, in REPLACE the fragments are also fragments of the source skeleton; this means they are first substituted to form a new object skeleton.

4a. D = ((XXX) EXPR (=EXPR=) A EXPR AA B EXPR BB)
   S = (XXX A B (B A R C A)) converts into (=EXPR= AA BB (BB AA R C AA)), i.e. the value of XXX is =EXPR=

4b. With D = ((XXX) SKEL (=EXPR=) A EXPR AA B EXPR BB), the result would be (BB BB R C BB), since it will substitute in the skeleton
   (=EXPR= A B (B A R C A))
   i.e., D defines XXX as an equivalent of the symbol =EXPR=

4c. With D = (XXX SKEL =EXPR= A EXPR AA B EXPR BB)
   the result would be (=EXPR= AA BB (BB AA R C AA))
   since XXX is defined as the expression skeleton =EXPR= (not a fragment skeleton), and since this does <u>not</u> affect the CDR of the S.

4d. With D = ((XXX) SKEL (=EXPR=) A EXPR AA B EXPR BB), we obtain the same result as in 4c.

Conclusion: when it is desired that part of the list affect the remainder to its right, one must define it as a fragment in the SKEL mode.

2-11.3 SKELETONS DEFINING SKELETONS

They are the version in fragment of their corresponding expression skeletons =QUOT=, =EXPR= and =SKEL=. The explanations below will, therefore, be particularly short. It is to be recalled that these definitions are "temporary," valid only in S'. Recall also that in a list of the form

$$(\begin{matrix}*\text{QUOT}*\\ *\text{EXPR}*\\ *\text{SKEL}*\end{matrix}\ N1\ S1\ \ldots\ \mathbf{S}')$$

if S' contains (=BEGN= E), the definitions $N_i = S_i$ will be forgotten upon returning to the first set and their application to E; nevertheless, if S' contains =REPT= or =CONT= in any one of their forms, the definitions $N_i = S_i$ prevail: they are "remembered."

(*QUOT* N1 S1 N2 S2 ... Nm Sm L') makes N_i EXPR S_i
Its value is the content of L' replaced (under the new dictionary). L' may be omitted. Note": S_i without replacement.

(*EXPR* N1 S1 N2 S2 ... Nm Sm L') makes N_i EXPR $S_{i\text{replaced}}$
Its value is the content of L' replaced under the new dictionary; L' may be omitted.

(*SKEL* N1 S1 N2 S2 ... Nm Sm L') makes N_i SKEL S_i
Its value is the content of L' replaced (under the new dictionary); L' may be omitted.

2-11.4 SKELETONS CONVERTING EXPRESSIONS INTO FRAGMENTS

(*FRAG* S1) S1 is a skeleton which after replacement converts into a list; the value of (*FRAG* S1) is the content of that list.

The following identity is satisfied:
(*NAME* A1 A2 ...) ≡ (*FRAG* (=NAME= A1 A2 ...))

2-11.5 SKELETONS OPERATING ON SETS

They are the version in fragment of its corresponding expression skeletons (original, page 2-30). Observe for greater detail.

(*COMP* S1 S2) Content of the complement of $S1_{replaced}$ and $S2_{replaced}$

(*INTS* S1 S2 ... Sn) Content of the intersection of S1, S2, ... previously replaced.

(*UNON* S1 S2 ... Sn) Content of the union of S1, S2, ... previously replaced.

(*CONC* S1 S2 ... Sn) Join S1, S2, ... previously replaced in a large list, with the value of (*CONC* ...) being the content of this list.

(*CART* S1 S2 ... Sn) Content of the Cartesian product of S1, S2, ... replaced.

EXAMPLE. D = ((XXX) EXPR ((A B) (C D)))
(*UNON* XXX (A F C)) = A B C D F

2-11.6 OPERATORS TO CONSTRUCT LISTS

Completely described in Section 2-14, "Operators in CONVERT."

Their value is an empty fragment.

(*STAR* X L)
(*STDR* X L)

2-11.7 INPUT AND OUTPUT OPERATORS

Completely described in Section 2-14, "Operators in CONVERT."

Their value is an empty fragment.

(*READ* Z)
(*WRIT* Z)
(*PNCH* Z)

2-11.8 RECURSIVE SKELETONS

Recursive skeletons apply the various sets of rules to intermediate results and take the content of the value obtained as a value. Before applying the corresponding set of rules, the skeleton S1 is replaced by the dictionary D.

(*BEGN* S1) We apply to S1 the first set of rules with all the variables returning to their initial state. This transformation must produce a list as a result, whose content is the value of (*BEGN* S1).

(*REPT* S1 K1 R1 K2 R2 ... Kn Rn) We apply to S1 the set K1, i.e. R1; furthermore, we define the sets K2 = R2, ..., Kn = Rn which may be called upon from any of the rules of R1. This application is made with the initial dictionary of variables, however, remembering those defined by ≠QUOT≠, ≠EXPR≠ and ≠SKEL≠; this produces a list as a value, whose content is the value of (*REPT* ...)

(*REPT* S1 K) may be considered as abbreviations of the general form
(*REPT* S1)

(*CONT* S1 K1 R1 K2 R2 ... Km Rm) We apply to S1 the set K1, i.e. R1; furthermore, the sets K2 = R2, ..., Km = Rm are defined to be called upon under rule of R1.

In R1, all the definitions existing in dictionary D are retained, i.e. all the variables connected by RESEMBLE plus those initially existing in M plus those defined by =QUOT=, *SKEL*, etc.

This application must produce as a result a list whose content (fragment) is taken as the value of (*CONT* ...); if this application produces an atom, it is in error.

(*CONT* S1 K1) may be considered as abbreviations of the general form
(*CONT* S1)

2-11.9 ITERATIVE SKELETONS

(*ITER* V1 S1 V2 S2 ... E)

V_i skeletons which upon substitution will produce atoms.
S_i skeletons which upon substitution will produce sets (lists).
E expression to be replaced.

The value of (*ITER* V1 S1 ... E) is a fragment whose elements are the result of substitution in E for each occurrence of $V_{i\text{replaced}}$, an element of the corresponding set $S_{i\text{replaced}}$.

(*ITER* ...) is the content of the list which will be produced by (=ITER= ...). See for more details the description of =ITER=, original, page 2-33, Section 2-10.9, and page 3-75. A table of the effects of *ITER* and =ITER= is found in the original, page 2-33; a new implementation of ITERA is discussed in Chapter VI, page 6-4.

2-11.10 FUNCTIONAL SKELETONS

Let us assume that a symbol has the meaning of a function name when introducing the additional modes CONT and REPT. If we see in the CONVERT program the skeleton (F A1 A2 ... An) (Comment on R 2-45)

TABLE OF SKELETONS IN REPLACE

SKELETONS PRODUCING EXPRESSIONS		SKELETONS PRODUCING FRAGMENTS
Terminal Skeletons		
=SAME= (=QUOT= S1) EXPR mode Atoms not in D () =LPAR= =RPAR= =BLNK= =DISI= =DISO= =GNSY= =INTG=	produce symbols	*SAME* (*QUOT* S1) EXPR mode (*DESN* A+)
Substitutional Skeletons		
SKEL mode		SKEL mode
Skeletons defining Skeletons		
(=QUOT= N1 S1 N2 S2 ... S') (=EXPR= N1 S1 N2 S2 ... S') (=SKEL= N1 S2 N2 S2 ... S')		(*QUOT* N1 S1 N2 S2 ... S') (*EXPR* N1 S1 N2 S2 ... S') (*SKEL* N1 S1 N2 S2 ... S')
Input and Output		
(=Prnt= S1)		(*READ* Z) (*WRIT* Z) (*PNCH* Z)
Arithmetic Operations		
(=DECM= A+) (=UDEC= Nu) (=PLUS= N1 N2 ...) (=MINS= N1 N2) (=TIMS= N1 N2 ...) (=DIVD= N1 N2)		

2-11.10

TABLE OF SKELETONS IN REPLACE (cont.)

SKELETONS PRODUCING EXPRESSIONS	SKELETONS PRODUCING FRAGMENTS
Arithmetic Operations (cont.)	
(=REMN= N1 N2) (=INCR= N1) (=DECR= N1)	
Which convert Expressions to Fragments	
	(*FRAG* S1)
Operations on Sets	
(=COMP= S1 S2) (=INTS= S1 S2 ...) (=UNON= S1 S2 ...) (=CONC= S1 S2 ...) (=CART= S1 S2 ...)	(*COMP* S1 S2) (*INTS* S1 S2 ...) (*UNON* S1 S2 ...) (*CONC* S1 S2 ...) (*CART* S1 S2 ...)
Random Element	
(=RAND= S1 S2)	
Operators for the Construction of Lists	
	(*STAR* X L) (*STDR* X L)
Recursive Skeletons	
(=BEGN= S1) (=REPT= S1 ...) (=CONT= S1 ...)	(*BEGN* S1) (*REPT* S1 ...) (*CONT* S1 ...)
Iterative Skeletons	
(=ITER= V1 S1 V2 S2 ... S')	(*ITER* V1 S1 V2 S2 ... S')
Operators of Moving Data	
(=PACK= C) (=STOP= O)	(*DIST* Z) (*DOST* Z) (*PCST* Z) *STST* Z)
Functional Skeletons	
	REPT mode CONT mode

where F has been declared a singlet in the REPT mode:

(F) REPT ((P1 S1) (P2 S2) ...)

we consider this combination as being equivalent to

(=REPT= (A1 A2 ... An) * ((P1 S1) ...))

The most common case, of course, is that where the set of rules which is the PARTNER of F in the declaration of modes consists of a single rule. This, in addition to being equivalent to a certain type of LAMBDA where the name and the arguments of a function are identified, permits us to use the functional notation rather than the rule notation.

CONT Mode. The CONT mode produces a similar substitution, using =CONT=. F must be declared as a fragment, in the form (F) -- in the dictionary -- since otherwise it loses its effect on CDR.

On the other hand, the value of (F A1 A2 ...) is always an expression, so that *FRAG* must be used explicitly if it is desired that the value be a fragment.

2-11.11 OPERATORS OF DATA MOVEMENT

Operators of data movements work on array elements. Their value is the empty fragment. They are described in detail in Section 2-14, entitled "Operators in CONVERT".

(*DIST* Z)
(*DOST* Z)
(*PCST* Z)
(*STST* Z)

2-12 RECURSION

If the skeletons are more complex than those described above, it can be said that a substitution is made in the CAR of the skeleton and in the CDR of same, with the results being combined with a CONS. Note the examples where the CAR has an influence on the CDR; pages 2-36, 2-37, original.

2-13 SUMMARY

Depending on how a sequence of conversion rules is searched, the function RESEMBLE provides as a value either the atom F or a dictionary which contains values for the different variables which have been recognized. Some of the variables, those which are in the PAT or STL mode, for example, may be ignored. Nevertheless, those which correspond to subsets or variables may be used to replace in the right-hand half the skeleton of the rule. Consequently, the dictionary which results as a value of RESEMBLE is edited somewhat, and is

converted into the argument D of REPLACE. The second argument S is a skeleton, built with a list of primitive skeletons.

As in RESEMBLE, atomic symbols may represent skeletons; those which are going to be used for this purpose are contained in the dictionary D, cycle 3. The modes are now:

SKEL, which means that the PARTNER of this atom is a skeleton, in which substitutions will be carried out ultimately.

EXPR, which means that the PARTNER of this atom is an expression which is simply copied in the skeleton instead of the original atom.

The designations for modes now contain four letters instead of three, which indicates that a distinction is being made between skeletons and patterns. The argument M of CONVERT may contain definitions which are only effective as skeletons, as well as those which are used as patterns. For a similar reason, the primitive patterns are designated by 3-letter symbols (or less, with OP being the exception), while the skeletons are designated by combinations of four letters. In both cases, framing them with = means expressions, while an * means a fragment.

Furthermore, the dictionary D may contain expressions as well as fragments, with the distinction being made by placing the NAME between parentheses.

The addition carried out between the application of the left-hand half of the rule and its right-hand half includes a transformation of the modes VAR, SEQ, MSS, PAV, CNT into the EXPR mode. See pages 3-47a, 3-47b, original.

2-14 OPERATORS IN CONVERT

An operator differs from a skeleton in that, in addition to having a value, i.e., to being replaced by something, it modifies its arguments, so that these are not the same before and after the skeleton-operator has been replaced. As in LISP, in CONVERT the operators are powerful, but they require care when being used, since they permanently alter the expressions.

ARRAY ELEMENTS

Although not an integral part of the language, the so-called array elements exist in CONVERT. These are sets of words or locations contiguous in the memory, where we can store information.

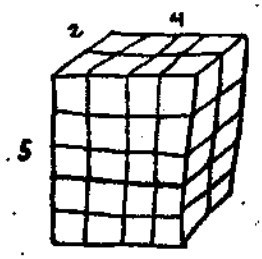
The information an array element may contain is (1) hollerith characters and octal numbers, i.e., data, or (2) atoms and lists, i.e., structures. The mixture of both types within the same array element is not admissible.

Generation of Array Elements which will Contain Data -- In the dictionary M of CONVERT, the cycles

```
A DARR (75)
B DARR (5 4 2)
```

will generate an array element A of 75 words, monodimensional, and another of 40 words, tridimensional, which we will call B:

DARR
array element
for data;
pseudo-mode.



array element B

Next, the pseudo-mode DARR changes to EXPR:

```
A EXPR titleA
B EXPR titleB
```

where $title_A$ and $title_B$ are the titles of the array elements A and B.

The information contained in the previous paragraph is not necessary; it suffices to reflect that A and B are array elements, so that, for example, (*DIST* A) will initialize =DISI= on array element A.

The array elements that contain data are used with the operators to generate symbols, to move data or for input and output. These array elements are n-dimensional, and are protected by the garbage-collector.

A 709 word is equivalent to 6 hollerith characters or 12 octal digits.

How to Generate Array Elements which Contain Expressions Found in the M dictionary of CONVERT, the cycles

```
R LARR (63)
U LARR (30 5)
```

will generate an array element R, monodimensional, of 63 words, and another, two-dimensional, called U, of 30 x 5 words; both are capable of containing structures. Next, the pseudo-mode LARR changes to REPT, so:

(R) REPT rule-set$_R$
(U) REPT rule-set$_U$

LARR
array element
for lists;
pseudo-mode.

where the partner is the selection function for the corresponding array element (note the change of atom to singlet), so that, in order to find element I, J of the array element U, it is sufficient to say (U I J). Similiarly, (R 18) is a skeleton whose value is the list and the atom guarded in word number 18 of array element R.

The changes

DARR → EXPR
LARR → REPT

are made automatically by the processor.

If I say (U) without subscripts, a numeral will result: the title of the array element.

To Store Expressions in the locations of this array element, use skeletons

(=KEEP= S U 15 4) its value is skeleton S, replaced
(*KEEP* S U 15 4) its value is an empty fragment

which store in array element U, in its word (15, 4) the value of skeleton S. The array elements which contain expressions are used to keep and modify atoms or lists (not fragments). They have a selection function, they are n-dimensional, and the garbage-collector examines their elements to keep them complete.

We will now give a list of the available operators. These operators, however, are not implemented in the Q-32, so that the following description refers only to the 709.

The reader who wishes more detailed information may consult Reference 8-4g.

Presently, the psuedo-modes DARR and LARR are in the process of implementation, hence they do not appear in the index in Chapter VII.

2-14.1 OPERATORS WHICH GENERATE SYMBOLS

=BLNK= whose value is the hollerith character blank

=LPAR= whose value is the hollerith character (

=RPAR= whose value is the hollerith character)

=GNSY= whose value is a new numeral whenever consulted again.

(=INTG= C1 C2 ...) its value is the atom obtained by combining the characters C1, C2, ... replaced; these C_i must be atoms (not numerals or lists), consisting of a single hollerith character; e.g., B, but not BAR.

(*DESN* A) its value is a fragment whose elements are the atoms which result from the decomposition $A_{replaced}$ into its hollerith characters.

=DISI= whose value each time it is consulted is the next hollerith character extracted from the array element upon which it had been previously initialized.

=DISO= whose value each time it is consulted is the next octal digit extracted from the array element upon which it had been initialized.

(*DIST* Z) whose value is an empty fragment, but which is an operator initializing =DISI= on array element Z.

(*DOST* Z) whose value is an empty fragment, however, which is an operator initializing =DISO= on array element Z.

2-14.2 OPERATORS TO CONSTRUCT LISTS

The symbol (), when it appears in a CONVERT program, always means the same empty list each time it appears, without taking into account how it was originally formed in the read-out, so that additional measures must be taken if a brand new empty list is desired.

=BILE= Each time it is found, it will produce a new empty list.

(*STAR* X L), whose value is an empty fragment, and which will cause the replaced value of X to be stored in the CAR of $L_{replaced}$. The CAR of L is now X.

(*STDR* X L), whose value is an empty fragment, which will cause the value of replaced X to be stored in the CDR of $L_{replaced}$. The CDR of L is now X.

2-14.3 OPERATORS FOR DATA MOVEMENT

=DISI=

=DISO= described on this page

(*DOST* Z)

(*DIST* Z)

(=PACK= C) whose value is the number of hollerith characters which may still be kept in the array element.
It is an operator which keeps the character C, replaced, in the nearest location available in the array element

upon which it had previously been initialized. Its value is a numeral.

(=STOR= O), same as =PACK=, except that it stores the octal digit O, replaced. Its value is (numeral) the number of octal digits which may still be kept in the array element.

(*PCST* Z) whose value is an empty fragment, is an operator which initializes =PACK= upon array element Z.

(*STST* Z) is a similar operator-fragment which initializes =STOR=.

2-14.4 INPUT AND OUTPUT OPERATORS

For the moment, there are three operators which read from the tape SYSPIT, write on SYSPOT, and write on a perforable output in SYSPCH, respectively. They all have as a value empty fragments.

(*READ* Z) reads sufficient records of SYSPIT to fill the array element Z.

(*WRIT* Z) writes the content of array element Z, probably filled with hollerith characters, in SYSPOT. Care must be taken to place the suitable control characters in column 1, to activate the control mechanism of the print carriage.

(*PNCH* Z) operates the same way, but places the record in SYSPCH. Column 1 may not be used for control purposes, otherwise it is assumed that it follows sequentially column 72 of the previous record.

CHAPTER III

DESIGN AND DESCRIPTION OF THE INTERPRETER CONVERT TO LISP

GENERAL OPERATION

3-1 Introduction.

CONVERT is presently being implemented in the IBM 709 and AN/FSQ-32 computers through a program written in LISP, which interprets -- and follows -- the CONVERT language.

On the 709, MBLISP has been used for its implementation; this is an interpreter of LISP; on the Q-32, LISP 1.5, a compiler, has been used.

Both LISPs present slight differences, the most important of which are the following:

1. In LISP 1.5, the empty list () is an atom, NIL.
2. In MBLISP, the value FALSE is the atom F; in LISP 1.5, the list/atom NIL.
3. LISP 1.5 gives to . and , special meanings.
4. In MBLISP we produce a number -- numeral -- by saying (DEC 8) while the reading routine of LISP 1.5 makes this conversion automatically.
5. LISP 1.5 has no (LAMBDA L ...) or (LAMBDA* ...).
6. MBLISP lacks a program feature.
7. There are other differences which have not interfered much.

Functions.

The principal functions used by the processor are: convert-conv, convert*-conv*, which apply to the expression E rule after rule, recognize =PRI= and =COM=; resemble and replace, which manipulate patterns and skeletons; edit, which transforms the dictionary by resemble into D used by replace; format, which permits recursion to replace convert.

3-2. Content of Chapter III.

This chapter will describe the functions necessary to implement CONVERT on the IBM 709 using MBLISP.

As frequently as necessary, we will indicate the differences with respect to the functions necessary for the implementation in 1.5 LISP Q-32.

(CONVERT M I E R)

1. M dictionary of cycle 3 of patterns and skeletons.
   I list of connectable variables and fragments -- will be considered of the UAR mode. --
   E expression to be converted.
   R dictionary of cycle 2 of sets of rules.

2. OPERATOR-FUNCTION

The value of (CONVERT M I E R) is the replaced skeleton of the rule whose pattern matched E.

Nevertheless, there are operator skeletons which, in addition to yielding a value, cause changes in their arguments, and messages be printed, tapes to be punched, etc.

3. DEFINITION

The definition of CONVERT IS

```
(CONVERT (LAMBDA (M I E R) ((LAMBDA (ALIST) ((LAMBDA (L0)
        ((LAMBDA (L1) (COND ((SET (DEC (QUOTE 99)) (QUOTE =AB65=))
        (CONV R L1 E ))) ) L))
     ) (ITLZI I))
     ) (CDDDDR (CDDDDR (ALIST)))
     )))
```

The definition of CONVERT on the Q-32 is found on page 3-48 of the original.

4. DESCRIPTION

Convert modifies I-M by means of (ITLZI I); it initializes the variables L0 and L1 to this value and the variable ALIST to (CDDDDR (CDDDDR (ALIST)))

Furthermore, the symbol generator =GENSY= initializes (dec 99). In MBLISP, the function (ALIST) gives as a value the present A-list; from it 8 elements are erased, i.e. the four arguments of CONVERT and their corresponding values.

On the Q-32, since no A-list exists, this step has been eliminated. After the initializations, convert calls upon CONV.

(CONV Q L E)

1. Q dictionary of sets of rules.
   L dictionary of cycle 3, formed by (ITLZI I), or LO, L* or L1.
   In any case, L is the dictionary under which conversion is made, i.e. that which will use RESEMBLE to compare patterns with E.
   E expression to be transformed.

2. Operator-FUNCTION

(CONV Q L E) applies CONVERT* to the first set of Q.

3. DEFINITION

The definition of CONV is

```
(CONV (LAMBDA ( Q L E) (CONVERT* (CADR Q) L E)))
```

(CONVERT* T L E)

1. T set of rules to be applied.
   L dictionary to be used by RESEMBLE during the application of T.
   E expression to be transformed under T.

2. Operator-FUNCTION

(CONVERT* T L E) applies each of the rules of T until finding one whose pattern compares with E, and then substitutes it in the corresponding skeleton of this rule. If no comparison is successful, the value of CONVERT* is the expression E.

Convert* uses CONV*, which is a function of a single variable, with L and E as free variables.

3. DEFINITION

The definition of CONVERT* is

```
(CONVERT* (LAMBDA (T L E) (CONV* T)))
```

(CONV* U)

1. U set of rules to be applied.
   Free variables:
   L dictionary to be used by resemble
   E expression to be replaced.
   Connected variables:
   L* the value of (RESEMBLE (CAAR U) L E) is assigned to L*

2. Operator-FUNCTION.

Applies rule after rule of U to E, accepting the first comparison.

3. DEFINITION.

The definition of CONV* is

```
(CONV* (LAMBDA (U) (COND
    ((NULL U) E)
    ((EQ (CAAR U) (QUOTE *COM=)) (CONV* (CDR U)))
    ((EQ (CAAR U) (QUOTE *PRI=)) (COND ((DO PRINT*
         (REPLACE (EDIT L) (CADAR U))) (CONV* (CDR U)))))
    ((ATOM (RESEMBLE (CAAR U) L E)) (CONV* (CDR U)))
    ((AND) ((LAMBDA (L*) (REPLACE (EDIT L*) (CADAR U)))
         (RESEMBLE 'CAAR U) L E))) )))
```

4. DESCRIPTION.

a. If U is null, i.e. if we have arrived at the end of the set of rules without finding one matching E, the value is E without change.

The result above is important; in fact, it specifies that at the last one of each set of rules, the processor "increases" the rule (== =SAME=).

This means that only the rules that modify expression E should be mentioned.

Nevertheless, sometimes, when we produce fragments as value, it is our aim that if the expression E matches with none of the rules, an empty fragment be produced. Then, the programer must terminate its set with the rule (== ()).

Note: (*rept* (A B =SAME= X) CONJS (
(. . . .)
(. . .. .)
.....
(== ())

b. Rules of the form (=COM= . . .) are ignored.

c. Rules of the form (=PRI= S1) are ignored except if the skeleton S1 is printed and replaced.

If the value of skeleton S1 contains numerals, they will be converted into common atoms before being printed by the printing sub-routine. This observation makes no sense on the Q-32.

Printing occurs on the SYSPOT tape, which is generally an A-3.

Furthermore, if Switch 3 is depressed, the last record written on the tape will be printed on-line by the printer.

