Bob Yates

A HANDBOOK OF LISP FUNCTIONS

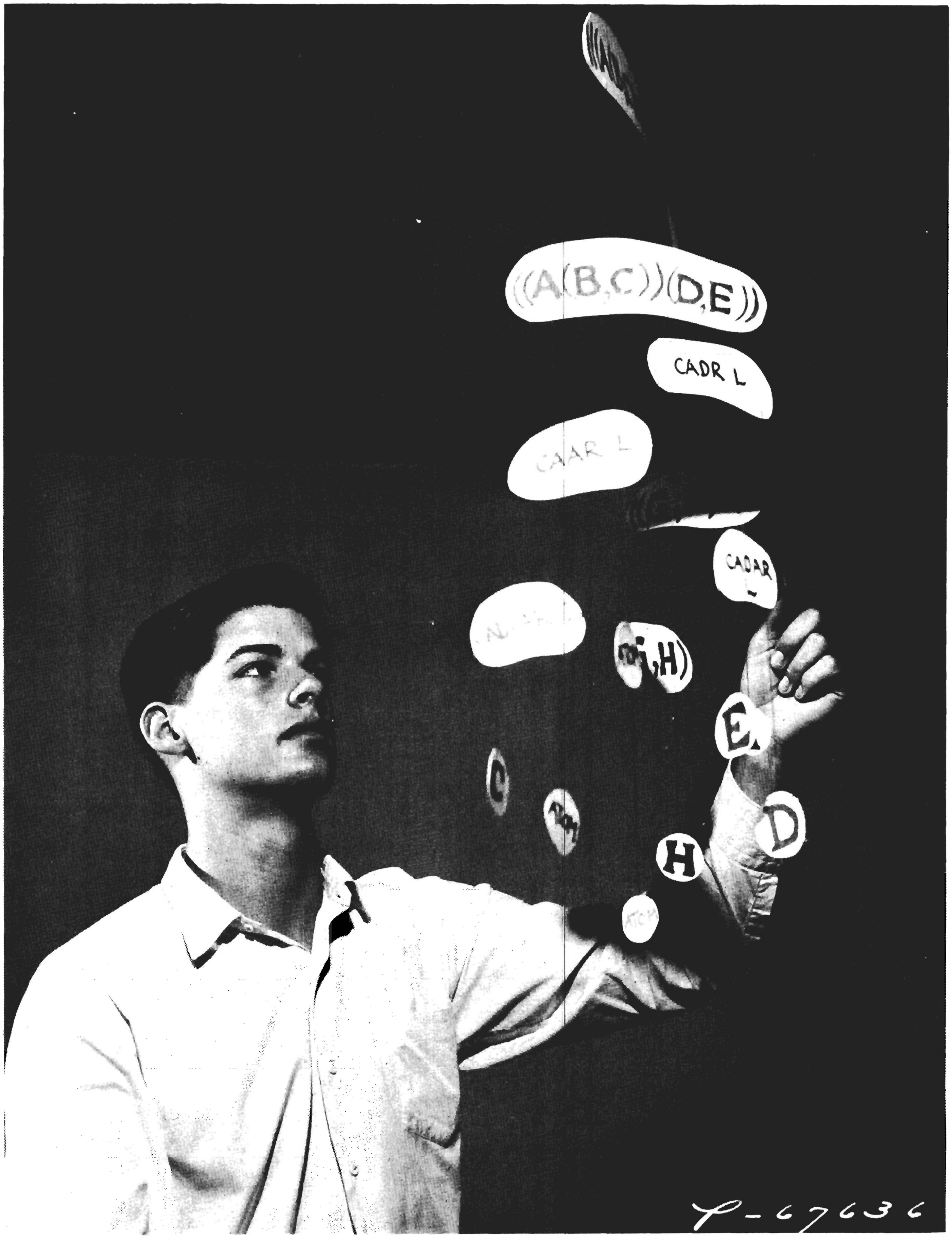
RIAS TECHNICAL REPORT 61-11

A HANDBOOK OF LISP FUNCTIONS

August, 1961

RIAS, 7212 Bellona Avenue, Baltimore, Md.

TABLE OF CONTENTS


((A,(B,C))(D,E))
CADR L
CAAR L
CADAR L
(G,H)
E
D
H
C
P-67636

STUDENTS IN THE SEMINAR ON THE THEORY AND APPLICATION OF LARGE COMPUTERS

Tony Conrad

Peter Conrad

Peter Hartline

Joe Schlessinger

Bob Yates

Richard Evans

William Skeen *

Bari Thompson

Judy Barnes *

Dan Hartline

John D. Baildon *

Robert Thompson *

*students who have prepared this manual.

(opposite --- Peter Conrad examining his mobile which illustrates the list structure of LISP)

INTRODUCTION

RIAS has been fortunate in obtaining informally a copy of the LISP system tape, as developed at MIT by Prof. McCarthy and his co-workers. This has permitted us to do a certain amount of experimentation with LISP as a programming language, and evaluate its performance with respect to other systems, such as SOS, or FORTRAN. In some ways, such a comparison is a little one-sided, since both the latter systems are much more elaborate than LISP, and consequently permit the solution of problems which would not be attempted in LISP. Nevertheless, LISP emerges as a surprisingly powerful language for all its simplicity in being based on a vocabulary of only five instructions. It not only has had considerable merit as a pedagogical tool, for teaching the essence of programming with respect to a readily comprehensible programming language; but as well shows considerable versatility with respect to some more serious problems which would require considerable programming effort in either SOS or FORTRAN. As in any system, growing skill shows the way to circumvent many of the inadequacies of the system, which at first sight seem to be very serious flaws. In some instances, certain errors to which the system is prone have been turned to advantage, and exploited to obtain some rather nice results. An example is the LISP page plotter (which still contains a disconcerting abundance of commas), which was inspired by the pages of "wallpaper" which one frequently generates by having tried to take CAR of an atom.

This manual is intended as a supplement to the LISP PROGRAMMERS MANUAL which accompanied our system tape, and is a collection of a number of LISP functions which have been found either interesting and/or useful. Some of the description concerns the mechanics of the MARTIN monitor system, since the LISP tape had to be modified to take into account the lack of an on-line card reader for MARTIN'S IBM 7090, and the incompatibility of the original clock routine; but the general reader will easily spot and ignore the few paragraphs of discussion given to this question.

For the most part, the manual has been prepared by the students of the RIAS Summer Institute in Computer Theory. We are extremely

pleased to acknowledge a considerable amount of advice and assistance, thanks to which we have had the opportunity to use LISP; we particularly thank Wayne Wiitanen and Steve Russel for assistance in obtaining the tape and advice in its use, as well as Solis James, who by his interest and encouragement greatly speeded our procurement, development and understanding of the system.

HAROLD V. McINTOSH

BALTIMORE, August 18, 1961.

BUILDING A LIBRARY OF LISP FUNCTIONS

Although it is possible to modify the LISP tape so that it will contain the definitions of additional functions, it is not advisable to do so, save for functions of the most basic and elementary nature. The reason is that although a problem area may have its useful specialized functions, if one is to adjoin to the tape all the functions from a variety of areas, he has filled it with what will be dead weight in any one problem, as well as increasing the chances of conflicting function names and the like.