Note that if the skeleton S1 contains operators, the program with the rule (=PRI= S1) will generally be different (with respect to the result) from that without it.

d. Rules whose pattern does not compare with E are skipped -- one proceeds to the next rule --.

e. We accept the first similarity or comparison of the pattern of a rule with E and we proceed substituting in the skeleton.

In the program there appears twice (RESEMBLE (CAAR U) L E) This means that we doubly compute RESEMBLE of the pattern being compared; this can be avoided by introducing an additional (LAMBDA (X) ...); nevertheless, the program such as it is on the previous page requires less of a pushdown list.

5. FUNCTIONS USED.

(CONV* U) uses RESEMBLE (original, page 3-10), REPLACE (original, page 3-49), EDIT (original, page 3-90).

(ITLZI I)

1. I list of NUMBERS -- atoms and singlets -- to be put in UAR mode.

Free variables

M dictionary of cycle 3 of patterns and skeletons.

2. FUNCTION.

The value of (ITLZI I) is a list which results from placing, in from of M -- modified by itlsm -- the elements of I, but with with UAR mode and partner ().

This means that if I = $(i_1\ i_2\ \ldots\ i_n)$
(ITLZM M) = (MMM) then
(ITLZI I) = $(i_1$ UAR () i_2 UAR () ... i_n UAR () MMM)

3. DEFINITION.

The definition of ITLZI is

```
(ITLZI (LAMBDA (I) (IF (NULL I) (ITLZM M) (CONS (CAR I) (CONS
      (QUOTE UAR) (CONS (LIST) (ITLZI (CDR I))))))))
```

4. DESCRIPTION.

5. FUNCTIONS USED.

ITLZM. (described below; see also original, page 3-48)

(ITLZM M) 1. M dictionary of cycle 3 of patterns and initial skeletons.

2. FUNCTION.

The value of (ITLZM M) is a list equal to its argument M, but with the following modifications:

CYCLE IN M	CYCLE IN (ITLZM M)	NOTES
(XXX) VAR L1	(XXX) VAR (L1 ())	L1 list
(XXX) CNT N	(XXX) CNT (M M)	If N numeral, M=N; if not, M=(dec N)
NAME (MOD) EXP	NAME MOD $EXP_{evaluated}$	

The remaining cycles of M pass without alteration to (itlzm m).

Modes between Parentheses.

A mode between parentheses indicates that its partner will first be evaluated [by EVAL, taking for ALIST the A-list associated to the variable ALIST] and subsequently this result will be the final "partner."

For example, if outside the CONVERT program
X = (b a r c o s) [ships], then the cycle M (VAR) (CDDR X)
is equivalent to the cycle M VAR (R C O S)

This means a singlet mode indicates a partner to be evaluated. These evaluations cannot make reference to any one of the four arguments M, I, E, R of CONVERT, since ALIST = (CDDDDR (CDDDDR (ALIST)))

On the Q-32, no partners are allowed to be evaluated; they will be ignored.

3. DEFINITION.

The definition of (ITLZM M) is

```
(ITLZM (LAMBDA (M) (COND ((NULL M) M)
      ((OR (NULL (CDR M)) (NULL (CDDR M))) (COND ((CONVERTERROR 2 0)
            (LIST)) ) )
      ((EQ (CADR M) (QUOTE VAR)) (CONS (CAR M) (CONS (CADR M) (CONS
          (IF (ATOM (CAR M)) (CADDR M)
            (LIST (CADDR M) (REDL (CADDR M))) )
            (ITLZM (CDDDR M)) )) ))
      ((EQ (CADR M) (QUOTE CNT)) (CONS (CAR M) (CONS (CADR M) (CONS
            (LIST (DEC (CADDR M)) (DEC (CADDR M))) (ITLZM (CDDDR M))))))
      ((ATOM (CADR M)) (CONS (CAR M) (CONS (CADR M) (CONS (CADDR M)
            (ITLZM (CDDDR M)) ) ) ))
      ((AND) (CONS (CAR M) (CONS (CAADR M) (CONS (EVAL (CADDR M) ALIST
            (ITLZM (CDDDR M)) ) ) ))  )))
```

4. DESCRIPTION.

The empty list added to the fragments in the VAR mode is not a new empty list, produced by (LIST), but is the empty list obtained if we take successive CDRs from L1 until it is impty.

In this way, the partner of VAR has two observers, which indicate respectively the beginning and the end of the associated fragment.

For example, (NNN) VAR (B A R B O N) is transformed into
(NNN) VAR (• ~)
(B A R B O N)

In general, all fragments are dealt with this way -- internally by resemble --; for example, (XXX) VAR ((O D I S E A) (S E A))
indicates XXX = O D I
This transformation to a notation of observers is internal and automatic; EDIT then re-transforms the results to the normal notation -- of list --, to construct the argument D of REPLACE, which uses no observer notation.

Furthermore, ITLZM checks the appropriate length of M, sending an error message if it is not cycle 3. The definition of ITLZM on the Q-32 is shown on page 3-48, original.

5. FUNCTIONS USED.

CONVERTERROR (original, page 3-8), REDL (original, page 3-8), EVAL.

(REDL K) 1. K list

2. FUNCTION.

The value of (REDL K) is the empty list obtained by taking successive CDRs from K until nulling it.

In this way, the value of REDL is always ().

3. DEFINITION.

The definition of REDL is

```
(REDL (LAMBDA (K) (IF (NULL K) K (REDL (CDR K)) )))
```

(CONVERTERROR N M)

1. N, M whole numbers (i.e. atoms, not numerals).

2. Operator-FUNCTION.

Prints an error message whose value is T or necessitates that the next APPLY be executed, skipping the one being carried out.

3. DEFINITION.

The definition of CONVERTERROR is

```
(CONVERTERROR (LAMBDA* (N M) (COND ((EQ N (QUOTE 1))
        (COND ((DO PRINT (CONC (LIST
       (QUOTE (*CONVERTERROR 1* IN (APPEND X Y)
       X IS )) (LIST X) (QUOTE (, Y IS)) (LIST Y))) ) SKP))) )
       ((EQ N (QUOTE 3)) (COND ((DO PRINT (CONC (LIST
       (QUOTE (*CONVERTERROR 3*
 IN L, THE DICTIONARY OF RESEMBLE, APPEARS AS NAME REP (PAT N)
  THE CYCLE)) (LIST (CAR X) (QUOTE REP) (CADR Y)) (QUOTE (.
  IF N WAS GIVEN THE VALUE 1 . CONTINUE THE CALCULATIONS)) )))
  (SKP) ))
  ((EQ N (QUOTE 2)) (COND ((DO PRINT (QUOTE (*CONVERTERROR 2*
        (IN (CONVERT M I E R ) M DOES NOT HAVE THE CORRECT
        LENGTH)))
            (SKP))))
   ((AND) (OR)) )))
```

4. DISCUSSION.

a. N = 1 indicates an error message in APPEND
   N = 2 indicates an error message in ITLZM
   N = 3 indicates an error message in the REP mode

b. M = 0 indicates "no jumps to next APPLY," with T the value of CONVERTERROR.
   M = 1 indicates "jump to next APPLY."

c. In all cases sufficient information is printed to notify what is happening.

Note that CONVERTERROR is a LAMBDA* function, since we do not wish its arguments to be evaluated -- we are interested in them as quoted material, i.e. as whole numbers --.

On the Q-32 LISP, since we do not evaluate the numbers, LAMBA* may be simply substituted for LAMBDA.

5. FUNCTIONS USED.

CONC, SKP (pages 3-72 and 3-9 of the original, respectively).

(SKP)

1. Free variables
M connected as CONVERTERROR

2. Operator-FUNCTION.

If M = 0, the value of (SKP) is T
M = 1, we print an error message and jump to carry out the next APPLY.

(SKIP) in a primitive function.

3. DEFINITION.

The definition of SKP is

```
(SKP (LAMBDA() (IF (EQ M (QUOTE 0)) (AND) (COND
                          ((DO PRINT (QUOTE (*GO
                TO NEXT APPLY*))) (SKIP) )))))
```

SKP has not yet been implemented on the Q-32.

3.3 DESCRIPTION OF RESEMBLE

(RESEMBLE X L E)

1. X pattern
L dictionary of cycle 3
E expression to be compared.

2. FUNCTION - semi-predicate.

The value of (RESEMBLE X L E) is F or a dictionary similar to L.

3. DEFINITION.

The definition of RESEMBLE is

```
(RESEMBLE (LAMBDA (X L E)
   (RESEMBLE* (LIST) L (LIST) (LIST X (LIST))
                                      (LIST E (LIST)))))
```

4. DESCRIPTION.

RESEMBLE calls upon RESEMBLE*, a function of 5 arguments, which is the one that really carries out the recursion.

(RESEMBLE* X L E XX EE)

1. X pattern to be compared with E
   L dictionary of cycle 3
   E expression to be compared with X
   XX pattern to be compared with EE in case X matches E
   EE expression perhaps to be compared with XX.

   Free variables
   LO initial dictionary produced by ITLZI; used in BUILD, SUB and MSU mode.

   Connected variables
   NOM connected to a name in the PAV mode
   K4 connected to a gensym
   K6 connected to a gensym

2. FUNCTION.

The value of (RESEMBLE* X L E XX EE) is F if there is no similarity between X-XX and E-EE, under L, or a dictionary to L, modified in the identified variables upon finding similarity.

3. DEFINITION.

The definition of RESEMBLE* is very broad, and is therefore omitted here; it is found in Chapter VII; furthermore, we will "examine minutely" the definition of RESEMBLE* to understand how its various patterns operate.

4. DESCRIPTION.

If X is a primitive pattern, it is compared with E, and if there is similarity, it is compared with (CAR XX) and

with (CAR EE), and so on, until finding a false response or until simultaneously exhausting XX and EE.

If X is not a primitive pattern, compare its CAR with (CAR E), placing its CDR in XX, and also keeping (CDR E) in EE; we proceed in this manner until finding a primitive pattern.

Consequently, if X and E ~~compare~~ match, we will compare the first element of XX with the first of EE; this makes it a TRUE function:

```
(TRUE (LAMBDA () (RESEMBLE* (CAR XX) L (CAR EE) (CDR XX)
                                                (CDR EE)) ))
```

If X is not primitive, none of the conditions of COND in RESEMBLE* will have been satisfied, and the last condition is then:

```
( (AND) (RESEMBLE* (CAR X) L (CAR E) (CONS (CDR X) XX) (CONS
                                                (CDR E) EE)) )
```

Thus, we see that RESEMBLE* assumes the value of its dictionary L when XX and EE have become empty:

```
( (NULL XX) (IF (NULL EE) L (OR)) )
```

In the entire program -- see Chapter VII "listed" -- this is the only part where RESEMBLE produces its response T, or TRUE; in the other cases, if a similarity occurs, RESEMBLE* calls upon (TRUE), i.e. it continues searching for similarity between XX and EE.

Note the manner in which RESEMBLE calls upon RESEMBLE*:

```
(RESEMBLE (LAMBDA (X L E)
                  (RESEMBLE* (LIST) L (LIST) (LIST X (LIST))
                                             (LIST E (LIST)) )))
```

A little study of the definition of RESEMBLE* will show that all the (LIST)'s of the definition above are necessary.

5. FUNCTIONS USED.

RESEMBLE* uses TRUE[3-47], EQUAL[3-47], REDUCE[3-27], ENTER[3-47], ENCOL[3-47], BUILD[3-25], SAME[3-24], INTERSECT[3-45], ASSSOC[3-46], FRAG[3-39], VFRAG[3-38], COMPA[3-38], DISMI[3-42], REP[3-43], CONVERTERROR[3-8], RED[3-35], DISMIN[3-44], ASSSOC*[3-32], ORD[3-45], SANDS[3-47].

The exponents indicate the page where the function has been described in the original.

PRIMITIVE PATTERNS IN RESEMBLE

==

2. PROPERTIES.

== compares with anything, list or atom. If various == appear in a single list, each one of them will compare, in general, with a different expression.

3. EXAMPLES

If X = (A (B ==) ==), L = () then X will compare with
E = (A (B RA) (Z O)), but not with
E = (A (B RA) Z O)

4. IMPLEMENTATION

The part of Resemble* which refers to == is
((EQ X (QUOTE ==)) (TRUE))
[found in the general COND of resemble 8]

6. FUNCTIONS USED

TRUE (original, page 3-11)

=ATO=

2. PROPERTIES

=ATO= compares with any atom -- including numerals. If various =ATO=s appear within a pattern, each one of them will generally compare with different atoms of E.

3. EXAMPLES

If X = (A (B =ATO=) =ATO=), L = (), then X will compare with E = (A (B R) 7), but not with (A (B R) (7))

4. IMPLEMENTATION

The part of resemble* which refers to =ATO= is
((EQ X (QUOTE =ATO=)) (IF (ATOM E) (TRUE) (OR)))

6. FUNCTIONS USED

TRUE (page 3-11, original)

=NUM=

2. PROPERTIES

=NUM= compares with any numeral. It compares with no atoms or lists. If various =NUM=s appear within a pattern, each one of them will compare in general with different numerals.

Note that these symbols, such as ==, =AND=, etc., which have a special interpretation if they appear as part of a pattern, do not have it if they appear in E, and are then treated as normal atoms.

3. EXAMPLES

X = (=NUM= =NUM= =NUM=), E = (7 8 9), then X will match with E on the Q-32; on the 709, E would have to be the value of
(LIST (DEC (QUOTE 7)) (DEC (QUOTE 8)) (DEC (QUOTE 9)));
consequently, the previous example results in F on the 709.

4. IMPLEMENTATION

The part of Resemble* which refers to =NUM= is
((EQ X (QUOTE =NUM=)) (IF (NUM E) (TRUE) (OR)))

5. DESCRIPTION

In MBLISP, (NUM X) is T if X is a numeral; if X is an atom with a numeric flag; numerals are obtained if we say (DEC Y) and Y are whole numbers (an atom without sign nor decimal point, formed by digits). Numerals in MBLISP have no written representation; consequently, they cannot be read or printed; (UNDEC NO) if NO is a numeral, will give as its value its atom or its corresponding [whole number].

In Q-32 LISP 1.5, a number is one of the following forms:

whole number	1	12	+2E4	-35
whole octal number	27Q	27Q3	-14Q	-14Qr
number with floating point	1.0	-0.5	+1.75	224.
	2.0	+357.75E-3		

Hence E is the power of base 10 for whole numbers and floating points;

Or it is the power of 8 for octals; these can only have positive exponents. Note: a number must begin with a character + - 0 1 ... 9

It may contain at the most one decimal point.

It may contain the letter E.

Reference: Lisp 1.5 Reference Manual for Q-32.

On the Q-32 we do not need to quote the numbers: for the same reason, the numbers cannot be used as connected variables: (LAMBDA (1 2 3) ...) will not work; we must use (LAMBDA (X Y Z) ...)

6. FUNCTIONS USED

TRUE (page 3-11 of the original)

=ORD=

2. PROPERTIES

=ORD= will compare with a list whose elements are arranged in an increasing order.

Among numerals, the greatest one is that whose value is greatest.

Remember that in MBLISP only positive numerals are permitted.

Among atoms consisting of a single hollerith character, the order is that given by its BCD representation; this means:
0 1 .. 9 = ' + A B ... I .) - J .. R $ * bland / S ... Z , (
i.e., for example, I is greater than A and less than K.

Among atoms formed by various letters, among lists, or among mixed elements, =ORD= is impredictable: nevertheless, it will match or fail.

3. EXAMPLES

=ORD= will match with (2 4 4 6); with (A B), (A), (), it will not match with A, HOMBR, (6 6 5); and will give impredictable results with ((A) (B) (C)).

4. IMPLEMENTATION

The part of resemble* which refers to =ORD= is ((EQ X (QUOTE =ORD=)) (ORD E))

6. FUNCTIONS USED

ORD (to be continued).

(ORD K) 1. K expression to see if it checks out.

2. FUNCTION

If K is not a list whose elements are arranged in an increasing order, the value of (ORD K) is F; otherwise, the elements of XX continue being compared with those of EE.

3. DEFINITION

The definition of ORD is

```
(ORD (LAMBDA (K) (COND
     ((ATOM K) (OR))

     ((OR (NULL K) (NULL (CDR K))) (TRUE))
     ((OR (EQ (CAR K) (CADR K)) (SL (CAR K) (CADR K))) (ORD (CDR K)))
     ((AND) (OR))      )))
```

4. DESCRIPTION

ORD assumes that the items to be compared are equal (EQ) or the first -- that of the left -- is less than that of the right.

On the Q-32, EQUAL is used to compare two numbers, and LESSP instead of SL.

5. FUNCTIONS USED

TRUE (page 3-11 of the original)

The definition in Q-32 LISP is

```
          (ORD (LAMBDA (K)
(COND ((ATOM K) NIL)
 ((OR (NULL K) (NULL (CDR K))) (TRUE))
 ((OR (EQUAL (CAR K) (CADR K)) (LESSP (CAR K) (CADR K)))
  (ORD (CDR K))) (T NIL))))
```

(ASSSOC X L)

1. X atom
   L list of cycle 3

2. FUNCTION.

ASSSOC searches for X among the first elements of each cycle of L; if it finds it, it returns a list containing second and third elements (mode and partner) of this cycle; if X is not in L, it returns (()()).

If L = (... X MODO SOCIO ...) [mode partner]
then (ASSSOC X L) = (MODE SOCIO)

3. DIFINITION

The definition of ASSSOC is

```
(ASSSOC (LAMBDA (X L) (COND
    ((NULL L) (QUOTE (() ())))
    ((EQ X (CAR L)) (LIST (CADR L) (CADDR L)))
    ((AND) (ASSSOC X (CDDDR L)))
    )))
```

VAR MODE

1. L = (... X VAR SOCIO ...)
   X is an atom which appears in the pattern and which is found in the L dictionary under the VAR mode.

2. PROPERTIES

An atom in the VAR mode causes its partner to be compared with expression E, by way of EQUAL -- they must be identical atoms or lists.

EQUAL does not recognize any special symbols in the partner [or in expression E] such as ==, ===, etc.

3. EXAMPLES

X = G; L = (G VAR (H O R M I G A S)),
E = (H O R M I G A S), then (RESEMBLE X L E) =
(G VAR (H O R M I G AS))
Y if E* = G, (RESEMBLE X L E*) = F.

4. IMPLEMENTATION

The part of RESEMBLE* which refers to the VAR mode is
((EQ (CAR Y) (QUOTE VAR))
(IF (EQUAL (CADR Y) E) (TRUE) (OR)))

hence Y is (ASSSOC X L) = (VAR SOCIO); (CADR Y) is the partner.

6. FUNCTIONS USED

EQUAL (original, page 3-17), TRUE (original, page 3-11).

(EQUAL X Y)

1. X, Y expressions (atoms or lists)

2. PREDICATE

The value of (EQUAL X Y) is T if X is the same atom as Y, or if both are equal lists; otherwise F.

Equal lists are those which have the same written representation; this means that they have the same number of elements with the same atoms and arranged in the same form. They may have a different internal representation; this means that EQ of two equal lists may be F, although EQUAL is T.

3. EXAMPLES

If X = (A (B C ()) (((D E)) F)),
Y = (A (B C()) (((D E)) F)), then (EQUAL X Y) = T
and (EQUAL X U) = F,
if U = (A (B C ()) ((D E) (F)))

3b. DEFINITION

The definition of EQUAL is

```
(EQUAL (LAMBDA (X Y) (COND
    ((ATOM X) (EQ X Y))
    ((NULL X) (NULL Y))
    ((ATOM Y) (OR))
    ((NULL Y) (OR))
    ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y)))
    ((AND) (OR))
    )))
```

In Q-32, EQUAL is a primitive function.

○ UAR MODE

1. L = (... X UAR SOCIO ...)
X is an atom which appears in the pattern and which is found in the dictionary L under the UAR mode.

2. PROPERTIES

An atom in the UAR mode matches with any expression, such as ==, changing after the comparison into a VAR mode which has as a partner the compared expression. Consequently, if it occurs several times in a pattern, then, at the corresponding place in expression E, the same expression must occur.

3. EXAMPLES

X = (Y A Y) L = (Y UAR **) E = (B A B)
G = ((BR) A (BR)) H = (B A C); then
(RESEMBLE X L E) = (Y VAR B)
(RESEMBLE X L G) = (Y VAR (BR))
(RESEMBLE X L H) = F

4. IMPLEMENTATION

The part referring to the UAR mode is

```
((EQ (CAR Y) (QUOTE UAR))
    (RESEMBLE*-(CAR XX) (ENTER (LIST X (QUOTE VAR) E) L)
    (CAR EE) (CDR XX) (CDR EE)))
```

6. FUNCTIONS USED

ENTER (to be continued).

(ENTER E L)

1. E list which has a length cycle 3
   L dictionary of cycle 3

2. FUNCTION

The value of (ENTER E L) is the L dictionary, however, with the content of E added as a cycle.

If E = (NAME MODE SOCIO) L = (... NAME MODE* SOCIO* ...) then (ENTER E L) = (... NAME MODE SOCIO ...); i.e. we replace in L the old cycle by a new cycle contained in E.

3. DEFINITION

The definition of ENTER is

```
(ENTER (LAMBDA (E L) (COND
    ((EQUAL (CAR E) (CAR L))
        (CONS (CAR E) (CONS (CADR E) (CONS (CADDR E) (CDDDR L)))))
    ((AND)
        (CONS (CAR L) (CONS (CADR L) (CONS (CADDR L)
        (ENTER E (CDDDR L))))))) )))
```

○ PAT MODE

1. L = (... X PAT SOCIO ...)
   X is an atom which appears in the pattern and which is found in the L dictionary under the PAT mode.

2. PROPERTIES

An atom in the PAT mode causes its partner to be compared with the expression E under the RESEMBLE rules in search of similarity. Consequently, the SOCIO [PARTNER] is treated as a pattern and therefore, in contrast with VAR, we will indeed recognize there special symbols.

3. EXAMPLES

```
X = (A B C D)  L = (B PAT (== C) C UAR *)
E = (A (N W) W D)  G = (A (N W) C D)  then
(RESEMBLE X L E) = (B PAT (== C) C VAR W)
(RESEMBLE X L G) = F
```

4. IMPLEMENTATION

The part of resemble* which refers to the PAT mode is

```
((EQ (CAR Y) (QUOTE PAT))
     (RESEMBLE* (CADR Y) L E XX EE))
```

○ PAV MODE

1. L = (... X PAV SOCIO ...)
   X is an atom which appears in the pattern and which is found in the L dictionary under the PAV mode.

2. PROPERTIES

Like PAT, an atom in the PAV mode causes its partner to be compared with expression E under the RESEMBLE rules in search of similarity. Consequently, the SOCIO is treated as a pattern. If such a similarity exists, the mode is transformed into VAR, and the compared expression passes as a SOCIO.

Then, if in the same pattern the same PAV atom occurs several times, it must be the same expression as that which occurs in the corresponding places of E.

PAV is a combination of PAT and UAR.

3. EXAMPLES

X = (Y Y) L = (Y PAV (===))
E = ((B A R C A S) (B A R C A S)) G = (B B)
H = ((B) (C)) I = (() ()) then
(RESEMBLE X L E) = (Y VAR (B A R C A S))
(RESEMBLE X L G) = F
(RESEMBLE X L H) = F
(RESEMBLE X L I) = (Y VAR ())
Meanwhile, if Y were in the PAT mode instead of the PAV mode, the results would have been (Y PAT (===)), F, (Y PAT (===)),(Y PAT (===)), respectively.

4. IMPLEMENTATION

The part of Resemble* referring to the PAV mode is

```
((EQ (CAR Y) (QUOTE PAV)) ((LAMBDA (NOM K4 K6) (REDUCE (RESEMBLE*
    K6 (CONS NOM (CONS (QUOTE PAT) (CONS K6 (CONS K4 (CONS (QUOTE
    VAR) (CONS E (CONS K6 (CONS (QUOTE PAT) (CONS (CADR Y) (ENTER
    (LIST NOM (QUOTE VAR) E) L) ))) ))) )))
    E (CONS (LIST) (CONS (QUOTE =DEF=) XX)) (CONS (LIST)EE) ))  )
    X(ENCOL X(QUOTE(2 7 A B 6 5)))(ENCOL X(QUOTE(2 4 J N 6 5)))))
```

5. DESCRIPTION

ENCOL produces two gensyms (atoms that have not occurred before), K4 and K6, and then we ask for
(RESEMBLE* K6 (NOM PAT K6 K4 VAR E K6 PAT SOCIO ... NOM VAR E ...) E
(() =DEF= XX) (() EE))

a. Having been before NOM PAV SOCIO, this cycle compares with NOM VAR E
b. K4 also retains the value E
c. We compare K6 -- as a new pattern -- with E instead of NOM
d. During the comparison of K6 with E, we maintain the definition NOM PAT K6 [=DEF= in XX will erase this cycle from the dictionary before starting to compare (car XX) with (car EE)], a temporary definition.

This is done with the objective of permitting recursive definitions of the NOM pattern of the type NOM PAV (=OR= =ATO= (NOM NOM NOM)) This definition is valid, and NOM winds up connected to the greater expression; for example, if E = ((A A A) B (C (D D D) E)), NOM will wind up connected to the previous list and not to (A A A) or B, although these expressions also satisfy the definition of NOM.

Finally, REDUCE reduces the dictionary, eliminating the GENSYMS and changing the partner of NOM for that of K4.

6. FUNCTIONS USED

REDUCE (original, page 3-21), ENTER (original, page 3-18), ENCOL.

(REDUCE K)

1. K dictionary of cycle 3 or atom

Free variables:
NOM a name which had the PAV mode
K6 a name (gensym) connected to the partner of PAV
K4 a name (gensym) whose partner is the value of NOM if there was similarity.

2. FUNCTION

The value of (REDUCE K) is

a. If K is an atom (presumably F), its value is K without modification.

b. Contrarily, it is assumed that K is a list of period 3 and copies it with the following modifications:
I. Cycles whose name is NOM or K6 are omitted
II. The cycle whose name is K4 is changed to NOM
III. The remaining cycles remain unchanged.

In this way, the new dictionary produced by REDUCE contains NOM with a partner which is the expression connected in the VAR mode.

3. DEFINITION

The definition of REDUCE is

```
(REDUCE (LAMBDA (K) (COND ((ATOM K)K)((NULL K) K)
((EQUAL NOM (CAR K)) (REDUCE (CDDDR K)))
((EQUAL K6 (CAR K)) (REDUCE (CDDDR K)))
((EQUAL K4 (CAR K)) (CONS NOM(CONS (CADR K) (CONS (CADDR K) (REDUCE
(CDDDR K)))) )) ((AND) (CONS (CAR K) (CONS (CADR K) (CONS (CADDR K)
(REDUCE (CDDDR K)))) ))) )))
```

For the Q-32 it is

```
(REDUCE (LAMBDA (K)
  (COND ((ATOM K) K)
   ((EQUAL NOM (CAR K)) (REDUCE (CDDDR K)))
   ((EQUAL K6 (CAR K)) (REDUCE (CDDDR K)))
   ((EQUAL K4 (CAR K))
    (CONS NOM (CONS (CADR K)
      (CONS (CADDR K) (REDUCE (CDDDR K))))))
   (T (CONS (CAR K)
     (CONS (CADR K) (CONS (CADDR K) (REDUCE (CDDDP K))))))))))
```

STL STG MODES

1. L = (... Y STL N ...)
   Y is an atom which appears in the dictionary L in the STL [STG] mode. Its partner is a numeral.

2. PROPERTIES

An atom with STL mode (strictly less) will compare with E if it is strictly less than the partner of this atom. Generally, such a comparison only makes sense if both E and PARTNER are numerals or atoms consisting of a single BCD character (see 3-3, =ORD=).

STG works similarly, only E must be strictly greater than the partner.

3. EXAMPLES

X = (A B) L = (A STL 6 B STG 4) matches with E = (2 9)

4. IMPLEMENTATION

The part of RESEMBLE* which refers to the STL, STG modes is

```
((EQ (CAR Y) (QUOTE STL)) (IF (SL E (CADR Y)) (TRUE) (OR)))
((EQ (CAR Y) (QUOTE STG)) (IF (SL (CADR Y) E) (TRUE) (OR)))
```

SUB MODE

1. L = (... Y SUB PATRON ...)
   [pattern]
   Y is an atom which appears in the L dictionary with the SUB mode. Its partner is a pattern.

2. PROPERTIES

An atom in the SUB mode matches any list; from it it selects the elements that present similarity with its partner and changes to the internal SEQ mode, having as a partner a list formed by the elements selected, added to the front of the pattern used to select them.

This means SEQ now has the following information: the criterion -- the pattern -- to select elements and a list of the elements selected. So that if the atom occurs again in the pattern, it must compare with a list and select from it the same elements as from the first, regardless of order.

The SUB mode requires that the sub-sets selected in different comparisons be the same.

4. IMPLEMENTATION

The part of resemble* which refers to the SUB and SEQ modes is

```
((EQ (CAR Y) (QUOTE SUB))
     (IF (ATOM E) (OR) (RESEMBLE* (CAR XX) (ENTER (LIST X
     (QUOTE SEQ) (CONS (CADR Y) (BUILD (CADR Y) E))) L)
     (CAR EE) (CDR XX) (CDR EE))))
((EQ (CAR Y) (QUOTE SEQ))
     (IF (ATOM E) (OR) (IF (SAME (CDADR Y) (BUILD (CAADR Y) E))
     (RESEMBLE* (CAR XX) L (CAR EE) (CDR XX) (CDR EE)) (OR))))
```

5. DESCRIPTION

The first time that a comparison is used, the cycle Y SUB PATRON [pattern] is transformed into Y SEQ (PATRON E_1 E_2 ... E_k), where the E_i's form the sub-set of E which has the property of comparing with the PATRON pattern.

Consequently, the next times that Y is called upon to compare, it will meet the SEQ mode.

BUILD selects -- from the new list -- the elements E_i' which compare with PATRON [which is now CAR of the SOCIO]; the value of BUILD is the new sub-set (E_1' E_2' ... E_r'); SAME compares this new list with the old (E_1 E_2 ... E_k); if these lists are the same, in the sense that they contain the same elements -- the order is unimportant, since sub-sets are involved -- the comparison is accepted; otherwise, the value of RESEMBLE is F.

If there is similarity, the cycle delivered by RESEMBLE is

Y SEQ (PATRON E_1 E_2 E_3 ... E_k)

so that EDIT must fix it up slightly.

6. FUNCTIONS USED

BUILD, SAME (described below).

(BUILD P E)

1. P pattern
   E list.

2. FUNCTION

The value of (BUILD P E) is the set of E elements which compare with pattern P.

This comparison is made independent of the comparisons made by RESEMBLE*; we use here the dictionary LO, the initial dictionary made by ITLZI on M-I, arguments of CONVERT.

3. DEFINITION

The definition of BUILD is

```
(BUILD (LAMBDA (P E) (COND
    ((NULL E) E)
    ((ATOM (RESEMBLE P LO (CAR E)))
        (BUILD P (CDR E)))
    ((AND) (CONS (CAR E) (BUILD P (CDR E))))
    )))
```

(SAME U V) 1. U, V lists -- sets --

2. Predicate

The value of (SAME U V) is T if U and V are lists which contain the same elements, although in a different order; otherwise F.

U and V have to have the same number of elements.

3. DEFINITION

The definition of SAME is

```
(SAME (LAMBDA (U V) (COND
    ((NULL U) (NULL V))
    ((NULL V) (OR))
    ((ELEMENT (CAR U) V) (SAME (CDR U) (REMOVE (CAR U) V)))
    ((AND) (OR))
    )))
```

4. EXAMPLES

(B A R C A) and (C A R B A) make SAME T; (GOL) and (G O O L) do not.

5. FUNCTIONS USED

SAME uses ELEMENT, REMOVE (described later).

(ELEMENT X L) X expression, L list.

Predicate: T if X is an element of L; if it belongs to L on the first level.

(REMOVE X L) X expression, L list.

Function: Remove, removes from X the first occurrence of X.

DEFINITIONS:

The definitions of ELEMENT and REMOVE are:

```
(ELEMENT (LAMBDA (X L) (AND   (NOT (NULL L))
                 (OR (EQUAL X (CAR L)) (ELEMENT X (CDR L))))) )

(REMOVE (LAMBDA (X L) (COND
    ((NULL L) L)
    ((EQUAL X (CAR L)) (CDR L))
    ((AND) (CONS (CAR L) (REMOVE X (CDR L))))
    )))
```

ELEMENT is a primitive function on the Q-32, called MEMBERX

MSU MODE

1. L = (... Y MSU SOCIO ...)
   Y is an atom which appears in the dictionary L with the MSU mode. Its partner is a pattern.

2. PROPERTIES

An atom in the MSU mode matches with any list; from it it selects those elements that have similarity with its partner and changes to the internal MSS mode, then having as a partner a list formed by the selected elements, adding to it in front the pattern used to select them.

MSS behaves like SEQ in the sense that if it is used again in comparisons, it uses CAR of the partner again as a pattern to form a new list. Nevertheless, in contrast with SEQ, it is not required that both sub-sets thus obtained are equal, except that MSS take its intersection.

The MSU mode then collects the maximum sub-set common to the lists with which it was compared.

4. IMPLEMENTATION

The part of RESEMBLE* which mentions the MSU and MSS modes is:

```
((EQ (CAR Y) (QUOTE MSU))
     (IF (ATOM E) (OR) (RESEMBLE* (CAR XX) (ENTER (LIST X
     (QUOTE MSS) (CONS (CADR Y) (BUILD (CADR Y) E))) L)
     (CAR EE) (CDR XX) (CDR EE))))
((EQ (CAR Y) (QUOTE MSS))
     (IF (ATOM E) (OR) (RESEMBLE* (CAR XX) (ENTER (LIST X
     (QUOTE MSS) (CONS (CAADR Y) (INTERSECT (CDADR Y) (BUILD
     (CAADR Y) E)))) L) (CAR EE) (CDR XX) (CDR EE))))
```

ATOMS THAT ARE NOT IN L

2. PROPERTIES

If an atom does not appear defined in the L dictionary, nor is it one of those recognized by RESEMBLE (for example, =DEC=), then this atom represents itself; it will match with E only if it is precisely the same atom.

3. EXAMPLES

If X = (F I D E L), L = (), then X alone will match with (F I D E L), and not with (C A S T R O), for example.

4. IMPLEMENTATION

The part of RESEMBLE* which mentions atoms that are not found in the dictionary is
((AND) (IF (EQ X E) (TRUE) (OR)))

In Q-32 LISP it was necessary to state:
(T (IF (EQUAL X E) (TRUE) NIL)) [EQUAL was used along X is a number].

EMPTY LIST

An empty list only matches with another empty list -- not necessarily the same --
((NULL X) (IF (NULL E) (TRUE) (OR)))

=BOR=

A "reserved" atom, of internal use. It causes the first cycle of the L dictionary to disappear, such as is done by =DEF= when it appears as CAR of XX. It is not expected that the CONVERT programer makes use of this atom in his patterns. He compares it with an empty fragment, so to speak.
((EQ (CAR X) (QUOTE =BOR=)) (RESEMBLE* (CDR X) (CDDDR L) E XX EE))

(=QUO= P1) 1. P1 expression (pattern).

2. PROPERTIES

The expression P1 by itself is compared with E, using EQUAL. No special symbols are recognized in P1; RESEMBLE is not used in the comparison.

3. IMPLEMENTATION

The part of RESEMBLE* which manipulates =QUO= is

```
((EQ (CAR X) (QUOTE =QUO=)) (IF (EQUAL (CADR X) E) (TRUE)
                                                    (OR)))
```

(=DEC= P1) 1. P1 is an atom

2. PROPERTIES

Compare (DEC P1) with E; convert P1 into a numeral and compare it with E, searching for equality. It is assumed that E is a numeral.

3. EXAMPLES

If E = (DEC (QUOTE 8)), L = (), then X = (=DEC= 8) will compare with E. On the Q-32, assuming that there is no need to convert an "entire" atom to numeral, the use of =DEC= is not necessary; therefore, for LISP 1.5, (=DEC= P1) is equivalent to (=QUO= P1).

4. IMPLEMENTATION

The portion of the Resemble* function related to =DEC= is

```
((EQ (CAR X) (QUOTE =DEC=)) (IF (EQ (CADR X)) E) (TRUE)
                                                 (OR)))
```

(=DEF= N1 P1 N2 P2 ... Nm Pm P') 1. The N_i's are names; i.e. they are atoms or singlets. The P_i's are patterns. Also P'.

2. PROPERTIES

Compare the P' pattern with expression E; during

this comparison, i.e. within P', the L dictionary is increased with the cycles
N1 PAT P1 N2 PAT P2 ... Nm PAT Pm

=DEF= defines temporarily the N_i's as being the PAT P_i.

If P' is omitted, Pm is compared with E.

3. EXAMPLES

4. IMPLEMENTATION

The portion of the resemble* function is

```
((EQ (CAR X) (QUOTE =DEF=))
     (IF (NULL (CDDR X)) (RESEMBLE* (CADR X) L E
                          XX         EE          )
                 (RESEMBLE* (IF (NULL (CDDDR X))(CADDR X)
                       (IF (NULL (CDDDDR X))(CADDDR  X) (CONS (CAR X)
                             (CDDDR X)) ))
                    (CONS (CADR X) (CONS (QUOTE PAT) (CONS (CADDR X) L)))
                 E (CONS (LIST)(CONS (QUOTE =DEF=) XX)) (CONS (LIST)EE)) ) )
```

and furthermore
((EQ (CAR XX) (QUOTE =DEF=)) (RESEMBLE* X (CDDDR L) E
(CDR XX) EE))

5. DESCRIPTION

a. If it is of the form (=DEF= P1), P1 is compared with E.

b. X of the form (=DEF= N1 P1) or (=DEF= N1 P1 P') cause a RESEMBLE* to be executed
[P1 or P'] (N1 PAT P1 LLL) E (() =DEF= XX) (() EE)

c. X of the general form (=DEF= N1 P1 N2 P2 ...) causes that a function be executed that has the form
RESEMBLE* (=DEF= N2 P2 ...) (N1 PAT P1 LLL) E (() =DEF= XX)
(() EE)

We add =DEF= to XX so that, upon completing the comparison between pattern P' and X, its first cycle be erased from L. Thus we obtain temporary definitions, valid only when comparing P'.

It is necessary to place () in front of =DEF= in order to hide it.

(=AND= P1 P2 ...) 1. P1, P2, ... patterns

2. PROPERTIES

This pattern matches with E if each one of its P_i's matches with E. The dictionary accumulates in the sense that any identification of variables during the comparison of P1 is retained -- and used -- for the comparison of P2, and so forth; this implies that if the same connectable variable [for example, UAR, PAV modes] appears in various P_i's, it must be compared with the same expression in the various comparisons.

3. EXAMPLES

X = (=AND= (== B ==) (A == ==)) will match with (A B C) but not with (O B E).

With L = ((XXX) UAR ()),
X = (=AND= (=== B XXX A ===) (=== A XXX B ===)),
E = (A O B M N A R B F A M N B O); then
(RESEMBLE X L E) = ((XXX) VAR (M N))
Note: the previous result will appear in the notation of observers, i.e.
((XXX) VAR ((M N A R B F A M N B O) (A R B F A M N B O))),
since this is the form in which resemble* manipulates the fragments. Before going on to the EXPR mode -- to be used by REPLACE --, they convert to a conventional notation, with RESTAUR.

4. IMPLEMENTATION

The part of resemble* which manipulates =AND= is

```
((EQ (CAR X) (QUOTE =AND=)) (COND
   ((NULL (CDR X)) (TRUE))
   ((AND) (RESEMBLE* (CADR X) L E (CONS (CONS (CAR X) (CDDR X)) XX)
   (CONS E EE)))))
```

5. DESCRIPTION

a. (=AND=) compares with any expression -- such as ==
b. (=AND= P1) is equivalent to P1 only.
c. (=AND= P1 P2 ...) causes RESEMBLE* to be executed
P1 L E ((=AND= P2 ...) XX) (E EE)

(=OR= P1 P2 ....) 1. P1, P2, ... patterns

2. PROPERTIES

A comparison between E and any of the P_i's is sought; the first similarity is accepted.

3. EXAMPLES

=OR= is used primarily to define recursive patterns. For example, in the L dictionary, the cycle
B PAT (=OR= =ATO= (B B)) defines B as a binary tree; this means it is

1. an atom
2. a list formed by two elements, each one of which is in turn a binary tree.

As such, B will compare with
((A B) ((C (D E)) (F ((G H) I)))), but not with (A B C).
X = (=DEF= R (=OR= A (=== R ===))) matches a list which contains an A at any level.

4. IMPLEMENTATION

The part of RESEMBLE* which manipulates =OR= is

```
((EQ (CAR X) (QUOTE =OR=)) (IF (NULL (CDR X)) (OR) ((LAMBDA (Y)
    (IF (ATOM Y) (RESEMBLE* (CONS (CAR X) (CDDR X)) L E XX EE) Y)
    ) (RESEMBLE* (CADR X) L E XX EE))))
```

5. DESCRIPTION

a. (=OR=) matches with nothing; it is the "unmatchable" pattern and always yields F as an answer.

b. (=OR= P1 P2 ...) is manipulated as follows: if RESEMBLE P1 L E matches, it is accepted; if not, we proceed with RESEMBLE (=OR= P2 ...) L E
Note that upon defining recursive patterns, the terminal condition goes first.

: (=NOT= P1) 1. P1 pattern

2. PROPERTIES

P1 is compared with E. If it matches the result is F. If it does not match, the result is L, and it is accepted and the comparison continues. So that (=NOT= P1) matches with an expression which has no similarity with P1.

3. EXAMPLES

X = (A B (=NOT= C) D) will match with (A B R D) but not with (A B C D).

4. IMPLEMENTATION

The part of resemble* which describes =NOT= is

```
((EQ (CAR X) (QUOTE =NOT=)) (IF (ATOM (RESEMBLE* (CADR X) L E XX EE))
   (RESEMBLE* (CAR XX) L (CAR EE) (CDR XX) (CDR EE)) (OR)))
```

(=== ...)

2. PROPERTIES

=== matches with any fragment, of arbitrary length, including the empty fragment or the zero fragment -- that whose length is zero --.

3. EXAMPLES

(=== A) matches with any list terminating in A: (B C A), (B A), (A), etc.

(=== ===) matches with any list; it is equivalent to (===).

4. IMPLEMENTATION

The part of resemble* which becomes charged with === is

```
((EQ (CAR X) (QUOTE ===)) (IF (NULL (CDR X)) (TRUE)
   ((LAMBDA (Y) (IF (ATOM Y) (IF (NULL E) (OR)
   (RESEMBLE* X L (CDR E) XX EE)) Y)) (RESEMBLE* (CDR X) L E XX EE)))
```

5. DESCRIPTION

a. (===) matches with any list.

b. (=== a b c ....) is treated as follows:
   I. We compare (a b c ...) with E and if it matches, it is accepted [this means that === matched with the empty fragment]
   II. If I does not match, we compare (=== a b c ...) with (CDR E) (recursive condition).
   III. Nevertheless, if II exhausts E without finding a comparison, the answer is F -- there was no similarity --. This means that === when it appears within an X pattern, will compare with any suitable fragment, so as to permit a comparison between the remainder of the X pattern and the remainder of the E expression.

FRAGMENTS

The modes listed below correspond not to atoms that appear in the L dictionary as atoms, but to atoms which appear in the dictionary as singlets (lists of atoms). This defines these atoms as representing a fragment; their partner, in general, is a list whose content, called associate fragment, will be compared with some fragment of E. The variable Y mentioned in the code is connected to the value of (ASSSOC* X L), which searches for the atom that represents a fragment in the L dictionary; it searches it as a singlet.

This function will be described below.

(ASSSOC* X L)

1. X atom
   L dictionary of cycle 3.

2. FUNCTION

ASSSOC* searches X within the first element of each cycle (i.e., merely inspects the cycles that refer to singlets); if it finds a cycle that satisfies, ASSSOC* returns as a value a list formed by the mode and the partner of that singlet; otherwise it returns (() ()).

3. DEFINITION

The definition of ASSSOC* is

```
(ASSSOC* (LAMBDA (X L) (COND
   ((NULL L) (QUOTE (() ())))

   ((ATOM (CAR L)) (ASSSOC* X (CDDDR L)))
   ((EQ X (CAAR L)) (LIST (CADR L) (CADDR L)))
   ((AND) (ASSSOC* X (CDDDR L)))
   )))
```

4. DESCRIPTION

This function, like ASSSOC [see page 3-16 of original], returns as a value (() ()) when atom X is not in the L dictionary; this is designed so that none of the examinations made on (CAR Y) or the CAR of the value of ASSSOC* be satisfied, and thus the final condition is reached; therefore, I repeat, if an atom is not special (for example, ==) and is not in the L dictionary, it represents itself, wherefore, it is necessary that (EQ X E).

○ VAR MODE, fragments. 1. L = (XXX (XXX) VAR SOCIO ...)

XXX is an atom that appears in L as a singlet. The partner is a list with two observers, i.e. is a list that contains two lists.

By convention, when we say that a cycle in L is, for example, (YYY) VAR (A B), we understand as a short notation instead of mentioning the observers, that they could be, for example,

(YYY) VAR ((A B P N O) (P N O))

The difference between these observers, i.e. the set of elements in the first list and not in the second, is the value of the partner. ITLZM changes from the normal notation to that of the observers; EDIT, through RESTAUR, does the inverse. [restore]

2. PROPERTIES

An atom in the VAR mode causes the associate fragment to compare with the corresponding fragment of the expression E, searching for equality, i.e. through EQUAL. No special symbols are recognized in the associate fragment.

3. EXAMPLES

X = (XXX XXX XXX) L = ((XXX) VAR (R U Z)) matches with E = (R U Z R U Z R U Z).

4. IMPLEMENTATION

The part of resemble* that manipulates the fragments in the VAR mode is

((EQ (CAR Y) (QUOTE VAR)) (FRAG (CAADR Y) E))

5. DESCRIPTION

FRAG compares through equal the elements of the partner with the first elements of E until the partner is finished (until both observers coincide); if they are equal, the calculation continues: it is called (TRUE); otherwise an F answer is obtained.

[illegible] this [illegible]

[illegible] as A [illegible], we [illegible]ly said
[illegible] TLZB [illegible] to the [illegible] of
[illegible] page 3-5 of the original.

[illegible] needs to know nothing about this
[illegible] observers than [illegible] rates with CONV[illegible]
[illegible] assume [illegible] something. [illegible]
[illegible] (XXX [illegible] (E [illegible]) will arrive in [illegible] over
[illegible] (R (B [illegible] 2 0 0 2 C 0))

[illegible] USED

[illegible] described below).

[illegible] P, E lists

The value of (FRAG R E) is is F is the elements of R [illegible] elements of E compared under EQUAL; otherwise [illegible] (TRUE). and continue comparing.

[illegible]

The definition of FRAG is

```
[illegible] (COND ((                    EQ
        (CADADR Y))   (RESEMBLE* (CDR X) L E XX EE))
    ((NULL E) (OR))       ((NULL R) (OR))
    ((EQ (CAR R) (CAR E)) (FRAG (CDR R) (CDR E)))  ((AND)(OR)) ))
```

[illegible]

The T2 which appears immediately to the right of COND asks when both lists are identical, otherwise they are compared as their elements are equal; when both observers coincide (i.e. all elements that had to be compared have already been compared), we compare the (CDR X) -- i.e. X without this XXX atom which has already been matched -- against the E which continues -- i.e. the original E reduced by the associate [illegible].

FUNCTIONS USED

[illegible] (page 3-17 of the original).

UAR MODE, fragments.

1. L = (... (XXX) UAR SOCIO ...)
XXX is an atom which appears in L as a singlet -- representing a fragment --. The partner is a list -- not two observers --.

2. PROPERTIES

The UAR mode is transformed into a WAR mode, changing its PARTNER, which is a list, to the representation of two observers, which as always, indicate the beginning and end of the first list. It is the WAR mode which does all the "heavy work."

4. IMPLEMENTATION

The part of resemble* which manipulates fragments in the UAR mode is
((EQ (CAR Y) (QUOTE UAR)) (RED (CADR Y) E))

6. FUNCTIONS USED

RED (described on this page).

(RED R B)

1. R, B lists

Free variables:
X L E XX EE -- used by resemble*

2. FUNCTION

Compares the elements of R with those of B, in search of equality. If there is equality, it calls upon resemble* to continue comparing, but changing the UAR mode from (CAR X) to WAR, and its partner to list with two observers.

If there is no equality, or if B ends ahead of R, we obtain F.

3. DEFINITION

The definition of RED is

```
(RED (LAMBDA (R B) (COND ((NULL R) (RESEMBLE* X (ENTER (LIST (LIST
     (CAR X)) (QUOTE WAR) (LIST E   B)) L) E XX EE))
     ((NULL B) (OR))
     ((EQUAL (CAR  R) (CAR B)) (RED (CDR R) (CDR B)))
     ((AND) (OR)) )))
```

4. DESCRIPTION

When the UAR mode calls upon RED, it does so in the form (RED (CADR Y) E), so that, originally, B is E, the expression to be compared.

We then begin to compare the elements of R -- which is the PARTNER -- against the first elements of E; if this comparison is successful in all elements, R will remain empty; a new cycle is then formed under the same NOMBRE [name] (singlet), but as a partner we will have (LIST E B), i.e.: to the left an observer to the original list; to the right an observer [B] to the clipped list; the difference between the two is the value of the associate fragment.

5. FUNCTIONS USED

EQUAL (page 3-17 of the original).

6. NOTE

By implementing RED we see that the partner of a UAR fragment has the following uses:

1. If it is an empty list, it indicates that the fragment may match with any convenient fragment, including the empty fragment.
2. If it is a list that contains some elements, this indicates that it may match with any fragment of a convenient final length, provided this fragment has as its first elements the elements of the partner.

This means that the partner of a fragment in the UAR mode indicates the elements with which the fragment must begin to match.

○ WAR MODE, fragments

1. L = (... (XXX) WAR SOCIO ...)
   XXX is an atom which appears in the L dictionary as a singlet. Its partner is a list which contains two observers.

2. PROPERTIES

WAR is an internal mode; it is not expected that the programer make explicit use of this mode. WAR originates in the UAR mode, fragments.

A fragment-atom in the WAR mode will compare with a fragment of arbitrary length, provided that:

a. The first elements of this fragment be the same as those of the associate fragment (see also RED, note item 6).
b. The rest of the elements of X compare with the rest of the elements of E.

If this similarity is possible, then WAR is transformed into VAR, and the associate fragment is now precisely the fragment with which it matched. Thus if an atom in the WAR mode is found several times in the X pattern, then in the corresponding places of the E expression the same fragment must appear.

4. IMPLEMENTATION

The part of RESEMBLE* which mentions the WAR mode is

```
((EQ (CAR Y) (QUOTE WAR)) (IF (NULL (CDR X)) (VFRAG (CAADR Y) E)
                              ((LAMBDA (R) (COND ((ATOM R) (COND((NULL
    E) (OR)) ((AND) (COMPA  (CAADR  Y) E))) )  ((AND) R)))
      (RESEMBLE*    X    (ENTER (LIST (LIST (CAR X)) (QUOTE VAR)
      (CADR Y)) L) E XX EE .))) )
```

5. DESCRIPTION

a. If XXX was the last fragment contained in X, then its final length is known, wherefore it is called VFRAG, which compares the content of the partner of XXX with the content of E, accepting or rejecting the assumed similarity immediately. If there is likeness, VFRAG changes WAR to VAR with the proper partner, as described. Otherwise F results.

b. WAR is changed into VAR, and it is established if there is similarity; if it exists, it is accepted.

c. In the opposite case, the length of the tentative fragment is increased, returning to the WAR mode. This is the recursive condition. F is obtained if E ends (NULL E) = T; this means that the tentative length of the fragment can no longer be increased.

6. FUNCTIONS USED

VFRAG, COMPA (described below); ENTER (page 3-18 of the original).

(VFRAG A B)

1. A, B lists

Free variables:
Y connected to the value of (ASSSOC* X L) or a list of the form (WAR SOCIO)

X L E XX EE -- variables of resemble*.

2. Semipredicate-FUNCTION

The value of (VFRAG A B) is F if the elements of A are not the same as the first elements of B, or if B is shorter than A; otherwise we continue with resemble*, but with WAR changed into VAR, and as a partner a list containing as the first observer E, and as the second observer a (), formed by REDL. This means it has as an associate fragment the content of E.

The question in the implementation of WAR: (IF (NULL (CDR X)) (VFRAG (CAADR Y) E) ...) is not necessary, since, if this tentative length of the fragment does not match, because it is too short, COMPA increases more and more until the final length is obtained (the total length, the entire content of expression E); nevertheless, in patterns which have connectable fragments with the last element, VFRAG provides speed.

3. DEFINITION

The definition of VFRAG is

```
(VFRAG (LAMBDA (A B) (COND ((                    EQ
A (CADADR Y)) (RESEMBLE* (CAR XX) (ENTER (LIST (LIST (CAR X))
     (QUOTE VAR) (LIST E (REDL B))) L) (CAR FE) (CDR XX) (CDR EE)))
((NULL B) (OR))
((EQUAL (CAR A) (CAR B)) (VFRAG (CDR A) (CDR B)))
((AND) (OR)) )))
```

4. DESCRIPTION

The question (EQ A (CADADR Y)) investigates if A -- the first observer -- is equal to (cadadr Y), the second observer. If this is so, it changes WAR to the VAR mode, the first observer is E, and the second, (). Otherwise it compares by way of equal element after element of the first observer with E, until the observers coincide or until E is exhausted.

In VFRAG, B is initially connected to E.

5. FUNCTIONS USED

ENTER (page 3-18, original), REDL (page 3-8, original).

(COMPA R B)

1. R, B lists

Free variables:

Y connected to the value of (ASSSOC* X L), or a list of the form (WAR SOCIO)
X L E XX EE -- used by resemble*.

2. Semipredicate-FUNCTION.

The value of (COMPA R B) is F if the elements of R arr not the same as the first elements of B, or if B is shorter than R; otherwise we continue with resemble*, but in the partner, the second observer loses an element, i.e. the new second observer is the CDR of the old second observer. Thus the tentative fragment in one element is increased. The WAR mode is preserved -- this means that the fragment, if necessary, can grow more.

3. DEFINITION

The definition of COMPA is

```
(COMPA (LAMBDA (R B) (COND ((NULL R) (OR))
     ((NULL B) (OR))
     ((                    EQ
      R    (CADADR Y)    ) (COND ((EQUAL (CAR R) (CAR B))
            (RESEMBLE* X (ENTER (LIST (LIST (CAR X)) (QUOTE WAR) (LIST
             (CAADR Y)   (CDR R))) L)  E XX EE))  ((AND)(OR))) )
     ((EQUAL (CAR R) (CAR B)) (COMPA (CDR R) (CDR B)))
     ((AND) (OR)) )))
```

PAT MODE, fragments

1. L = (... (XXX) PAT SOCIO ...)
XXX is an atom which appears in L as a singlet under the PAT mode.
The partner is a list.

2. PROPERTIES

An atom under the PAT mode represents a fragment pattern; it causes the content of its partner, i.e. its associate fragment, to take its place in X, and this new X is compared with E. All sorts of special atoms recognized by RESEMBLE are found in the partner.

3. EXAMPLES

(A XXX B) if L = ((XXX) PAT (+ = =) + UAR *)
then will compare with E = (A K R I B), yielding the value ((XXX) PAT (+ = =) + VAR K); in turn, it will <u>not</u> compare with (A (+ = =) B).

4. IMPLEMENTATION

The part of resemble* which loads the fragments in the pat mode is

```
((EQ (CAR Y) (QUOTE PAT))
     (RESEMBLE* (APPEND (CADR Y) (CDR X)) L E XX EE))
```

PAV MODE, fragments

1. L = (... (XXX) PAV SOCIO ...) XXX is an atom which is found in L as a singlet under the PAV mode; its partner is a list whose content will be treated as a pattern-fragment, as in PAT.

2. PROPERTIES

As in PAT, an atom under the PAV mode represents a pattern-fragment; it causes the content of its partner, i.e. its associate fragment, to take its place in X, and this new X compares with E. If the comparison is successful, PAT retains -- as an associate fragment -- the values of the fragment with which it matched, and its mode changes to VAR. Thus various occurrences of an atom with the PAV mode cause a comparison with the same fragment.

In recursive definitions, PAV retains the value of the largest fragment; for example, if it defined (EVEN) PAV ((*OR* () (== == EVEN))) and used the pattern (A EVEN B) to compare with expression (A 1 2 3 4 5 6 B), the result will be EVEN VAR 1 2 3 4 5 6, even if the fragments 1 2 3 4, 1 2 and empty matched with EVEN.

4. IMPLEMENTATION

The part of the function resemble* that manipulates the fragments in the PAV mode is

```
((EQ (CAR Y) (QUOTE PAV)) ((LAMBDA (NOM K4 K6) (REDUCE (RESEMBLE*
     (CONS (LIST (QUOTE SAND$) K4 K6)
                                   (CONS (QUOTE =BOR=)
                                   (CDR X))))(CONS NOM (CONS
     (QUOTE PAT) (CONS K6 (CONS K4 (CONS (QUOTE UAR) (CONS (LIST)
     (CONS K6 (CONS (QUOTE PAT) (CONS (CADR Y) (ENTER (LIST NOM
     (QUOTE PAT) (LIST (LIST (QUOTE SAND$) K4 K6))) L) ))) ))) )))
     E                XX                EE      )))
     (LIST (CAR X)) (LIST (ENCOL (CAR X) (QUOTE (2 7 A B 6 5))))
     (LIST (ENCOL (CAR X) (QUOTE (2 4 J N 6 5) ) )) ))
```

5. DESCRIPTION

PAV means variable pattern.

If we compare (XXX a b ...) with E = (e_1 e_2 ...), the process is the following:

a. We execute the function

```
RESEMBLE* ((SAND$ K4 K6) =BOR= a b ...)
          (NOM PAT K6 K4 UAR () K6 PAT SOCIO . . . NOM PAT ((SAND$ K4 K6)) ... )
          E
          XX
          EE
```

b. If similarity was found above, REDUCE eliminates K4 and K6 and changes to NOM -- or better, to its partner -- so that there remains (XXX) VAR (matched value).

The =BOR= will eliminate the first cycle of L while the fragment (AND K4 K6) is compared and similarity is found; consequently, the cycle NOM PAT K6 is temporary and only valid within (AND K4 K6). K4 and K6 are two gensym atoms produced by ENCOL.

6. FUNCTIONS USED

REDUCE (page 3-21, original), ENTER (page 3-18, original), ENCOL (described below).

(ENCOL X L) X atom, L list

FUNCTION. Produces a gensym (an atom that had not occurred before), making a reintegrate of X and list L; for example, if
X = ARBOL
L = (B E R Z A), then (ENCOL X L) = ARBOLBERZA
L must be a list whose elements are atoms.

DEFINITION. The definitions of ENCOL, DISINTEGRATE and DISINTEGRATE* are:

```
(DISINTEGRATE (LAMBDA (X) (COND ((DISSET X) (DISINTEGRATE* (DISINT))

(DISINTEGRATE* (LAMBDA (X) (IF (NULL X) X (CONS X (DISINTEGRATE*
      (DISINT ))) )))
(ENCOL (LAMBDA (X L) (REINTEGRATE (APPEND (DISINTEGRATE X) L)) ))
```

REP MODE, fragments L = (... (XXX) REP SOCIO ...)
XXX is an atom which appears in L as a singlet under the REP mode.

The partner may be of the form N or (PATR N) where N is a number and PATR a pattern.

2. PROPERTIES

An atom in the REP mode is equivalent to a pattern-fragment formed by N PATR patterns; if the partner is of the form N, it is then equivalent ot a pattern-fragment formed by N ==.

3. EXAMPLES

X = (N A V A) L = ((N) REP ((== B) 3)) will match with E = ((A B) (R B) (G B) A V A) but not with ((A B) (R B) (G B) (U B) A V A) nor with (A B R B G B A V A).

4. IMPLEMENTATION

The part of resemble* which manipulates the REP mode is

```
((EQ (CAR Y) (QUOTE REP)) (COND ((NUM (CADR Y)) (DISMI (CADR Y) E))
     ((ATOM (CADR Y)) (DISMI (DEC (CADR Y)) F))
     ((AND) (RESEMBLE* (APPEND (COND ((NUM (CADADR Y)) (REP
     (CADADR Y))) ((ATOM (CADADR Y)) (REP (DEC (CADADR Y))) )
      ((CONVERTERROR 3 0) (DEC (QUOTE 1)))) (CDR X)) L E XX EE))))
```

5. DESCRIPTION

The N to which reference reference was made in 1. and 2. can be a numeral or a "whole" atom (formed by digits); assumably, this distinction makes no sense on the Q-32.

REP does <u>not</u> retain the value of the matched fragment. In this regard, it is like PAT.
<u>Example</u>: X = (A NNN), ((NNN) REP 5) = L, E = (A R B O L E), then (RESEMBLE X L E) = ((NNN) REP 5), but with E = (A R B O L E S) will yield F.

6. FUNCTIONS USED

DISMI, REP (given in detail below).

(DISMI R E)

1. R numeral
   E list

2. Semipredicate-FUNCTION

Remove from E as many elements as indicated by R, and then apply resemble* to (cdr X) with the shortened E that remained. It will give a false answer if E is empty and R has not reached zero.

3. DEFINITION

The definition of DISMI is

```
(DISMI (LAMBDA ( R E ) (COND ((ZER R) (RESEMBLE* (CDR X) L E XX EE))
    ((NULL E) ( OR)) ((AND) (DISMI (DECR R) (CDR E))) )))
```

(REP N)

1. N numeral

Free variables:
Y connected to (ASSSOC* X L) = (REP SOCIO)

2. FUNCTION

Produces a list of as many elements as indicated by N (equal to SOCIO).

3. DEFINITION

The definition of REP is

```
(REP (LAMBDA (N) (IF (ZER N) (LIST) (CONS (CAADR Y) (REP
                                                (DECR N))) )
```

✪ CNT MODE, fragments

1. L = ((XXX) CNT SOCIO ...)
XXX is an atom found in L as a singlet under the CNT (count) mode.
The partner is a list formed by two numerals.

2. PROPERTIES

An atom under the CNT mode represents a fragment of undetermined final length but equal or greater than that indicated by its SOCIO [partner]. (Note: CNT is processed as a list of two numerals; the second numeral is that which indicates the length of the fragment at that moment; the first, the number of elements it must compare with). If the programer desires a fragment with the least number of elements, it is only necessary to make (TTT) CNT 2, since ITLZM transforms the previous cycle to the proper internal notation.

3. IMPLEMENTATION

The part of resemble* that deals with fragments in the CNT mode is

```
((EQ (CAR Y) (QUOTE CNT)) ((LAMBDA (Z) (COND ((ATOM (CADR Z))
      (COND ((NULL E) (OR)) ((NULL (CAR Z)) (OR))
     ((AND) (RESEMBLE* X (ENTER (LIST (LIST (CAR X)) (QUOTE CNT
     (LIST (DEC (QUOTE 1)) (INCR (CADADR Y)))) L) (CAR Z) XX EE
      )) )) ((AND) (CADR Z)) )) (DISMIN (CAADR Y) E)))
```

4. DESCRIPTION

DISMIN changes from CNT to REP in order to match; if this comparison is successful, the length of the fragment is accepted; if not, it is increased by 1 and one returns to CNT.

Note: actually, the change is not made to REP, but to REC; both modes are very similar, differing only in that REC produces the output REPLACE, i.e. it changes to the VAR mode, and REP does not.

CNT, if it found similarity, is transformed (through the internal REC mode) to the VAR mode, but XXX changes in the dictionary from singlet to atom; furthermore, EDIT -- before it is used by REPLACE -- transforms the list of two numerals to the atom corresponding to the numeral that indicates the length of the fragment with which it matched.

If an atom in the CNT mode appears several times in a pattern, then in the corresponding places of expression E the fragments with which it matched must have the same length, although they could supposedly consist of different elements.

On the Q-32, due to the confusion between atom and numeral, the description of DISMIN is simplified.

6. FUNCTIONS USED

ENTER (page 3-18, original), DISMIN (described below).

(DISMIN R E)

1. R, E lists

   Free variables
   Y -connected to (ASSSOC* X L) = (CNT SOCIO)
   X L XX EE - connected in resemble*

2. FUNCTION.

The value of DISMIN is a list of E and the value of resemble* with the CNT mode changed to REC; see also CNT mode on the previous page.

3. DEFINITION.

The definition of DISMIN is

```
(DISMIN (LAMBDA (R E ) (COND ((ZER R) (LIST E (RESEMBLE* (CDR X) (ENTER
     (LIST (LIST (CAR X)) (QUOTE REC) (CADADR Y)) L) E XX EE )))
     ((NULL E) (LIST E (OR)))
     ((AND) (DISMIN (DECR R) (CDR E))) )))
```

On the Q-32 it is

```
(DISMIN (LAMBDA (R E)
  (COND ((ZEROP R)
    (LIST E (RESEMBLEN (CDR X)
      (ENTER (LIST (LIST (CAR X))
        (QUOTE REC) (CADADR Y)) L) E XX EE)))
   ((NULL E) (LIST E NIL)) (T (DISMIN (SUB1 R) (CDR E))))))
```

○ (*NOT* P1)

X = (... (*NOT* P1) ...)
must appear always within a list, because it represents a fragment.

2. PROPERTIES

(*NOT* P1) compares with any fragment that does not compare with the content of P1; P1 is a list whose content is treated as a pattern.

3. EXAMPLES

(A (*NOT* (B C)) D) will compare with (A C B D) but not with (A B C D).

4. IMPLEMENTATION

The part of resemble* that manipulates *NOT* is

```
((EQ (CAAR X) (QUOTE *NOT*)) (IF
    (ATOM (RESEMBLE* (APPEND (CADAR X) (CDR X)) L E XX EE))
    (RESEMBLE* (CAR XX) L (CAR EE) (CDR XX) (CDR EE)) (OR)))
```

○ (*OR* P1 P2 ...)

1. This pattern must appear in a list, assuming that it represents a fragment.

P1, P2, ... , are lists whose content will be treated as a pattern-fragment.

2. PROPERTIES

(*OR* P1 P2 ...) compares with a fragment which matches with any of the contents of P1, P2, ... The comparison is made: first P1, then P2, ...

3. EXAMPLES

OR is mainly used to define recursive fragments; for example, a fragment of odd length is

(NON) PAT ((*OR* (==) (== == NON)))

Observe that the terminal condition is the first to be mentioned. With the above definition, X = (A NON B) will match with (A R Y E B), but not with (A R Y B).

(A (*OR* (=== M ===) (=== N ===)) C D) will match with (A R M E C D) and with (A N C D), but not with (A B C D).

4. IMPLEMENTATION

The part of resemble* that manipulates *OR* is

```
((EQ (CAAR X) (QUOTE *OR*)) (IF (NULL (CDAR X)) (OR) ((LAMBDA (Y)
    (IF (ATOM Y) (RESEMBLE* (CONS (CONS (CAAR X) (CDDAR X)) (CDR X))
    L E XX EE) Y)) (RESEMBLE* (APPEND (CADAR X) (CDR X)) L E XX EE))))
```

5. DESCRIPTION

a. If it is (*OR*), the result is F.

b. We make an APPEND of P1 and (CDR X) and compare it with E; if it matches, this comparison is accepted and the variables connected in it.

c. Otherwise, P1 is eliminated, and the process is repeated.

(*AND* P1 P2 ...)

1. This pattern must appear in a list, assuming that it represents a fragment.
P1, P2, ... are lists whose content is treated as a pattern-fragment.

2. PROPERTIES

X of the form (ΔΔΔ (*AND* (PPP) (QQQ) ...) ∇∇∇) is equivalent to one of the form (=AND= (ΔΔΔPPP∇∇∇) (ΔΔΔQQQ∇∇∇) ...)
i.e., to match with X, E must have fragments that match with the content of P1, that of P2, etc. Nevertheless, it is not required that the same fragment that matched with the content of P1 matches with that of P2, etc. The AND, which is "stronger," requires that it be the same.

3. EXAMPLES

(*AND*) is equivalent to ===; it will match with any fragment. X = (A (*AND* (=== B ===) (XXX)) C), L = ((XXX) UAR ()) asks for a fragment which has B as an element and retains it under XXX; for example, if E = (A N T E B L O C), then
(RESEMBLE X L E) = ((XXX) VAR (N T E B L O))

4. IMPLEMENTATION

The part of RESEMBLE* that deals with *AND* is

```
((EQ (CAAR X) (QUOTE *AND*)) (COND
    ((NULL (CDAR X)) (RESEMBLE* (CONS (QUOTE ===) (CDR X)) L E XX EE))
    ((AND)  (RESEMBLE* (APPEND (CADAR X) (CDR X)) L E (CONS
    (CONS (CONS (CAAR X) (CDDAR X)) (CDR X)) XX) (CONS E EE)))))
```

(AND P1 P2 ...)

1. This pattern must appear in a list, since it represents a fragment.
   P1, P2, ... are lists whose content is treated as a pattern-fragment.

2. PROPERTIES

(AND P1 P2 ...) will match with a fragment of the E expression if this same fragment compares simultaneously with the content of P1 and of P2 and of P3, etc. In contrast with *AND*, it is required that the fragment which compares with P1 be the same as that which compares with P2.

3. EXAMPLES

X = (=== A (AND (EVEN) (XXX)) B ===) with
L = ((XXX) UAR () (EVEN) PAT ((*OR* () (== == EVEN))))
allows us to detect a fragment of even length located between A and B, and to have this information under XXX;

For example, compared with
E = (1 2 A 1 6 3 B 1 2 A A 1 2 3 B 1 2) the result will be
(RESEMBLE X L E) = ((XXX) VAR (A 1 2 3) (EVEN) PAT ((*OR*
() (== == EVEN))))
Meanwhile, if we use *AND* to compare it with the same expression, the result will be
((XXX) VAR (1 6 3) (EVEN) PAT ((*OR* () (== == EVEN))))
since it would compare the following patterns with E

(=== A EVEN B ===) and
(=== A XXX B ===).

A result equivalent to the first example will be obtained if, instead of (AND ...) we only use EVEN, thus: X = (=== A EVEN B ===), but defining EVEN in the PAV mode.

4. IMPLEMENTATION

The part of resemble* that manipulates AND is
((EQ (CAAR X) (QUOTE AND)) (AND (LIST) E))

6. FUNCTIONS USED

AND (described below).

(AND A B)

1. A and B lists; of a form so that (APPEND A B) is always E.

Free variables:
X L XX EE - connected to resemble*

2. Semipredicate-FUNCTION

AND operates as follows:

a. It compares (=AND= P1 P2 ...) with A, and (CDR X) with B. If this comparison is successful, it is accepted.

b. Otherwise, it increases the final value of A by the first element of B which thus becomes reduced, and the part a. is repeated. A false answer results if B terminates (under b.) and a. has not found any similarity. In this way, it is insured that the same fragment that matches P1 is the same as that with P2, etc. [this fragment is the content of A].

3. DEFINITION

The definition of AND is

INTERNAL MANIPULATION OF MODES IN RESEMBLE

(a) MODOS	(b) ENTRADA EN EL PROGRAMA FUENTE (como el programador los escribe)	(c) PROCESO	(d) SALIDA PRODUCIDA POR RESEMBLE	(e) FORMA EN QUE SON USADOS POR REPLACE
VAR	X VAR EXPRESION	(f) X VAR EXPRESION	X VAR EXPRESION	X EXPR EXPRESION
UAR	X UAR **	X UAR ** X VAR EXPR	X UAR **	(h) Ignorada
PAT	X PAT SOCIO (g)	X PAT SOCIO	X PAT SOCIO	Ignorada
PAV	X PAV SOCIO	K4, K6 -vease modo PAV- X PAT (...)	X VAR EXPRESION	X EXPR EXPRESION
SUB	X SUB PATRON	X SUB PATRON X SEQ (PAT . . .)	(i) X SUB PATRON	Ignorada
SEQ	(j) -Interno-	X SEQ (PATRON A_1 A_2..)	X SEQ (PATRON A_1 A_2 ..)	X EXPR (A_1 A_2 ...)
MSU	X MSU PATR	X MSU PATR X MSS (PATR A_1 A_2..)	X MSU PATR	Ignorada
MSS	-Interno-	X MSS (PATR A_1 A_2..)	X MSS (PATR A_1 A_2..)	X EXPR (A_1 A_2 ..)
STL	X STL At	X STL At	X STL At	Ignorada
STG	X STG At	X STG At	X STG At	Ignorada

modos associated with expressions

Legend: (a) Modes; (b) Inputs in the source program (how the programer writes them); (c) Process; (d) Output produced by RESEMBLE; (e) Form in which they are used by REPLACE; (f) X VAR expression; (g) X PAT PARTNER; (h) Unknown; (i) X SUB pattern; (j) Internal

INTERNAL MANIPULATION OF MODES IN RESEMBLE

MODOS	ENTRADA EN EL PROGRAMA FUENTE (como el programador los escribe)	PROCESO	SALIDA PRODUCIDA POR RESEMBLE	FORMA EN QUE SON USADOS POR REPLACE
PAT	(XXX) PAT LISTA (a)	(XXX) PAT LISTA	(XXX) PAT LISTA	Ignorada
PAV	(XXX) PAV LISTA	K4, K6 -véase (c) modo PAV- (XXX) PAT (..)	(XXX) VAR EXPRESION	(XXX) EXPR EXPRESION
VAR	(XXX) VAR LISTA	(XXX) VAP Lista con (d) dos apuntadores.	(XXX) VAP Lista con dos apuntadores	(XXX) EXPR LISTA
UAR	(XXX) UAP LISTA	(XXX) UAP LISTA (XXX) WAP Lista con dos apuntadores	(XXX) UAP LISTA	Ignorada
WAR	-Interno-	(XXX) WAR lista c/2 ap. (XXX) VAP lista c/2 ap.	(XXX) VAP lista c/2 apuntadores	(XXX) EXPR LISTA
REP	(XXX) PEP (expr (b) núm ó átomo) ó (XXX) PEP número ó átomo	(XXX) REP (expr núm ó átomo) ó (XXX) PEP número ó átomo	(XXX) PEP (expr núm ó átomo) ó (XXX) PEP número ó átomo	Ignorada
CNT	(XXX) CNT ATOMO	(XXX) CNT (num_1 num_2) (XXX) REC num_2	(XXX) CNT (num_1 num_2)	Ignorada (a)
REC	-Interno-	(XXX) REC numeral	(XXX) REC numeral	XXX EXPR ATOMO observe que es XXX, no (XXX)

modes associated with fragments

Legend: Same as for table on preceding page, plus: (a) (XXX) PAT list; (b) Expression, number or atom; (c) See mode; (d) List with two observers; (e) EXPR ATOM observe that it is XXX, not (XXX)

```
(SANDS (LAMBDA (A B) ((LAMBDA (C) (IF (ATOM C) (IF (NULL B) (OR)
    (SANDS (APPEND A (LIST (CAR B)))(CDR B))) C))
    (RESEMBLE* (CONS (CONS (QUOTE *AND*) (CDAR X)) (CDR X)) L (CONS
    A B ) XX EE ) )))
```

RECURSIVE PART OF RESEMBLE*.

The patterns described before are the so-called basic patterns, i.e. those that resemble* recognizes immediately.

More complex patterns are analyzed by comparing their CAR with the CAR of E, and their CDR with the CDR of E. The corresponding code is

```
((AND)
   (RESEMBLE* (CAR X) L (CAR E) (CONS (CDR X) XX) (CONS
                                          (CDR E) EE)))
```

Definitions of ITLZM and CONVERT on the Q-32.

```
(ITLZM DEFINE ((ITLZM (LAMBDA (M)
    (COND ((NULL M) [illegible]
     ((OR (NULL (CDR M)) (NULL (CDDR M)))
      (COND ((CONVERTERROR 2 0) (LIST)) (T T)))
     ((EQ (CADR M) (QUOTE VAR))
      (CONS (CAR M)
       (CONS (CADR M)
        (CONS (IF (ATOMO (CAR M))
          (CADDR M) (LIST (CADDR M) (MEOL (CADDR M))))
         (ITLZM (CDDDR M))))))
     ((EQ (CADR M) (QUOTE CNT))
      (CONS (CAR M)
       (CONS (CADR M)
        (CONS (LIST (CADDR M) (CADDR M)) (ITLZM (CDDDR M))))))
     (T (CONS (CAR M)
       (CONS (CADR M) (CONS (CADDR M) (ITLZM (CDDDR M)))))))))
(CONVERT DEFINE ((CONVERT (LAMBDA (M L E A)
    ((LAMBDA (L0) ((LAMBDA (L1) (CONV M L1 E)) L0)) (ITLZM L))))))
```

3-4 DESCRIPTION OF R E P L A C E

(REPLACE D S)

1. D dictionary of the third cycle or period.
   S skeleton to be replaced or "filled."

   Free variables:
   E expression E, connected to CONVERT.
   LO Initial dictionary, value of ITLZI; connected in CONVERT.

L1 "Initial" dictionary L0 added to variables =QUOT=, =EXPR=, =SKEL=, *QUOT*, *EXPR*, *SKEL*.
L* "Present" dictionary; in addition to L1, it contains the variables connected by RESEMBLE on the left side of the rule.
R Dictionary of sets of rules. Connected in CONVERT.

Connected variables:
L0, L1, L* } brings them up to date conveniently
S connected in =ITER= and *ITER* to (CAR S)
N connected in =ITER= and *ITER* to a variable.

2. Operator FUNCTION.

The value of (REPLACE D S) is the skeleton S, with certain symbols substituted of the dictionary D, and certain others interpreted or obeyed in a special manner according to the particular rules of REPLACE and the CONVERT language.

3. DEFINITION

The definition of REPLACE is very broad, hence we omit it here; it is found in Chapter VII; furthermore, we will go into "detail" below regarding the definition of REPLACE to understand how its various skeletons are substituted.

4. DESCRIPTION

REPLACE is a large function, but very simple.

If S is a primitive skeleton, it replaces it conveniently.

If S is a more complex skeleton -- the recursive part -- then it replaces it element by element of the skeleton; this means it makes a CONS of the result of replacing the CAR with the result of replacing the CDR.

The majority of the atoms of S pass into the result as themselves without any alteration; this means as quoted material; only special atoms (those recognized by REPLACE) and those that appear in the dictionary D connected to a certain mode are "transformed."

5. FUNCTIONS USED

The numbers under the name indicate the page in the original. ASSSOC, PEINTEGRATE, XT, COMPLEMENT, INTERSECTION, UNION, CONC, CAR-

TESIAN, SUM, MINUS, PROD, DIVIDE, PEXP, FORMAT, CONV, ITERA, APPEND, CONVERT*,

ASSSOC*, DISINTEGRATE, SEQ, SET, PESET., *ABS* , LAST

SKELETONS IN REPLACE

=SAME=

2. PROPERTIES

The value of this skeleton is the expression E of CONVERT. In general, when we speak of the value of a skeleton, we understand the expression or fragment obtained if we substitute this skeleton, i.e. if we substitute in this skeleton (by means of REPLACE and of a dictionary D).

4. IMPLEMENTATION

The part of REPLACE which replaces the skeleton =SAME= is

((EQ S (QUOTE =SAME=)) E)

5. DESCRIPTION

The value of =SAME= is the expression which matched with the left side of the rule that contains on its right side the skeleton =SAME=.

=GNSY=

2. PROPERTIES

Its value is,

a. In the CONVERT version for the 709, a new numeral (starting with the numeral 99) each time it is called. A numeral is an atom. The numerals are non-printable. (UNDEC NO) or (=UDEC= NO) where NO is a numeral will result in a corresponding, whole numeral, capable of being printed.

b. In the CONVERT version forthe shared time system of the Q-32, it produces a new atom (not a number) each time it is substituted.

3. EXAMPLES

D = (A SKEL (=UDEC= =GNSY=)), S = (A A B A A)
will produce (709)
REPLACE D S) = (100 101 B 102 103).

4. IMPLEMENTATION

On the Q-32, the part of REPLACE which substitutes =GNSY= is
((EQ S (QUOTE =GNSY=)) (GENSYM))

On the 709, the implementation is
((EQ S (QUOTE =GNSY=)) (SEQ =AB65=))

Note also (page 3-2 of the original) the initialization made from the value of =AB65= in the definition of CONVERT.

6. FUNCTIONS USED.

SEQ (described below), =AB65= (described below).

```
(SEQ (LAMBDA* (X) (XAR (X (VAL X)) (CIDR X))))

(=AB65= INCR)
```

5. DESCRIPTION

CONVERT initializes the VALue from =AB65= to 99; furthermore, I define the function =AB65= as INCR [i.e. the function which assumes the value 1 greater than its argument]; next, each time I see =GNSY=, I apply SEQ to =AB65=; now then, (SEQ R) is a function to which the definition of R for the VALue of R applies, to produce a new VALue; this value of (SEQ R) is the old VALue of R, but also SEQ is an operator that computes -- and stores under R -- its new VALue, accordto the same definition of R. In the case of =AB65=, which initially has the VALue 99, and the definition INCR, when SEQ is applied to it, produces 99 as a value (i.e. the old VALue), and calculates the next sequential value, which turns out to be 100; it keeps this value in the VALUE cell of =AB65=, leaving it listed for the next SEQ.

A more detailed description of SEQ is found below, under the heading "VALUE CELLS: SEQ, VAL, SET OPERATORS."

The VALue of an atom is the content of its value cell; do not confuse with the value (lower case) of a skeleton, which is the result of substituting in it dictionary D.

VALUE CELLS. SEQ, VAL, SET OPERATORS
Only in 709 MBLISP

1. VALUE CELLS

Each atom has two cells associated with it, called value cells; here information can be stored associated with the atom according to different conventions.

The figure shows the internal structure of an atom in MBLISP. The CAR of the atom -- R in this case -- is a numeral that tells us of how many words make up its print-name: the CDR (broken line arrow: - - -) points to a word whose CAR is the definition of this atom, if it is defined as a function; the CDR of such a word (dotted region) is an observer for a sub-routine which tells us how to manipulate the definition.

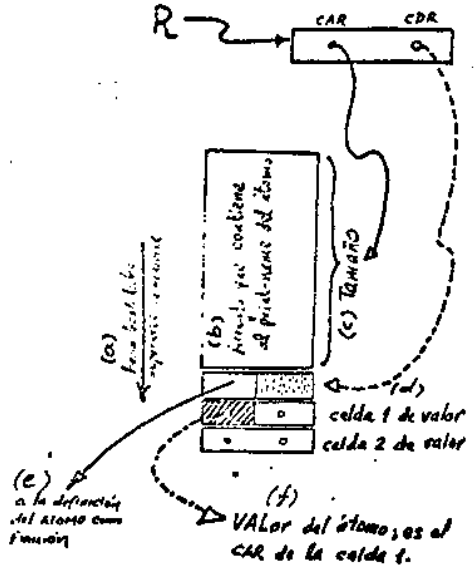
Next (i.e. in successive upper memory positions), we find the value cells; there are two value cells: the number 1 or nearest, and the number 2 or the farthest. These cells may have information in their CAR or their CDR (atoms or lists in the CAR, and lists in the CDR); hence we have or can keep four expressions in these cells. For example, (CAR (QUOTE CAMOTES)) = the numeral 2, since CAMOTES occupies two words;
CAMOTE S77777, where 7 is the illegal hollerith character 77.

(CADR (QUOTE APPEND)) = (LAMBDA (X Y) (IF (NULL X) Y
...)),
i.e., the definition of the append function.

(CAR (INCR (CDR (QUOTE R)))) is the cross-hatched part of the figure below or the one we called the VALue of an atom.

(CAR (INCR (INCR (CDR (QUOTE R))))) is the CAR of cell 2 of atom R.

As we see, these cells are ready to be used; it is a question of defining corresponding functions which keep and take values from these cells.



Legend:
(a) to locations [remainder illegible...]
(b) array element containing atom print-name
(c) size
(d) value cell 1 value cell 2
(e) to definition of ATOM as a function
(f) VALue of atom; this is the CAR of cell 1

2. THE SET, VAL AND SEQ OPERATORS.

They manipulate with the CAR of the cell 1 of an atom; this means with the VALue of that atom.

(SET X Y)

1. X expression
   Y atom.

2. OPERATOR

Its value is T.
It makes the VALue of Y equal to X. X and Y are evaluated.

3. EXAMPLES

(SET (QUOTE (1 2 3 4 5)) (QUOTE CAMOTES)) will place under the VALue of CAMOTES list (1 2 3 4 5).

3b. DEFINITION

The definition of SET is

```
(SET (LAMBDA (X Y) (SAR (PROTECT X) (CDR Y))))

(PROTECT(LAMBDA(X)((LAMBDA(Z)X)(INSERT X(SGARLIS)))))
(INSERT (LAMBDA (E X) (SAR E(CDR (CAR X) (CDR (CDR X)(LIST)))X))))
```

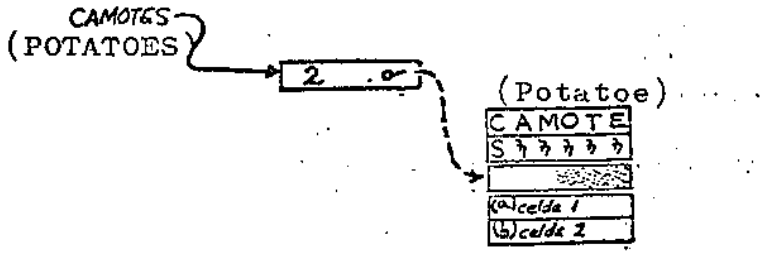
PROTECT gives as a value its argument, but it protects it from the garbage-collector.

(VAL X)

1. X is an atom. X is not evaluated.

2. FUNCTION

The value of (VAL X) is the VALue of the atom X. The value of an atom is the content of the left half of the value cell 1.



Legend: (a) Cell 1; (b) Cell 2.

3. EXAMPLES

If CAMOTES has been initialized as above, then (VAL CAMOTES) = (1 2 3 4 5); note that we should not say (VAL (QUOTE CAMOTES)).

CONCLUSION: (SET X A) gives A the VALue X; (VAL A) reproduces it.

4. DEFINITION

The definition of VAL is given below. VAL* is also defined, it will evaluate its argument; (VAL* (QUOTE CAMOTES)) = (VAL CAMOTES).

```
(VAL (LAMBDA* (X) (CAIDR X)))
(VAL* (LAMBDA (X) (CAIDR X)))
```

Note: The definition of VAL is CAP, INCR, CDR, i.e. CAIDR.

SEQUENTIAL VARIABLES

We would like to have a variable which, whenever it is consulted, produces a new value in accordance with an arbitrary function desired. The idea is, given an initial value to a variable -- to an atom --, to apply to it a function F (given by the programer) to produce a new value; the next value will result by applying to this value the same F, etc. So that:

initial value of R = Ao
then value 1 of R = F(Ao)
value 2 of R = F(F(Ao)) = F(value 1)
value 3 of R = F(F(FAo))) = F(value 2) = F(F(value 1))
etc.

(SEQ X)

1. X is an atom. X is not evaluated. Furthermore, X must satisfy the following conditions:
   a. It must be defined as a function of a variable.
   b. It must be initialized, i.e. before (SEQ X) a (SET == X) must have been made.
   Note: If an atom is not initialized, its VALue is the numeral 0; this means the CAR of cell 1 [and 2] is the numeral 0; the CDR is an empty list.

2. Operator FUNCTION.

The value of (SEQ X) is the VALue of X; furthermore, it computes the new VALue of X, applying X [since X is a function] to the actual VALue of X.

(SEQ X) computes the new VALue and delivers the old.

(SEK X), in turn, computes the new VALue, and delivers the new.

(VAL X) delivers the present value of X; it does not compute or change anything.

3. EXAMPLES

If CAMOTES is defined by Cdr, i.e. (CAMOTES CDR) or (CAMOTES (LAMBDA (Z) (CDR Z))), and furthermore, it is initialized to (1 2 3 4 5), then:

```
(SEQ CAMOTES) = (2 3 4 5)
(VAL CAMOTES) = (3 4 5)
(VAL CAMOTES) = (3 4 5)
(SEQ CAMOTES) = (3 4 5)
(SEQ CAMOTES) = (4 5)
(SEQ CAMOTES) = (5)
```

It is thus possible to define sequential variables, i.e. that take successive values according to a function.

Presently, the functions SET, VAL and SEQ are primitive functions in MBLISP.

Note that the VALUE CELLS 1 and 2 need not be used necessarily as described above, through functions SEQ and SEK; they may use VAL and SET to insert and retrieve data arbitrarily; they may define similar functions, since we already know the relation or placement of the value cells with respect to the atom. The definition of SEQ is given on page 3-51 of the original.

(ITER X) 1. X predicate
(generally, X is an operator)

2. Operator-PREDICATE.

ITER continues evaluating its argument until it is T; then the value of ITER is T.

Together with DO, and the VALUE CELLS, ITER is a powerful function. See below.

3. EXAMPLES

Let us take as an example the definition of ASSOC, which searches its first argument among the second, and yields as a value the next following element. The normal definition of ASSOC is

```
(ASSOC (LAMBDA (X L) (IF (EQ X (CAR L)) (CADR L) (ASSOC X
                                                  (CDDR L)))))
```

Using operators, the definition is

```
(ASSOC (LAMBDA (X L) (COND
     ((AND (SET (CONS (LIST) (CONS (LIST) L)) (QUOTE AT))
          (ITER (EQ X (CAR (SEK AT))) )) (CADR (VAL AT)) ) )))
```

with the additional definition (AT CDDR).

The above definition assumes that X is always found in X; doubting this, we would define AT as

```
(AT (LAMBDA (Z) (IF (NULL Z) (LIST X X) (CDDR Z) )))
```

ASSOC operates as follows:

a. The VALue of AT is made L, with () () added in front.
b. SEK calculates immediately the new VALue CDDR of L; then comes the comparison (EQ X (CAR (SEK AT))); if it is correct, the CADR of the VALue is taken; if is wrong, ITER evaluates again its argument; this means item b. is repeated. This ASSOC (which defines (AT CDDR)) has the advantage that it does not consume pushdown list, and is defined only in terms of primitive functions.

Iter is a primitive function.

=BLNK=

The value of this skeleton is the BCD "blank" character (blank space).

=RPAR=

The value of this skeleton is the BCD character) (right-hand parenthesis).

=LPAR=

The value of this skeleton is the BCD character ((left-hand parenthesis). The implementation of these three skeletons is as follows:

```
((EQ S (QUOTE =BLNK=)) (BLANK))
((EQ S (QUOTE =RPAR=)) (RPAREN))
((EQ S (QUOTE =LPAR=)) (LPAREN))
```

EXAMPLES

```
      (REPLACE (LIST) (QUOTE (=RPAR= =RPAR= =RPAR= =LPAR= =LPAR= =BLANK=))) =
 = ()))(( )
```

EXPR MODE

1. D = (... A EXPR SOCIO ...)
   A is an atom under the EXPR mode; its partner is an expression.

2. PROPERTIES

The value of an atom in the EXPR mode is its

partner without later replacement; its partner is copied as quoted material.

3. EXAMPLES

(D = (B EXPR (=SAME= G B)), S = (H A B A N A), then (REPLACE D S) = (H A (=SAME= G B) A N A).

4. IMPLEMENTATION

The part of replace which manipulates atoms in the EXPR mode is

((EQ (CAR Y) (QUOTE EXPR)) CADR Y))

5. DESCRIPTION

In the above code, Y is connected to the value of (ASSSOC S D) = = (EXPR SOCIO). The definition of asssoc is given on page 3-16 of the original.

SKEL MODE

1. D = (... A SKEL SOCIO ...)
A is an atom under the SKEL mode; its partner is an expression.

2. PROPERTIES

An atom in the SKEL mode causes a replacement on its partner; this means the value of this atom is the value of its partner.

I repeat, the value of a skeleton is the result obtained by substituting, according to the replace rules, in such a skeleton dictionary D.

3. EXAMPLES

D = (B SKEL (C G V) C EXPR CC), S = (B O L A C O); then (REPLACE D S) = ((CC G V) O L A CC O).

4. IMPLEMENTATION

The value of REPLACE which substitutes atoms under the SKEL mode is

((EQ (CAR Y) (QUOTE SKEL)) (REPLACE D (CADR Y)))

Note that the SKEL mode allows us to make chains of definitions which will have the same final value; for example, A SKEL B B SKEL C C SKEL D E SKEL B.

ATOMS NOT FOUND IN D.

If a skeleton is an atom not defined in the dictionary, it is replaced by itself; i.e. it remains unchanged.

EMPTY LIST

An empty list is replaced by itself; i.e. it does not change. Note: an empty list is not replaced by another empty list, but by the same empty list.

IMPLEMENTATION

The part of replace which takes into account atoms that do not appear in D is:

((AND S)

where, as in EXPR and SKEL, Y is connected to the value of (ASSSOC S D).

The part of replace that manipulates empty lists is

((NULL S) S)

(=PRNT= S1) 1. S1 skeleton.

2. PROPERTIES

(=PRNT= S1) carries out a replacement in S1, and also prints on the output tape SYSPOT (A-3) [on the Q-32, it prints on the TTY] the value of S1.

4. IMPLEMENTATION

As follows:

((EQ (CAR S) (QUOTE =PRNT=)) (PRINT* (REPLACE D (CADR S))))

5. OBSERVATIONS

a. Since the numerals are non-printable -- 709 --, we use PRINT*, which checks the presence of numerals and converts them into the corresponding atoms.

b. This skeleton is used to trace programs with errors as well as to learn about intermediate results. Furthermore, the rule (=PRNT= S1) is equivalent to S1.

(=DECM= S1)

1. S1, after being replaced, must be an atom composed of digits.

2. PROPERTIES

The value of (=DECM= S1) is the numeral corresponding to S1 replaced. In the CONVERT version running in the Q-32 computer, (=DECM= S1) is simply S1, hence =DECM= may be eliminated.

4. IMPLEMENTATION

The part of REPLACE which operates at =DECM= is

```
((EQ (CAR S) (QUOTE =DECM=)) (DEC (REPLACE D (CADR S))))
```

(=UDEC= S1)

1. S1 is a skeleton which, after being replaced, must produce a numeral.

2. PROPERTIES

The value of (=UDEC= S1) is the atom corresponding to the numeral S1 replaced.

4. IMPLEMENTATION

The part of REPLACE which operates at =UDEC= is

```
((EQ (CAR S) (QUOTE =UDEC=)) (UNDEC (REPLACE D (CADR S))))
```

(=INTG= S1 S2 ...)

1. S1, S2, ... are skeletons which, after being replaced, must be converted into atoms of a single character, for example, R, K, *, but not AB or CARROTE.

2. PROPERTIES

The value of (=INTG= S1 S2 ...) is the atom which results from "joining" the characters S1, S2, ... , replaced.

3. EXAMPLES

D = (AA EXPR A), S = (=INTG= B AA R C AA 7 8 *), then (REPLACE D S) = BARCA78*.

4. IMPLEMENTATION

The part of REPLACE which is charged with the production of atoms in this form, with =INTG=, is

((EQ (CAR S) (QUOTE =INTG=)) (REINTEGRATE (REPLACE D (CDR S))))

(=QUOT= S1) 1. S1 skeleton

2. PROPERTIES

Its value is S1 without any replacement; like quoted material. It is used if it is desired to obtain at the output atoms such as =SAME=, =UDEC=, etc., which are recognized and treated specially by REPLACE.

(=QUOT= N1 S1 N2 S2 ... Nm Sm S') 1. N1, N2, ... are names (atoms or singlets)
S1, S2, ... are skeletons.

2. PROPERTIES

Replaced in the last skeleton of the list, S' or Sm, if S' is omitted. During this replacement the temporary definitions N1 EXPR S1 N2 EXPR S2 ... Nm EXPR Sm apply.

Nevertheless, if a skeleton exists in S' =REPT=, =CONT=, or its version with *, the "temporary" definitions will continue in them; this does not apply to =BEGN= or *BEGN*, which re-initiates systematically the calculations on the initial dictionary.

4. IMPLEMENTATION

The part of REPLACE which manipulates a list whose CAR is =QUOT= is

```
((EQ (CAR S) (QUOTE =QUOT=)) (IF (NULL (CDDR S)) (CADR S)
   ((LAMBDA (L1 L*) (REPLACE (XT D (QUOTE EXPR) (CDR S)) (LAST S)))
   (XT L1 (QUOTE VAR) (CDR S)) (XT L* (QUOTE VAR) (CDR S)) )))
```

5. DESCRIPTION

a. The front part manipulates both forms of the =QUOT=, that which has only one element: (=QUOT= S1) and

that which has many; both signify different transformations; for example, (=QUOT= S1) does not substitute in S1.

b. The last skeleton of the list is replaced, but with the dictionary D increased in Ni EXPR Si, by XT.

c. L1 and L* are "brought up to date" by adding to them Ni VAR Si; so that

I. These "temporary" definitions become effective or have priority in REPT and CONT, which are those that use L1 and L*, respectively.

II. In REPT and CONT (mentioned in I), the earlier definitions are valid <u>even</u> on the left-hand side or patterns of the rules, since they pass as VAR.

6. FUNCTIONS USED

LAST (described below), XT (described below).

(LAST L)

1. L list

Gives us the last element of list L.

```
(LAST (LAMBDA (L) (IF (NULL (CDR L)) (CAR L) (LAST (CDR L)))))
```

(XT X M S)

1. X dictionary to which we add a few cycles
   M atom or singlet. If M is an atom, we will not make REPLACE from the second elements of S; otherwise, we replace before adding to X.
   S dictionary of cycle 2 of names (atoms or singlets) and skeletons to be added to X under the mode indicated by M.

2. FUNCTION

(XT X M S) yields as a value dictionary X, to which have been added cycles of length 3, as follows:

M FORM	S FORM	FROM THE RESULT FORMS THE VALUE
VAR	(N1 S1 N2 S2 ...)	(... N_i VAR S_i ... XXX)
(VAR)	(N1 S1 N2 S2 ...)	(... N_i VAR $S_{ireplaced}$ ... XXX)
EXPR	(N1 S1 N2 S2 ...)	(... N_i EXPR S_i ... XXX)
(EXPR)	(N1 S1 N2 S2 ...)	(... N_i EXPR $S_{ireplaced}$ ... XXX)
SKEL	(N1 S1 N2 S2 ...)	(... N_i SKEL S_i ... XXX)

The dictionary used to obtain the value Sireplaced is the present D dictionary. In the table above, XXX was the old content of X.

3. DEFINITION

The definition of XT is

```
(XT (LAMBDA (X M S) (IF (OR (NULL S) (NULL (CDR S))) X
    (CONS (CAR S) (CONS (IF (ATOM M) M (CAR M)) (CONS
    (IF (ATOM M) (CADR S) (REPLACE D (CADR S))) (XT X M (CDDR S))))))))
```

(*QUOT* S1) S1 skeleton -- is not a fragment --

2. PROPERTIES

The value of (*QUOT* S1) is the content of S1 without replacement.

(*QUOT* N1 S1 N2 S2 ... S') 1. N1, N2, ... NOMBRES (names) (atoms or singlets)
S1, S2, ... skeletons.

2. PROPERTIES

Its value is the content of the expression which would be produced by =QUOT=, if it were in place of *QUOT*.

Substitute in S', taking the content of the resulting value. Furthermore, the Ni's are defined as being EXPR Si; the dictionaries L1 and L* are modified, being enriched with the cycles Ni VAR Si; the Si's remain without substitution.

A more detailed explanation is found under =QUOT=; page 3-61 of the original.

3. EXAMPLES

S = (A (*QUOT* (G =SAME= R)) B) produces (A G =SAME= R B). If D = (G SKEL GG), S= (G (*QUOT* G JE H ACHE (G A C H)) H O G), then (REPLACE D S) = (GG JE A C ACHE H O GG).

Note that the temporary definitions have precedence; in this case, within the skeleton (G A C H), G is worth JE, although outside it is worth GG.

Note that a replacement in the last skeleton located within the list (*QUOT* ...) has been made, except if it is of the form (*QUOT* S1), in which case no replacement is made on S1.

4. IMPLEMENTATION

The part of REPLACE which manipulates *QUOT* is

```
((EQ (CAAR S) (QUOTE *QUOT*)) (APPEND
    (REPLACE D (CONS (QUOTE *QUOT*) (CDAR S))) (REPLACE D (CDR S))))
```

Note how first the value of S' is computed, and then an APPEND of this value is made to (REPLACE D (CDR S)).

(=EXPR= N1 S1 N2 S2 ... N_k S_k S')

1. N1, N2, ... are names (atoms or singlets)
   S1, S2, ..., S' are skeletons.

2. PROPERTIES

With the present dictionary D we substitute (evaluate) the skeletons S1, S2, ..., Sk; these values are placed under the EXPR MODE and the names N1, N2, ..., Nk, respectively, with this enriched dictionary being the one used to substitute skeleton S' or Sk, if S' is omitted.

The value of (=EXPR= . . . S') is the result of that substitution in S'; the value of (*EXPR* . . . S') is the content of the previous result; this means, a fragment.

The definitions Ni EXPR Si are only valid within S'; nevertheless, if in S' there exist any of the skeletons =REPT=, *REPT*, =CONT= or *CONT*, the "temporary" definitions will continue in them; this does not apply to *BEGN* or =BEGN=, which systematically re-initiates the calculations on the initial dictionary L0.

These definitions will continue even on the left side -- patterns -- of rules invoked by $\overset{*}{=}$REPT$\overset{*}{=}$ or $\overset{*}{=}$CONT$\overset{*}{=}$, since they pass onto L1 and L* as VAR; on the right side -- skeletons -- of these rules, the definitions continue as EXPR, since EDIT converts VAR to EXPR.

3. EXAMPLES

```
      M = () ; I = ((XXX)(YYY)(ZZZ)) ;  E = (C R I A T U R A S) ;
R = (C1 (                                   [creatures]
( (XXX A YYY A ZZZ)  (R (*EXPR* R N (YYY) (XXX YYY) (ENT (=REPT= =SAME= C2) R)) R))
      )
C2 (
( (=== R ===) SALIDA1 )     [output 1]
( (=== YYY ===) SALIDA2)
( == (SALIDA3 XXX YYY ZZZ) )
  ) )
```

Then, (CONVERT M I E R) = (R ENT (SALIDA3 XXX CRITUR ZZZ N R)
This result is obtained as follows:

The only rule of C1 matches and yields:
XXX = C R I; YYY = T U R; ZZZ = S,
then the skeleton is formed: first we place an R; then we define R as N and (YYY) as (C R I T U R); we substitute in S', which is the emphasized part of the program; this requires placing ENT at the beginning and R (whose value is N) at the end; furthermore, it requires applying rule C2 to (C R I A T U R A S) again.

The first rule of C2 fails, since it searches for an N in (C R I A T U R A S). The second rule of C2 fails because it searches for the fragment C R I T U R. The third rule matches [== matches with all], so that the value of the skeleton is (SALIDA3 XXX C R I T U R ZZZ); XXX and ZZZ are replaced by themselves, since they are not connected; YYY is connected to C R I T U R. So that the value of (=REPT= =SAME= C2) is (SALIDA3 XXX C R I T U R ZZZ). The value of the skeleton (ENT (=REPT= =SAME= C2) is (ENT (SALIDA3 XXX C R I T U R ZZZ) N);
a value which passes as a fragment to become the final result (as a fragment because it is *EXPR*, not =EXPR=); the final result has an R as the last element, this R is worth again R and not N, since it is outside of (*EXPR* ...)

4. IMPLEMENTATION

The part of REPLACE which manipulates =EXPR= is

```
((EQ (CAR S) (QUOTE =EXPR=)) ((LAMBDA (L1 L*) (REPLACE
    (XT D (QUOTE (EXPR)) (CDR S)) (LAST S)))
    (XT L1 (QUOTE (VAR)) (CDR S)) (XT L* (QUOTE (VAR)) (CDR S)) ))
```

The part of REPLACE that manipulates *EXPR* is

```
((EQ (CAAR S) (QUOTE *EXPR*)) (APPEND
    (REPLACE D (CONS (QUOTE =EXPR=) (CDAR S))) (REPLACE D (CDR S))))
```

FUNCTIONS USED

XT (page 3-62 of the original); LAST (page 3-61 of the original).

(*EXPR* N1 S1 N2 S2 ... S') 1. N1, N2, ... names (atoms or singlets)
S1, S2, ... skeletons [not fragments]

Their value is the content (i.e. the fragment) of the value which produces =EXPR= working on the same list; see =EXPR=, page 3-63 of the original.

The impliemntation of *EXPR* is found on the same page, above.

(=SKEL= N1 S1 N2 S2 ... Nk Sk S')

(*SKEL* N1 S1 N2 S2 ... Nk Sk S') 1. N1, N2, ..., Nk names (atoms or singlets)
S1, S2, lll, Sk, S' skeletons (not fragments)

2. PROPERTIES

Replaced in the last skeleton of the list, whether S' or Sk, if S' is omitted. During this replacement operation, the following temporary definitions apply: N1 SKEL S1 N2 SKEL S2 ... Nk SKEL Sk.

Nevertheless, if there exists a skeleton in S' ≛REPT≛ or ≛CONT≛, the "temporary" definitions will continue to prevail; this does not occur with ≛BEGN≛, which systematically re-initiates the calculations on the initial dictionary.

If it is =SKEL=, the value is the result of substitution on S'; if it is *SKEL*, the value of (*SKEL* ...S') is the content of the substitution on S', for which care must be taken that in this case the value of S' be a list.

These definitions will prevail only on the right side -- skeletons -- of rules invoked by ≛REPT≛ or ≛CONT≛, since they become L1 and L* as SKEL. This means, this connection of variables will not have any effect in the patterns.

3. EXAMPLES

D = (R SKEL RR), S = (R (*SKEL* R (RAR) (R A D A R)) F R)

will produce as a value (RR RAR A D A RAR F RR).

Note that the "temporary" definitions have precedence.

4. IMPLEMENTATION

The part of REPLACE which manipulates =SKEL= is

```
((EQ (CAR S) (QUOTE =SKEL=)) ((LAMBDA (L1 L*) (REPLACE
    (XT D (QUOTE SKEL) (CDR S)) (LAST S)))
    (XT L1 (QUOTE SKEL) (CDR S)) (XT L* (QUOTE SKEL) (CDR S)) ))
```

The part of REPLACE which manipulates *SKEL* is

```
((EQ (CAAR S) (QUOTE *SKEL*)) (APPEND
   (REPLACE D (CONS (QUOTE *SKEL*) (CDAR S))) (REPLACE D (CDR S))))
```

6. FUNCTIONS USED

XT (page 3-62, original), LAST (page 3-61, original).

(=RAND= S1 S2) 1. S1, S2, skeletons.

2. PROPERTIES

The value of (=RAND= S1 S2) is S1 or S2, substituted, making a "random" selection among these two skeletons.

The selection is really pseudo-random, in the sense that if the same CONVERT 2 program is run twice the same values will be obtained. It used (RANDOM), which is a primitive function in MBLISP, which is T or F, aleatorily.

Its principal application consists in the generation of skeletons which obey a certain law; it is useful in the generation of skeletons providing a certain probability to variables (see Chapter V, EXAMPLES).

3. EXAMPLES

S = (=SKEL= TR (=RAND= (=RAND= A B) (=RAND= C TR)))

gives the skeletons A, B and C the probabilities 1/3, 1/3, 1/3 of occurrence. Based on a similar skeleton is the generator of skeletons which produce a given distribution of probabilities, described in Chapter V of this Thesis.

4. IMPLEMENTATION

The implementation of =RAND= in REPLACE is very simple:

```
(EQ (CAR S) (QUOTE =RAND=))
   (REPLACE D (IF (RANDOM) (CADR S) (CADDR S))))
```

(=COMP= S1 S2)

(*COMP* S1 S2)

1. S1, S2 skeletons which after being substituted are lists -- sets --.

2. PROPERTIES

A set of elements is formed which exist in S1 but not in S2. The value of (=COMP= ...) is such a set; that of (*COMP* ...), the content of such a set.

3. EXAMPLES

If D = (C1 EXPR (A B C) (C2 EXPR (C U B O)), then
with S = (=COMP= C1 C2), (REPLACE D S) = (A)
with V = (=COMP= C2 C1), (REPLACE D V) = (U O)
with W = (=COMP= (B (*COMP* (C A M A S) C1) E) (=COMP= C2 (P O L I))),
we have (REPLACE D W) = (M S E),
since (*COMP* (C A M A S) C1) = M S
and (=COMP= C2 (P O L I)) = (C U B)

4. IMPLEMENTATION

The part of REPLACE which manipulates =COMP= is

```
((EQ (CAR S) (QUOTE =COMP=)) (COMPLEMENT (REPLACE D (CADR S))
    (REPLACE D (CADDR S))))
```

The part of REPLACE which manipulates *COMP* is

```
((EQ (CAAR S) (QUOTE *COMP*))
    (APPEND (COMPLEMENT (REPLACE D (CADAR S))(REPLACE D (CADDAR S)))
            (REPLACE D (CDR S))))
```

Correcto.

6. FUNCTIONS USED

APPEND, COMPLEMENT (described below).

(COMPLEMENT S T) 1. S, T lists -- sets --.

2. FUNCTION

The value of (COMPLEMENT S T) is a list of the elements in S and which are simultaneously not in T, which do not belong to T.

3. DEFINITION

The definition of COMPLEMENT is

```
(COMPLEMENT (LAMBDA (S T) (COND
    ((NULL S) S)
    ((ELEMENT (CAR S) T) (COMPLEMENT (CDR S) T))
    ((AND) (CONS (CAR S) (COMPLEMENT (CDR S) T)))
    )))
```

4. NOTES

In Q-32, since the atom T cannot be used to connect values, the atom TT was used.

5. FUNCTIONS USED

ELEMENT (page 3-24, original).

(=INTS= S1 S2 ... Sn)

(*INTS* S1 S2 ... Sn)

1. S1, S2, ..., Sn skeletons which after replacement become lists.

2. PROPERTIES

We form the intersection of the sets S1, S2, ..., Sn; this means the collection of elements found simultaneously in all the Si's.

(=INTS= ...) produces as a value such a list or collection; (*INTS* ...) is substituted by the content of such a list.

3. EXAMPLES

If D = (C1 EXPR (A B C) C2 EXPR (C U B O) (C3) EXPR ((2 3) (4 2 5)))

then
with S = (=INTS= C2 C2), (REPLACE D S) = (B C)
S = (=INTS= C2 C1), (REPLACE D S) = (C B)
if S = (=INTS= C3), (REPLACE D S) = (2), which is the intersection of (2 3) and of (4 2 5), which are lists produced by the fragment called C3.

U = (R (*INTS* (C A B A L L O) (C A L L O) (H A L L A B A))),
[horse] [corn] [spoke]
then REPLACE D U) = (R A L L). Note that the result will contain repeat elements only if all the S_i's have them.

4. IMPLEMENTATION

The part of REPLACE which operates with =INTS= is

```
((EQ (CAR S) (QUOTE =INTS=)) (INTERSECTION (REPLACE D
                                            (CDR S))))
```

The part of REPLACE which operates with *INTS* is

```
((EQ (CAAR S) (QUOTE *INTS*))
   (APPEND (INTERSECTION (REPLACE D (CDAR S))) (REPLACE D
                                                (CDR S))))
```

Note that first we replace (S1 S2 ... Sn) and then we call the INTERSECTION.

6. FUNCTIONS USED

INTERSECTION, APPEND (both described below).

(INTERSECTION L)

1. L = (L1 L2 ... Ln) where each Li is in turn a list.

2. FUNCTION

The value of (INTERSECTION L) is the intersection of L1, L2, ..., i.e. a list of all the elements common to each element of L.

3. DEFINITION

The definition of INTERSECTION is

```
(INTERSECTION (LAMBDA (L) (IF (NULL (CDR L)) (CAR L)
   (INTERSECT (CAR L) (INTERSECTION (CDR L))))))

(INTERSECT (LAMBDA (S T) (COND
   ((NULL S) S)
   ((NULL T) T)
   ((ELEMENT (CAR S) T) (CONS (CAR S) (INTERSECT (CDR S)
                        (REMOVE (CAR S) T))))
   ((AND) (INTERSECT (CDR S) T))
   )))
```

5. FUNCTIONS USED

INTERSECT (described here), ELEMENT (page 3-24 of the original), REMOVE (page 3-24, original).

(APPEND X Y) 1. X, Y lists

2. FUNCTION

The value of (APPEND X Y) is the list which results from adding to Y, in the same order and to the left, the elements of X. If X = (X1 X2 ... Xn), Y = (Y1 Y2 ... Yn), then (APPEND X Y) = (X 1 X 2 ... Xn Y1 Y2 ... Yn).

3. DEFINITION

In Q-32, APPEND is a machine function. In MBLISP, its definition is

```
((APPEND (LAMBDA (X Y) (COND ((ATOM X)(CONVERTERROR 1 1))
        ((ATOM Y) (CONVERTERROR 1 1))
        ((NULL X) Y)
        ((NULL Y) X)
        ((AND)((LABEL APPEND (LAMBDA(L)(IF(NULL L)Y (CONS (CAR L)
                                        (APPEND (CDR L))))))) X)) )))
```

4. DESCRIPTION

a. If X or Y are atoms, an error message is sent.

b. If X or Y are null, the result is the other variable.

c. On the contrary, the LABEL is used to re-define APPEND as a function of a single variable L; in this definition, Y is a free variable.

(=UNON= S1 S2 ... Sm)

(*UNON* S1 S2 ... Sm) 1. S1, S2, ... skeletons which after replacement will produce sets (lists).

2. PROPERTIES

A UNION of all the elements of S1, S2, ..., i.e. a list containing without repetitions the elements in S1 or in S2, ... is formed (=UNON= ...) and takes this list as its value; (*UNON* ...) takes as its value the content of this list.

(=UNON= S), where S is a single list, is used to eliminate from S the repeat elements.

3. EXAMPLES

If D = (C1 EXPR (A B C) C2 EXPR (B C D E)), then (REPLACE D (QUOTE (=UNON= C1 C2 (B A C A S)))) = (A B C D E S)

4. IMPLEMENTATION

The part of REPLACE which manipulates =UNON= is

```
((EQ (CAR S) (QUOTE =UNON=)) (UNION (REPLACE D (CDR S))))
```

The part of REPLACE which manipulates *UNON* is

```
((EQ (CAAR S) (QUOTE *UNON*))
   (APPEND (UNION (REPLACE D (CDAR S))) (REPLACE D (CDR S))))
```

6. FUNCTIONS USED

UNION (described below).

(UNION L)

1. L = (L1 L2 ... Ln), where L_i is in turn a list.

2. FUNCTION

The value of (UNION L) is the union of L1, L2, ..., i.e. the elements -- without repetition -- present in L1 or in L2 or in L3 or in ...

3. DEFINITION

The definition of UNION is

```
(UNION (LAMBDA (L) (IF (NULL L) L (EXTEND (CAR L) (UNION
                                                  (CDR L))))))
```

5. FUNCTIONS USED

EXTEND (described below).

(EXTEND S T)

1. S, T lists

2. FUNCTION

(EXTEND S T) adds to T the elements of S which are not present in T; it is a binary "union."

3. DEFINITION

The definition of EXTEND is

```
(EXTEND (LAMBDA (S T) (COND
    ((NULL S) T)
    ((ELEMENT (CAR S) T) (EXTEND (CDR S) T))
    ((AND) (EXTEND (CDR S) (CONS (CAR S) T)))
    )))
```

4. NOTES

TT was used instead of T in the CONVERT version which runs on the Q-32.

5. FUNCTIONS USED

ELEMENT (page 3-24 of the original).

(=CONC= S1 S2 ... Sn)

(*CONC* S1 S2 ... Sn)

1. S1, S2, ... skeletons which after substitution will produce lists -- sets -- as a value.

2. PROPERTIES

A major set containing -- in this order -- the elements of S1 plus those of S2 plus those of S3 plus those of S3 plus those of ... is formed. Repeat elements are not eliminated. It is like a multiple of APPEND.

3. EXAMPLES

```
(*CONC* (A B C) (C D E) (E F G H) =
A B C C D E E F G H
```

4. IMPLEMENTATION

The part of REPLACE which manipulates =CONC= is

```
((EQ (CAR S) (QUOTE =CONC=)) (CONC (REPLACE D (CDR S))))
```

The part of REPLACE which manipulates *CONC* is

```
((EQ (CAARS) (QUOTE *CONC*))
  (APPEND (CONC (REPLACE D (CDAR S))) (REPLACE D (CDR S))))
```

In the Q-32, CONCA has been used instead of CONC, since this is a pre-defined form with an arbitrary number of arguments.

6. FUNCTIONS USED

CONC (whose definition will be given below).

```
(CONC (LAMBDA (L) (IF (NULL L) L (APPEND (CAR L) (CONC
                                                 (CDR L))))))
```

(=CART= S1 S2 ... Sn)
(*CART* S1 S2 ... Sn)

1. S1, S2, ... skeletons which after being substituted will produce lists -- sets -- as a value.

2. PROPERTIES

The value of (*CART* ...) is the content of the value which would be produced by (=CART= ...).

The value of (=CART= ...) is a list whose elements are lists whose first element originates in S1, the second in S2, etc.; all the combinations are present and none repeated.

If any S_i is a fragment, then they will be sets to be taken into account in the formation of the cartesian product.

3. EXAMPLES

D = (C1 EXPR (A B C) C2 EXPR (1 2) (C3) EXPR ((S T) ((U) (D) (T)))).
If S = ((*CART* C1 C2) NN), then (REPLACE D S) = ((A 1) (A 2) (B 1) (B 2) (C 1) (C 2) NN).

If U = (=CART= (*) (X Y Z) C2), then
(REPLACE D U) = ((* X 1) (*X 2) (* Y 1) (* Y 2) (* Z 1) (* Z 2)).

If V = (=CART= C3 (A B)), then (REPLACE D V) = ((S (U) A) (S (U) B) (S (D) A) (S (D) B) (S (T) A) (S (T) B) (T (U) A) (T (U) B) (T (D) A) (T (D) B) (T (T) A) (T (T) B)), since the cartesian product of (S T), ((U) (D) (T)) and (A B) is carried out.

4. IMPLEMENTATION

The part of REPLACE which manipulates =CART= is

```
((EQ (CAR S) (QUOTE =CART=)) (CARTESIAN (REPLACE D (CDR S))))
```

The part of REPLACE which manipulates *CART* is

```
((EQ (CAARS) (QUOTE *CART))
  (APPEND (CARTESIAN (REPLACE D (CDAR S))) (REPLACE D
                                            (CDR S))))
```

5. NOTES

Note that first we substitute the S_i's and then we call the CARTESIAN.

6. FUNCTIONS USED

CARTESIAN (described below).

(CARTESIAN L) 1. L = (l1 L2 ... Ln) where each L_i is a list among which the cartesian product is carried out.

2. FUNCTION

The value of (CARTESIAN L) is the cartesian product among the elements of L, which are assumed to be lists.

3. DEFINITION

The definitions of CARTESIAN, CART and CART* are the following:

```
(CARTESIAN (LAMBDA (L) (IF (NULL L) (LIST L) (CART (CAR L)
    (CARTESIAN (CDR L))))))

(CART (LAMBDA (S T) (IF (NULL S) S (CART* T))))

(CART* (LAMBDA (T*) (IF (NULL T*) (CART (CDR S) T)
    (CONS (CONS (CAR S) (CAR T*)) (CART* (CDR T*))))))
```

4. DESCRIPTION

CARTESIAN applies CART to the first element of L and to CARTESIAN of its CDR. Assume that L = ((A1 A2 A3) (B1 B2 B3 B4) (C1 C2)); then assuming as already formed (CARTESIAN (CDR L)) = ((B1 C1) (B1 C2) (B2 C1) (B2 C2) (B3 C1) (B3 C2) (B4 C1) B4 C2)), to form the complete cartesian product, we must add in front of each element first A1; then A2 and then A3. We must assume that this is what CART does.

We could have defined CART as

```
(CART (LAMBDA (S T) (IF (NULL S) S
                        (APPEND (CART* (CAR S) T)
                                (CART (CDR S) T)) )))
```

where CART* adds (CAR S) to each element of T, and CART makes a recursion on S, so as to comprise the action of CART* within all its elements. It is like a MAPCAR [MAPAPPEND].

Then, the definition of CART* would be

```
(CART* (LAMBDA (E T*) (IF (NULL T*) T*
                          (CONS (CONS E (CAR T*))
                                (CART* E (CDR T*))) )))
```

CART* increases by E the elements of T*; (CART S T) applies the "incrementing" action of CART* to T for each element of S, making an APPEND of the results.

Compare now the previous definitions with those given in 3. (previous page) and observe how APPEND is eliminated.

In general, when the code says (APPEND (function 1 ...) (function 2 ...)) and function 1 and function 2 do not call upon each other, being recursive only upon themselves, then it is possible to eliminate APPEND, placing in the branch which has the terminal condition for function 1 the call for function 2, and in the terminal branch for function 2 the re-entry into function 1 with modified arguments. Observe and reflect.

The terminal condition in CARTESIAN, (IF (NUL L) (LIST L) ...) is correct, as can be derived from (CARTESIAN (QUOTE (A B C))) = (A B C).

5. FUNCTIONS USED

CART, CART* (described here, item 3. on the previous page).

(=ITER= N1 S1 N2 S2 ... Nk Sk S')

(*ITER* N1 S1 N2 S2 ... Nk Sk S')

1. N1, N2, ..., Nk skeletons which upon being substituted will produce an atom which will be our iteration variable.

↠ S1, S2, ..., Sk skeletons which upon being substituted will produce a list (a set), over which the corresponding variable produced by N_i will have its dominion.

If the value of any S_i is not a list, as expected, but a numeral, say N, then it is equivalent to holding that the value of S_i was not N but the set formed by the 1, 2, ..., to N;

i.e., in this case, the variable produced by the corresponding N_i will assume the values (numerals) 1, 2, ... through N.

→ S' is a skeleton.

2. PROPERTIES

(*ITER* ...) produces the content of what is produced by (=ITER= ...). The value of (=ITER= ...) is a list whose elements are formed by substituting S', with the dictionary D enriched by the cycles M1 EXPR σ1 M2 EXPR σ2 ... Mk EXPR σk where M_i are atoms (called variables of iteration), atoms which resulted from substituting in the corresponding N_i's, and the σ_i's are elements of $S_i$replaced. Each σ_i has all its values (of the set S_i) taken, without omitting nor repeating combinations, and the value of (=ITER= ...) is a list of the results of substituting S' in this form.

3. EXAMPLES

(=ITER= A (1 2 3) B (U V) (R A B)) = ((R 1 U) (R 1 V) (R 2 U) (R 2 V) (R 3 U) (R 3 V)).

The variables connected by =ITER= have preference over those already existing in the dictionary; for example, in the previous case, D was assumed empty (or at least it would not interfere with A, B, R); the same result would have been obtained if D had been (A EXPR GRAGRA), since within (R A B), A was connected to 1, 2 or 3.

D = (N SKEL (=UDEC= J) V EXPR J),
S = (A (*ITER* V (=DECM= 10) N) AR), then
(REPLACE D S) = (A 1 2 3 4 5 6 7 8 9 10 AR), since J, which is the value of V, assumes the values numeral^1, numeral^2, ... numeral^{10}, and N converts it into an atom.

```
(APPLY REPLACE ((A EXPR (1 2 3) B EXPR N B EXPR (A B E) B SKEL (B A N C
 O)) (=ITER= A B C ( C)))....

(GC)..

(((1 2 3) 1 A N O) ((1 2 3) 1 B N O) ((1 2 3) 1 E N O) ((1 2 3) 2 A N O
) ((1 2 3) 2 B N O) ((1 2 3) 2 E N O) ((1 2 3) 3 A N O) ((1 2 3) 3 B N
O) ((1 2 3) 3 E N O))..

(APPLY REPLACE ((B EXPR (1 2 3) C EXPR N D EXPR (A B E) E SKEL (B A N C
 O)) (=ITER= A B (=ITER= C D E))))..

(GC)..
```

```
(((1 2 3) 1 A N O) ((1 2 3) 1 B N O) ((1 2 3) 1 E N O)) (((1 2 3) 2 A
N O) ((1 2 3) 2 B N O) ((1 2 3) 2 E N O)) (((1 2 3) 3 A N O) ((1 2 3) 3
 B N O) ((1 2 3) 3 E N O)))..
                          --
(APPLY REPLACE ((B EXPR (1 2 3) C EXPR N D EXPR (A B E) E SKEL (B A N C
 O)) ((*ITER* A B C D E))))..
                          --
(((1 2 3) 1 A N O) ((1 2 3) 1 B N O) ((1 2 3) 1 E N O) ((1 2 3) 2 A N O
) ((1 2 3) 2 B N O) ((1 2 3) 2 E N O) ((1 2 3) 3 A N O) ((1 2 3) 3 B N
O) ((1 2 3) 3 E N O))..
                          --
(APPLY REPLACE ((B EXPR (1 2 3) C EXPR N D EXPR (A B E) E SKEL (B A N C
 O)) (=ITER= A D (*ITER* C D E))))..
                          -
(GC)..
      -
(((1 2 3) 1 A N O) ((1 2 3) 1 B N O) ((1 2 3) 1 E N O) ((1 2 3) 2 A N O
) ((1 2 3) 2 B N O) ((1 2 3) 2 E N O) ((1 2 3) 3 A N O) ((1 2 3) 3 B N
O) ((1 2 3) 3 E N O))..
                          --
(STOP).....
      -----
```

Note: Although S' is a skeleton, $\overset{*}{=}$ITER$\overset{*}{=}$ will produce correct results if a fragment is involved; for example, D = ((SSS) SKEL (1 2 K)), and if S = (=ITER= K (A B C) SSS), it will produce as a value (1 2 A 1 2 B 1 2 C); this may change and may not be correct for future implementations of CONVERT; nevertheless, it appears to be a useful property of $\overset{*}{=}$ITER$\overset{*}{=}$ which is worthwhile preserving.

4. IMPLEMENTATION

The part of REPLACE which is in charge of =ITER= is

```
((EQ (CAR S) (QUOTE =ITER=)) (IF (NULL (CDDR S)) (REPLACE D (CDR S))
     ((LAMBDA(N)(ITERA(REPLACE D(CADDR S))) )(REPLACE D(CADR S))) ))
```

The part of REPLACE which is in charge of *ITER* is

```
((EQ (CAAR S) (QUOTE *ITER*)) (IF (NULL (CDDAR S)) (APPEND (REPLACE D
     (CDAR  S)) (REPLACE D (CDR S)))  (APPEND ((LAMBDA(Z1 S N)(ITERA Z1)
     ) (REPLACE D (CADDAR S)) (CAR S) (REPLACE D (CADAR S)) )
          (REPLACE D (CDR S))) ))
```

5. DESCRIPTION

a. (=ITER= S') is converted into ($S'_{replaced}$), as it should be.

b. (*ITER* N1 S1 N2 S2 ... S') is converted to (itera (replace (N2 S2 ... S'))), where ITERA makes the conversion of the variable N, which is N_1 replaced, on the value of S1.

6. FUNCTIONS USED

ITERA (described below).

(ITERA C)

1. C is a list -- a set -- or
   C is a numeral.

2. FUNCTION

a. If C is a numeral, ITERA calls upon ITERAN.

b. If C is a set, we calculate
REPLACE (N EXPR (CAR C) DDD) (=ITER= N2 S2 ... S')
and ITERA is repeated upon (CDR C), so that N is connected under the EXPR mode, to each one of the elements of C. We therefore have an APPEND.

N is (REPLACE D (CADR S)); i.e. the value of N1.

3. DEFINITION

The definition of ITERA is

```
(ITERA (LAMBDA (C) (IF (NUM C) (ITERAN (DEC (QUOTE 1)))
     (IF (NULL C) C (APPEND (REPLACE (CONS N (CONS (QUOTE EXPR) (CONS
(CAR C) D))) (CONS (QUOTE =ITER=) (CDDDR S))) (ITERA (CDR C) ) )))) )
```

The definition of ITERAN is

```
(ITERAN (LAMBDA (M) (IF (SL C M) (LIST) (APPEND (REPLACE (CONS N
(CONS (QUOTE EXPR) (CONS M D))) (CONS (QUOTE =ITER=) (CDDDR S)))
     (ITERAN (INCR M))) )))
```

In ITERAN, C is the upper portion of the variable, i.e. it varies from 1 through C. If M is strictly greater than C (i.e. C is strictly less than M), then we "cut" the APPEND with a (list).

(=CONT= S1 N1 C1 N2 C2 ...)

(*CONT* S1 N1 C1 N2 C2 ...)

1. S1 skeleton
   N1, N2, ... names (atoms) of the set of rules
   C1, C2, ...

2. PROPERTIES

The value of (*CONT* ...) is the content of what (=CONT= ...) would produce. With the present dictionary D, we substitute in the skeleton S1 to form a new expression E; to this expression we apply the set of rules whose name is N1; i.e. we apply C1.

In addition to defining N1 as the set C1, the skeleton may contain the set of -- additional -- rules C2, C3, ..., with its corresponding names N2, N3, ... in the form of a dictionary of cycle 2.

The value of the skeleton (=CONT= S1 N1 C1 N2 C2 ...) is the result of applying the rules of C1 to the new expression E formed by S1 replaced.

During the conversion C1 carries out on this expression, all the variables presently connected are retained, i.e. those previously defined in M of CONVERT plus those defined by RESEMBLE toward the left side of the rule plus those defined by ≟QUOT≟, ≟EXPR≟, and ≟SKEL≟; ≟CONT≟ uses dictionary L*.

Let us recapitulate: the variables defined in M will have an effect on patterns and skeletons of C1, on patterns only or on skeletons only according to the mode by which they were originally defined; for example, A VAR BB will be a pattern; later on, EDIT will convert this to A EXPR BB, for which reason its influence extends to both sides of the rule; in turn, C EXPR KJIL only affects -- is valid only -- in SKELETONS, since B PAT (= ===) will only be used when comparing patterns. The description of each mode contains information about the final destination of its partner; furthermore, on page 3-47A of the original, a table of interconversion of modes is shown. Please consult.

The variables connected by the similarity toward the left side of the rule pass on to dictionary D of REPLACE (hence their effect on skeletons), as EXPR, depending on the details of the mode by which they were connected.

The variables connected by ≟QUOT≟ and ≟EXPR≟ affect both sides of the rule -- patterns and skeletons -- since they pass onto to L* under the VAR mode.

The variables connected by ≟SKEL≟ only affect the skeletons of the C1 rules, since they pass onto L as SKEL.

L* is the dictionary which RESEMBLE will use for a comparison of the new E with the patterns of the C1 rules.

ABBREVIATED FORMS

(≟CONT≟ S1 N1)

indicates that we will transform the (substituted) skeleton S1 by a set of rules called N1, which has already been mentioned -- defined -- earlier.

(*CONT* S1)

indicates that we will transform the (substituted) skeleton S1 by means of the present set of rules, i.e. the set which contains this skeleton (*CONT* S1).

Both are abbreviated forms of the more general form given on the previous page. As a convenient notation, we may consider *CONT*, whose obvious meaning is =CONT= and/or *CONT*.

3. EXAMPLES

```
M = (), I = (X (XXX)) .
  R = (C1 (
           ( (X XXX) (=CONT= (XXX)) )
       ) )
```

Applied to E = ((N) (N) (N) (T) (U) R), the result will be: (CONVERT M I E R) = ((N) (N) (T) (U) R), since the only rule of C1, the first time it is applied, gives to X a value (N), and to XXX the value (N) (N) (T) (U) R; next =CONT= retains these values and applies the pattern (X XXX), i.e. ((N) (N) (N) (T) (U) R), to (XXX), i.e. to ((N) (N) (T) (U) R), hence the rule fails (this rule requires that a list be equal to its CDR), yielding the expression E, i.e. (XXX), as a value.

If instead of =CONT= we had used =REPT=, which does not maintain connected values, we would have obtained () as a value, since each time the rule is applied, X would be free (UAR mode) again, hence the pattern (X XXX) would fail only when E = ().

With this same example, if R were

```
R = (C1(
      ( (X XXX) (=J= (*CONT* (XXX) C2 (
                                ( (=== X ===) (DOSVECES) )
                                ( (== == == ===) (XXX) )
                               )) =J=) )
    ) )
```

The first rule of C2 says: if E contains (N), write (DOSVECES) [two times].

The second rule of C2: if E contains three or more elements, write ((N) (N) (T) (U) R); note how X and XXX regained the value (connected in C1) on the left side of the C2 rules. The result of applying CONVERT with this new R is (=J= DOSVECES =J=).

If the previous R is applied to E = (A B C D), there would result (=J= B C D =J=); applied to (A B), the result would be (=J= B =J=).

4. IMPLEMENTATION

The part of REPLACE which manipulates =CONT= is

```
((EQ (CAR S) (QUOTE =CONT=)) (FORMAT D L* S))
```

The part of REPLACE which manipulates *CONT* is

```
((EQ (CAAR S) (QUOTE *CONT*))
  (APPEND (FORMAT D L* (CAR S)) (REPLACE D (CDR S))))
```

6. FUNCTIONS USED

FORMAT (described below), APPEND (page 3-67, original).

(FORMAT D H S)

1. H dictionary, with which the calculations proceed.
   D dictionary D or REPLACE, with which we will substitute skeleton S1.
   S skeleton of the form (=CONT= ...), (*REPT* ...), etc. Useful to determine what type of form is being used (e.g., abbreviated form).

2. Operator FUNCTION.

FORMAT applies CONVERT to skeleton S1 replaced. The dictionary H is L1 if =REPT= or *REPT* is involved, or L* if =CONT= or *CONT* are involved.

a. If S = (≛REPT≛ S1) or (≛CONT≛ S1), then CONVERT* applies with the set T. This is a free variable, precisely connected in CONVERT*; it is the set being used, which contains the rule which contains that skeleton.

b. If S = (≛REPT≛ S1 N1) or (≛CONT≛ S1 N1), then search for N1 in Q, which is a dictionary of a set of rules, applying CONVERT* with the set that has the number N1.

   Q is the original dictionary R of sets, increased perhaps by new definitions of sets of rules.

c. If S = (≛REPT≛ S1 N1 R1 N2 R2 ...) or (≛CONT≛ S1 N1 R1 N2 R2...) then it manufactures a new Q, adding N1 R1 N2 R2 ... to the front of the old Q and processing the first set of the new Q; i.e. applying R1 to $S1_{replaced}$.

d. H is the dictionary used in all these cases, as a dictionary for comparing by means of RESEMBLE; H also may bring cycles with modes EXPR, SKEL, CONT or REPT, which do not affect

RESEMBLE, but which EDIT will recognize and will let pass on to dictionary D used by REPLACE.

3. DEFINITION

The definition of FORMAT is

```
(FORMAT (LAMBDA (D H S) (COND
    ((NULL (CDDR S))
        (CONVERT* T H (REPLACE D (CADR S))))
    ((NULL (CDDDR S))
        (CONVERT* (ASSOC (CADDR S) Q) H (REPLACE D (CADR S))))
    ((AND)
        (CONV (APPEND (CDDR S) Q) H (REPLACE D (CADR S))))
    )))
```

4. DESCRIPTION. HOW SETS CALL EACH OTHER

If R = (N1 C1 N2 C2 ...), where the C_i's are sets of rules and the N_i's, their corresponding names, by the previous implementation of FORMAT it can be seen that:

a. From any C_i we can call any other C_i, e.g. in C4 a rule may exist whose skeleton contains (*REPT* (XXX YYY) C2).

b. Let us assume that some rule of C_i contains skeletons which define new sets; e.g., in C3 there exists the skeleton

(=CONT= (A ZZZ R) M1 (= = = =) M2 (= = = =) M3 (= = = =))

(= = = =): rules that correspond to M1

(= = = =): rules of M2

(= = = =): rules of M3

Then, as before, from any M_i we can call any other M_i; furthermore, from any M_i we can call C3, since C3 contains them; but from no M_i can we call another C_i except C3.

c. From any rule -- at any level -- I can say $\overset{*}{=}$BEGN$\overset{*}{=}$ to the first set of R (not the first set of Q); this allows me to apply the complete transformation process to partial results. This is equivalent in LISP to calling oneself a function, but with different arguments.

d. Presently (see 2.c., FORMAT, page 3-81 of the original), a new Q is being made to which is added N1 R1 N2 R2 ... -- the definitions of new sets -- hence point c. above is covered, and we can still call from M1 to C5, e.g., if the rule C5 ordered us to go to C3 and from there to M1; this property may not prevail in later implementations of CONVERT.

e. Certainly C_i cannot call upon internal sets at another C_i; e.g., from C2 we cannot call set M1, which is in C3, without first going through C3 -- i.e. without first calling C3, and the latter calling M1 (exceptions to this point are found in d.).

5. FUNCTIONS USED

CONVERT* (page 3-3, original), REPLACE (page 3-49, original), CONV (page 3-3, original), APPEND (page 3-69, original).

(=REPT= S1 N1 C1 N2 C2 ...)

(*REPT* S1 N1 C1 N2 C2 ...)

1. S1 skeleton
   N1, N2, ... names (atoms) of the sets of rules C1, C2, ...

2. PROPERTIES

The value of (*REPT* ...) is the content of what (=REPT= ...) would produce. With the present dictionary D, we substitute in the skeleton S1 to form a new expression E; to this expression we apply the set of rules whose name is N1; i.e. we apply to it C1.

In addition to defining N1 as a C1 set, the skeleton may contain the -- additional -- sets of rules C2, C3, ..., with their corresponding names N2, N3, ..., in the form of a dictionary of cycle 2.

The value of the skeleton (=REPT= S1 N1 C1 N2 C2 ..O is the result of applying the rules of C1 to the new expression E formed by S1 replaced.

During the conversion C1 carries out on this expression, we return to the initial dictionary (the one produced by ITLZI-ITLZM), incremented perhaps only by variables defined by ≟QUOT≟, ≟EXPR≟ and ≟SKEL≟.

The above means that all the variables return to their initial state of definition or non-definition; the variables identified by RESEMBLE are not retained on the left side of the rule.

ABBREVIATED FORMS

(≟REPT≟ S1 N1)

Indicates that we will transform the skeleton (previously substituted) S1 by way of the set of rules called N1, which had already been defined before (see also, "How Sets call Each Other," page 3-81, original, to understand what is an "already defined" set of rules).

(≛REPT≛ S1)

Indicates that we will transform the skeleton S1 replaced by means of the current set of rules, i.e. the set which contains this skeleton (≛REPT≛ S1).

Both are abbreviated forms of the more general form given on the previous page; both return to the dictionary L1 (the initial one plus certain "temporary" variables).

Note: The symbol ≛REPT≛ must be understood as: "=REPT= or *REPT*"; use = if the expression is desired into which S1 is transformed; use * if its content is desired; the symbol ≛ does not exist either on the keyboard of the perforator nor on the TELEX, and it is only understood as a convenient notation employed in this Thesis.

4. 'IMPLEMENTATION

The part of REPLACE that manipulates =REPT= is

```
((EQ (CAR S) (QUOTE =REPT=)) (FORMAT D L1 S))
```

The part of REPLACE that manipulates *REPT* is

```
((EQ (CAARS) (QUOTE *REPT*))
  (APPEND (FORMAT D L1 (CAR S)) (REPLACE D (CDR S))))
```

5. FUNCTIONS USED

FORMAT (page 3-80, original), APPEND (page 3-69, original).

(=BEGN= S1)

(*BEGN* S1) 1. S1 skeleton.

2. PROPERTIES

The value of (*BEGN* S1) is the content of the value that would be produced by (=BEGN= S1).

With the present dictionary D, we substitute in skeleton S1 to form a new expression E; to this expression we apply the first set of the argument R of CONVERT, with the dictionary L0.

This means that we apply CONVERT again to the skeleton S1 replaced, with the initial variables, without taking into account =QUOT=, =EXPR=, etc. This conversion produces as a result an expression which is the value of

the skeleton (=BEGN= S1); the value of (*BEGN* S1) is the content of the same.

ABBREVIATED FORMS

=BEGN= has no abbreviated forms.

4. IMPLEMENTATION

The part of REPLACE that operates with =BEGN= is

```
((EQ (CAR S) (QUOTE =BEGN=)) (CONV R LO (REPLACE D (CADR S))))
```

The part of the function REPLACE which corresponds to *BEGN* is none other than

```
((EQ (CAAR S) (QUOTE *BEGN*))
  (APPEND (CONV R LO (REPLACE D (CADAR S))) (REPLACE D CDR S))))
```

5. FUNCTIONS USED

CONV (page 3-3, original). FORMAT is not used.

OPERATORS

Since its implementation consists almost exclusively of primitive functions, it will not be described; nevertheless, the reader may consult the lists found in Chapter VII of this Thesis.

ARITHMETIC SKELETONS

(=PLUS= N1 N2 N3 ...)	N1 + N2 + N3 + ...
(=MINS= N1 N2)	N1 − N2
(=TIMS= N1 N2 N3 ...)	N1 × N3 × N3 × ...
(=DIVD= N1 N2)	N1/N2
(=REMM= N1 N2)	(a) residuo de N1/N2
(=INCP= N1)	N1 + 1
(=DECP= N1)	N1 − 1

Legend: (a) Remainder of N1/N2.

On the 709, the N_i are numerals; the results are numerals. In CONVERT Q-32, the N_i are numbers.

IMPLEMENTATION

The implementation of the arithmetic skeletons follows almost entirely the corresponding functions in MBLISP, hence we only give the list, without discussion.

```
((EQ (CAR S) (QUOTE =PLUS=)) (SUM (REPLACE D (CDR S))))
((EQ (CAR S) (QUOTE =MINS=)) (MINUS (REPLACE D (CDR S))))
((EQ (CAR S) (QUOTE =TIMS=)) (PROD (REPLACE D (CDR S))))
((EQ (CAR S) (QUOTE =DIVD=)) (DIVIDE (REPLACE D (CDR S))))
((EQ (CAR S) (QUOTE =REMN=)) (REMN (REPLACE D (CDR S))))
((EQ (CAR S) (QUOTE =INCR=)) (INCR (REPLACE D (CDR S))))
((EQ (CAR S) (QUOTE =DECR=)) (DECR (REPLACE D (CDR S))))

(SUM (LAMBDA (L) (IF (NULL L) (DEC (QUOTE 0)) (2NDVAL (SPLUS
    (CAR L) (SUM (CDR L)))))))

(MINUS (LAMBDA (L) (2NDVAL (SMINUS (CAR L) (CADR L)))))

(PROD (LAMBDA (L) (IF (NULL L) (DEC (QUOTE 1))
    (2NDVAL (STIMES (CAR L) (PROD (CDR L)))))))

(REMN (LAMBDA (L) (2NDVAL (SDIVIDE (CAR L) (CADR L)))))

(DIVIDE (LAMBDA (L) (1STVAL (SDIVIDE (CAR L) (CADR L)))))
```

(*DESN* S1)

1. S1 skeleton which, after substituted, must be with one ATOM (not a list nor a numeral).

2. PROPERTIES

The value of (*DESN* S1) is the fragment that results from decomposing the value of S1 into its resulting hollerith characters; it is a chain of atoms, each one of which consists of a single character.

3. EXAMPLES

If D = (BR EXPR ESIME), S = ((*DESN* BR)) will produce (E S I M E)

4. IMPLEMENTATION

The part of REPLACE which will disintegrate atoms with *DESN* is

DESN has no counterpart =DESN=.

(*FRAG* S1) 1. S1 skeleton which, after substitution, must be a list. ---

2. PROPERTIES

We replace dictionary D on skeleton 1, yielding an expression whose content is the value of (*FRAG* S1).

3. EXAMPLES

D = (A SKEL (B B B) (B) SKEL (CAM))
S = (RR (*FRAG* A)) will produce
(REPLACE D S) = (RR C A M C A M C A M).

4. IMPLEMENTATION

The part of REPLACE which converts expressions into fragments is

```
((EQ (CAARS) (QUOTE *FRAG*))
    (APPEND (REPLACE D (CADAR S)) (REPLACE D (CDR S)))
```

5. DESCRIPTION

Note that first we carry out the replacement on S1, and then we make an APPEND of the result; thus, if (REPLACE D S1) produces a skeleton which affects CDR, for example, if (REPLACE D S1) = (=SKEL= A B C), then the value of (*FRAG* S1) is =SKEL= A B C, and this fragment is added to the result of replacing CDR, hence it no longer "has any effect." See in this respect "general comment on substitutional pattern-fragments," page 2-19 of the original.

In general, all fragments produces by skeletons of the form (*NAME* ...) are unable to modify the rest of the skeleton to the right; for example, S = ((*UNON* A B C) K L M), if D = (A EXPR (=QUOT= N1) B EXPR () C EXPR (S1)), then the value of (*UNON* A B C) is =QUOT= N1 S1 so that the final result is (=QUOT= N1 S1 K L M); in turn, if the fragment =QUOT= N1 S1 had been generated by a singlet in the SKEL mode, for example, if S = (HHH K L M) and D = ((HHH) SKEL (=QUOT= N1 S1)), then the result of (REPLACE D S) would be M, since it would have defined N1 as S1 and K as L, and it would have substituted in M.

This phenomenon is also present in RESEMBLE, for example, upon defining (PPP) PAT (=DEF= A B C)

Note: to convert a fragment to an expression is a

question of enclosing it between parentheses: if XXX = A B C, then (XXX) will have the value (A B C). However, in a skeleton, to convert an expression to fragment requires the use of *FRAG*; for example, if A = (M N P) and I want to use in a skeleton the fragment M N P, I must say (*FRAG* A); another way of doing the same would be to say (*CONT* A C3 ((=OR=) ==))), C3 is a set whose only rule fails, hence A returns without modification, but *CONT* takes its content.

SAME

Its value is the content of expression E.

((EQ (CAR S) (QUOTE *SAME*)) (APPEND E (REPLACE D (CDR S))))

'EXPR MODE, fragments.

1. D = (... (XXX) EXPR SOCIO ...)
XXX is an atom which appears in D as a singlet under the EXPR mode.
Its partner is a list; its content is called associate fragment.

2. PROPERTIES

An atom under the EXPR mode (fragments) is replaced by the associate fragment, such as that found in D, i.e. without later replacement.

4. IMPLEMENTATION

The part of REPLACE that manipulates atoms representing fragments in the EXPR mode is

```
((EQ (CAR Y) (QUOTE EXPR)) (APPEND (CADR Y) (REPLACE D
                                                (CDR S))))
```

In the previous code, Y is connected to (ASSSOC* (CAR S) D).

SKEL MODE, fragments.

1. D = (... (XXX) SKEL SOCIO ...)
XXX is an atom which appears in the dictionary D under the SKEL mode.
Its partner is a list whose content is treated as a skeleton-fragment.

2. PROPERTIES

An atom under the SKEL mode (fragments) causes its associate fragment to occupy its place in the skeleton and the new (skeleton) thus formed is replaced. Thus within the partner we recognize special symbols; furthermore, there exists the possibility that this atom modify the rest of the skeleton to its right (some examples are given in Chapter II, page 2-36 of the original).

4. IMPLEMENTATION

The part of REPLACE that manipulates atoms that represent fragments in the SKEL mode is

```
((EQ (CAR Y) (QUOTE SKEL)) (REPLACE D (APPEND (CADR Y
                                               (CDR S))))
```

REPT MODE

CONT MODE

1. D = (... (XXX) REPT (R1 R2 ...) ...)

XXX is an atom that appears in D as a singlet under the REPT mode.
Its partner is a set of rules.
It is a list.

2. PROPERTIES

The skeleton (XXX σ_1 σ_2 ...), where XXX is in the REPT mode -- as a singlet -- causes the following:

a. The present dictionary is replaced in the list (σ_1 σ_2 σ_3 ...) [i.e. in the CDR of the skeleton], and an expression is obtained (not a fragment) as the result or value.

b. To the value obtained above in a. the partner of XXX is applied (which is a set of rules), under the L1 dictionary (since a REPT is involved), and we obtain an expression which is the value of the skeleton (XXX σ_1 σ_2 ...).

The CONT mode operates in the same manner, no more than we use L* instead of L1; i.e. we preserve all the variables upon applying the set to (σ_1 σ_2 ...).

In summary: (XXX σ_1 σ_2 ...), with (XXX) CONT ((P1 S1) (P2 S2) ...), is equivalent to (=CONT= (σ_1 σ_2 ...) ** ((P1 S1) (P2 S2) ...)).

The CONT and REPT modes always yield expressions as a value, and it is necessary to use *FRAG* to obtain fragments.

3. EXAMPLES

```
M = ((CD) REPT ( ((X XXX) (XXX)) ) A EXPR B)
I = (X (XXX) (YYY) (ZZZ) Y Z)
E = ( (M A R) (C I A N) (O S))
R = (C1( (( X Y Z) ((CD X) (CD Y) (CD Z)))
    ) )
```

then (CONVERT M I E R) will be ((A R) (I A N) (S)), since CD takes CDR from its "arguments;" these REPT and CONT modes are similar to the functional notation used in LISP.

4. IMPLEMENTATION

The part of REPLACE that manipulates the REPT and CONT modes is

```
((EQ (CAR Y) (QUOTE REPT)) (CONVERT* (CADR Y) L1
     (REPLACE D (CDR S))))
((EQ (CAR Y) (QUOTE CONT)) (CONVERT* (CADR Y) L*
     (REPLACE D (CDR S))))
```

5. FUNCTIONS USED

CONVERT* (page 3-3 of the original).

RECURSIVE PART OF REPLACE

The skeletons described above are the so-called basic or primitive skeletons, i.e. those that replace recognizes immediately.

More complex skeletons are evaluated or substituted, making a CONS of the result of substituting CAR with the result of substituting CDR; the corresponding code is

((AND) (CONS (REPLACE D (CAR S)) (REPLACE D (CDR S))))

The recursive condition, such as it is implemented, allows skeletons of the following type to have meaning:

(V W =UNON= (A B C) (C D E)) = (V W A B C D E) = (V W (*UNON* (A B C) (C D E)));

and the rule is that if contained in a skeleton, the last element is one of the form (*NAME* σ_1 σ_2 ...), it may be replaced by (the fragment) =NAME= σ_1 σ_2 ...

The above rule may not be applicable for future CONVERT processors; I do not consider it an essential part of the language.

(EDIT L)

1. L dictionary of cycle 3, launched by RESEMBLE whenever similarity is encountered.

2. FUNCTION

The value of (EDIT L) is a dictionary similar to L, but modified to the modes characteristic of REPLACE.

The modifications are:

CYCLES IN L	CYCLES IN (EDIT L)
X VAR SOCIO	X EXPR SOCIO
(XXX) VAR (apuntador (a) apuntador)	(XXX) EXPR (lista restaurada (b))
(XXX) REC (numeral numeral)	XXX EXPR átomo (c) corresp. al 2° numeral
X SEQ (pat A B C ..)	X EXPR (A B C ..)
X MSS (pat A B C ..)	X EXPR (A B C...)
X EXPR SOCIO	X EXPR SOCIO
X SKEL SOCIO	X SKEL SOCIO
(XXX) REPT SOCIO	(XXX) REPT SOCIO
(XXX) CONT SOCIO	(XXX) CONT SOCIO

the other modes will not be copied

Legend: (a) Observer; (b) Restored list; (c) Atom corresponding to the second numeral.

3. DEFINITION

The definition of EDIT is

```
(EDIT (LAMBDA (L) (COND
    ((NULL L) L)
     ((EQ (CADR L) (QUOTE VAR)) (CONS (CAR L) (CONS (QUOTE EXPR) (CONS
          (IF (ATOM (CAR L)) (CADDR L)
          (RESTAUR (CAADDR L))) (EDIT (CDDDR L))) )))
      ((EQ (CADR L) (QUOTE REC)) (CONS(CAAR L) (CONS (QUOTE EXPR) (CONS
           (UNDEC (CADDR L)) (EDIT (CDDDR L))) )))
    ((OR (EQ (CADR L) (QUOTE SEQ)) (EQ (CADR L) (QUOTE MSS)))
         (CONS (CAR L) (CONS (QUOTE EXPR) (CONS (CDADDR L)
         (EDIT (CDDDR L))))))
    ((OR (EQ (CADR L) (QUOTE EXPR)) (EQ (CADR L) (QUOTE SKEL))
         (EQ (CADR L) (QUOTE CONT)) (EQ (CADR L) (QUOTE REPT)))
         (CONS (CAR L) (CONS (CADR L) (CONS (CADDR L) (EDIT (CDDDR L))
         ))))
    ((AND) (EDIT (CDDDR L)))
    )))
```

5. FUNCTIONS USED

RESTAUR (described below).

RESTAUR [restore] passes from the notation of two observers to that of a list; for example, E = ((C A M I S A S) (S A S)) is transformed into C A M I).

```
(RESTAUR (LAMBDA (E) (IF (                          EQ
    E (CADR (CADDR L))) (LIST) (CONS (CAR E) (RESTAUR (CDR E))) )))
```

The definition of EDIT in the CONVERT version found running in the Q-32 computer of Santa Monica, California, USA, is

```
          (EDIT (LAMBDA (L)
(COND ((NULL L) L)
 ((EQ (CADR L) (QUOTE VAR))
  (CONS (CAR L)
   (CONS (QUOTE EXPR)
    (CONS (IF (ATOM (CAR L)) (CADDR L) (RESTAUR (CAADDR L)))
     (EDIT (CDDDR L))))))
 ((EQ (CADR L) (QUOTE REC))
  (CONS (CAR L)
   (CONS (QUOTE EXPR) (CONS (CADDR L) (EDIT (CDDDR L))))))
 ((OR (EQ (CADR L) (QUOTE SEQ)) (EQ (CADR L) (QUOTE MSS)))
  (CONS (CAR L)
   (CONS (QUOTE EXPR) (CONS (CDADDR L) (EDIT (CDDDR L))))))
 ((OR (EQ (CADR L) (QUOTE EXPR))
   (EQ (CADR L) (QUOTE SKEL))
   (EQ (CADR L) (QUOTE CONT)) (EQ (CADR L) (QUOTE REPT)))
  (CONS (CAR L)
   (CONS (CADR L) (CONS (CADDR L) (EDIT (CDDDR L))))))
 (T (EDIT (CDDDR L))))))
```

CHAPTER IV

HOW TO USE THE CONVERT SYSTEM

As can be observed in
Chapter VII, CONVERT
is a LISP function,
hence the reader familiar
with MBLISP or with
LISP 1.5 may omit
Chapter IV.

IV. HOW TO USE THE CONVERT SYSTEM

Presently, CONVERT is implemented on the IBM-709 computer of the Instituto Politecnico Nacional, and AN/FSQ-32 of Systems Developemnt Corporation, Santa Monica, California, USA; in both it is interpreted by LISP.

4.1 PREPARATION OF A PROGRAM

As in LISP, in the CONVERT programs the excess blanks are ignored, and one is sufficient to designate atoms. Furthermore, parentheses have the common meaning they have in LISP, i.e. they are used to delimit lists.

Let us assume that we want to run a program alternating two lists of the same length, but one from left to right, and the other from right to left, i.e.

If A = (A1 A2 ... An)
B = (B1 B2 ... Bn)
the result must be (A1 Bn A2 B_{n-1} A3 B_{n-2} ... An B1).

The definition of this function could be:

(ALTERNR A B)

1. A, B lists of equal length (same number of elements).

2. FUNCTION

The value of (ALTERNR A B) is the list that contains the elements of A in the same order and, alternatively with them, but in the inverse order, the elements of B.

Since CONVERT only operates with a single expression, and ALTERNR is a function of two variables, we make a list of them to produce the argument E of CONVERT.

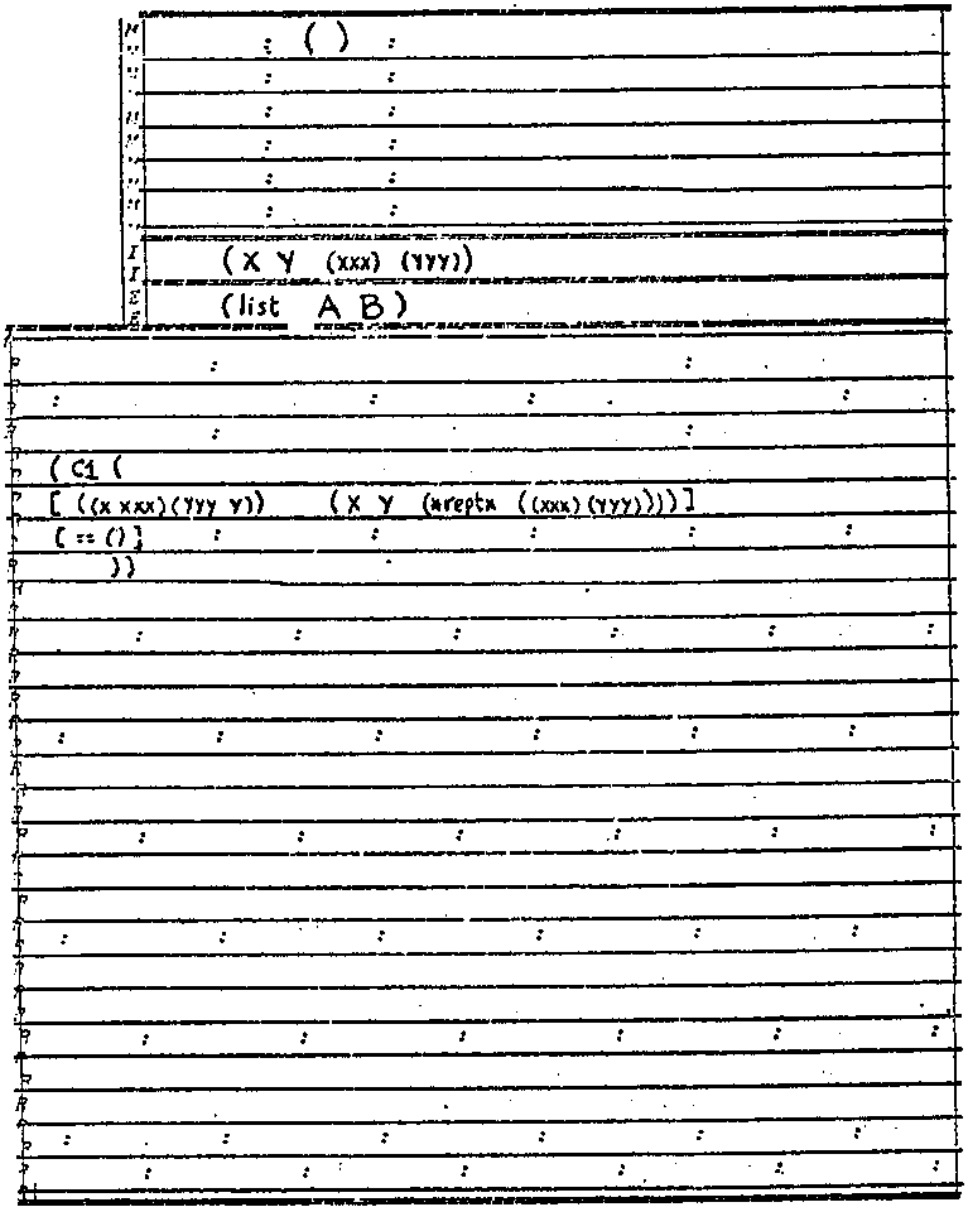
In reality, the program consists of the single rule

(((X XXX) (YYY Y)) (X Y (*BEGN* ((XXX) (YYY)))))

The complete program appears on the following sheet.

Finally, the definition in LISP is the following:

```
(ALTERNR (LAMBDA (A B) (CONVERT  (QUOTE ())
                                 (QUOTE (X Y (XXX) (YYY)))
                                 (LIST A B)
                                 (QUOTE (C1 (
(((X XXX) (YYY Y)) (X Y (*REPT* ((XXX) (YYY))) ))    (== ())
)))
)))
```



CONVERT coding sheet 6/20/66 CIEA-IPN

Note the (QUOTE ...)'s introduced in M, I and R.

We must define this function; on the 709 this is done with (DEFINE (ALTERNR ... ...))

This means ahead of the cards constituting the definition of ALTERNR we place the card (DEFINE and at the end of the card).

In Q-32, we would say

DEFINE (((ALTERNR ... ...)))

If we wish to use this program to "alternate" lists (1 2 3 4 5) and (A B C D E), on the Q-32 computer we would say (we would write), after having defined ALTERNR,

ALTERNR ((1 2 3 4 4) (A B C D E))

Note: name of the function and a list with the arguments.

and the contestation would be:

(1 E 2 D 3 C 4 B 5 A)

In MBLISP, 709, following the cards (DEFINE ... ...) we would put

(APPLY ALTERNR ((1 2 3 4 5) (A B C D E)))

Note: APPLY — name of the function — list of arguments

With the result being the same, sinc (1 E 2 D 3 C 4 B 5 A)

4-2. 709 OPERATION

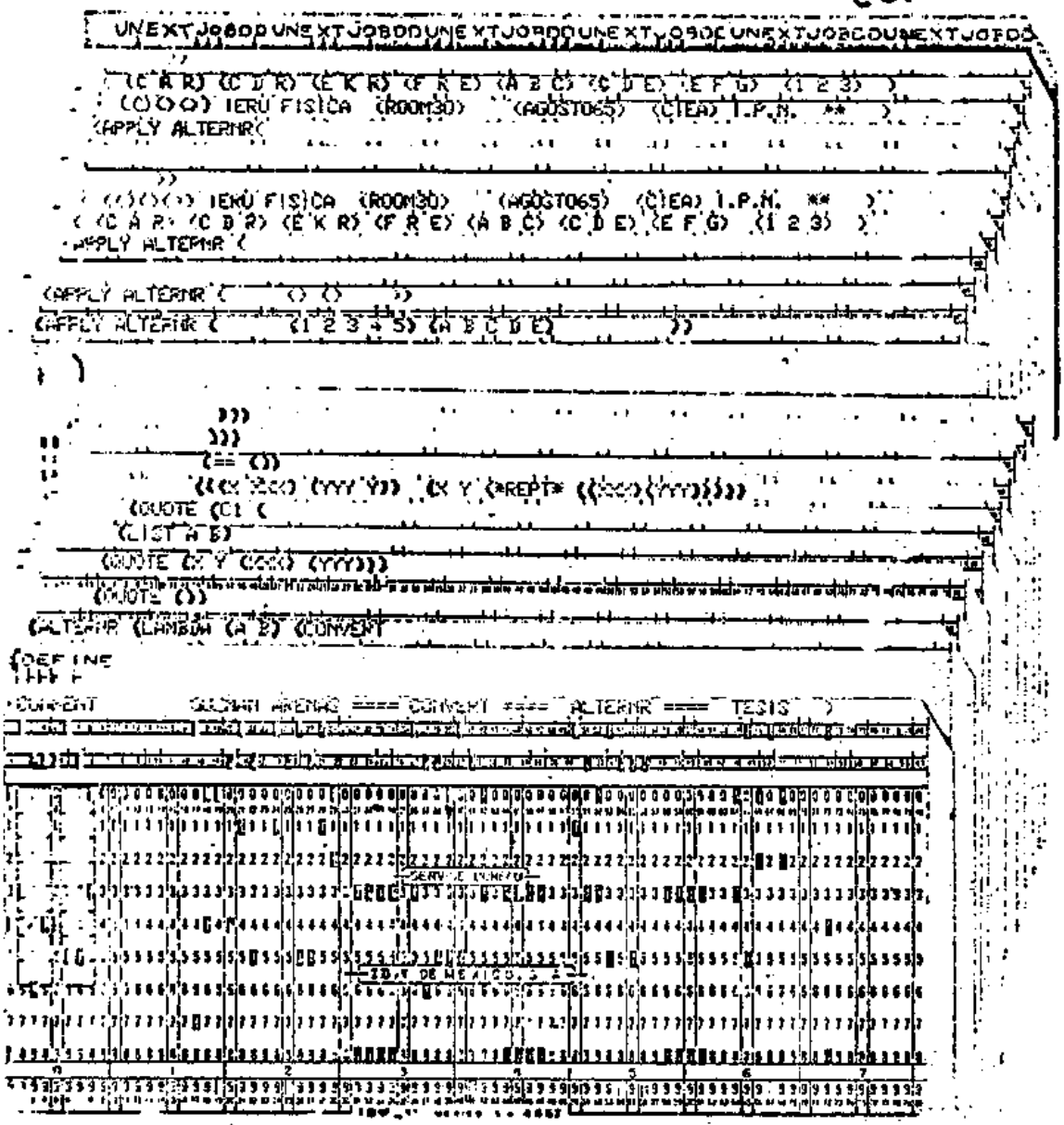
1. CONVERT tape in B-7
2. Program in A-2 (input tape). The program used as an example is shown on page 4-4, original, such as it must read on tape A-2. Note the cards of COMMENT, NEXTJOB and EOF.
3. Call CONVERT from B-7; Card LOAD B7 in the reader.
   Start in the reader; (READY light)
   reset
   load cards in CPU 709

The result appears on the A-3 tape (output).

4-3. OPERATION ON Q-32 TSS (Time Sharing System)

1. LOGIN

eof



2. LOAD CONVERT

3. DEFINE ((
(ALTERNR....)
))

4. ALTERNR ((1 2 3 4 5)
(A B C D E)
)

5. other examples

As we see, to use CONVERT language, LISP functions are defined which use the CONVERT function.

4.4 ERROR MESSAGES

Common error messages in LISP are handled, such as MISSING ARGUMENT, PUSHDOWN LIST EXHAUSTED, OUT OF BPS, etc., and, on the 709, if the job is to make an APPEND and one of the arguments is an atom (see CONVERTERROR, page 3-8 of the original).

Program Aids

The pseudo-rule (=COM= here any convenient comment) allows us to insert comments in the body of the program; it will be ignored by the processor. If the comments have parentheses, they must be balanced.

The pseudo-rule (=PRI= S1) prints skeleton S1, replaced, but is otherwise ignored by CONVERT.

It is useful, particularly on the 709, to trace programs, since partial results may be printed, for example, with

(=PRI= =SAME=)

Or even with some more elaborate card:

```
(=PRI= (EL ARGUM (=QUOT= X) ES X == (=QUOT= YYY) ES YYY ==
       [the argument]       [is]                E ES =SAME= ))
```

Furthermore, the functions LISP PRINT, TRACE and (in MBLISP) MUSMEA, are disposed of, to trace LISP functions; for example, it is useful to scent RESEMBLE or REPLACE; EDIT.

The skeleton is also disposed of in CONVERT (=PRNT= S1), whose value is the result of replacing S1, but which prints on the output tape [in Q-32, it prints on the teletype] the value of S1 replaced.

The description of =PRI= is found on page 3-4, original, under CONV*. The description of =PRNT= is found on page 3-59, original.

```
   TAPE A-2          REWOUND       +0 (OCTAL) RECORDS


(COMMENT          GUZMAN ARENAS ==== CONVERT ====  ALTERNR ====  TESIS    )
(DEFINE
(ALTERNR (LAMBDA (A B) (CONVERT
     (QUOTE ())
     (QUOTE (X Y (XXX) (YYY)))
     (LIST A B)
     (QUOTE (C1 (
          (((X XXX) (YYY Y)) (X Y (*REPT* (XXX)(YYY)))))
          (== ())
          )))
          )))
```

```
   TAPE A-3          REWOUND       +0 (OCTAL) RECORDS

(COMMENT GUZMAN ARENAS ==== CONVERT ==== ALTERNR ==== TESIS)•••••

(DEFINE (ALTERNR (LAMBDA (A B) (CONVERT (QUOTE ()) (QUOTE (X Y (XXX) (Y
YY))) (LIST A B) (QUOTE (C1 (((X XXX) (YYY Y)) (X Y (*REPT* (XXX) (YY
Y))))) (== ()))))))))••

FUNCTIONS DEFINED •••      ALTERNR•

(APPLY ALTERNR ((1 2 3 4 5) (A B C D E)))
(1 E 2 D 3 C 4 B 5 A)••

(APPLY ALTERNR (() ()))
()•••

(APPLY ALTERNR (((C A R) (C D R) (E K R) (F R E) (A B C) (C D E) (E F G
) (1 2 3)) ((() () ()) IERU FISICA (ROOM30) (AGOST065) (CIEA) I•P•N• **
                                                  [August]
)))••

((C A R) ** (C D R) I•P•N• (E K R) (CIEA) (F R E) (AGOST065) (A B C) (R
OOM30) (C D E) FISICA (E F G) IERU (1 2 3) (() () ()))•••••

(APPLY ALTERNR ((((() () ()) IERU FISICA (ROOM30) (AGOST065) (CIEA) I•P•
N• **) ((C A R) (C D R) (E K R) (F R E) (A B C) (C D E) (E F G) (1 2 3)
)))••

((() () ()) (1 2 3) IERU (E F G) FISICA (C D E) (ROOM30) (A B C) (AGOST
065) (F R E) (CIEA) (E K R) I•P•N• (C D R) ** (C A R))•••••

(NEXTJOB)••
```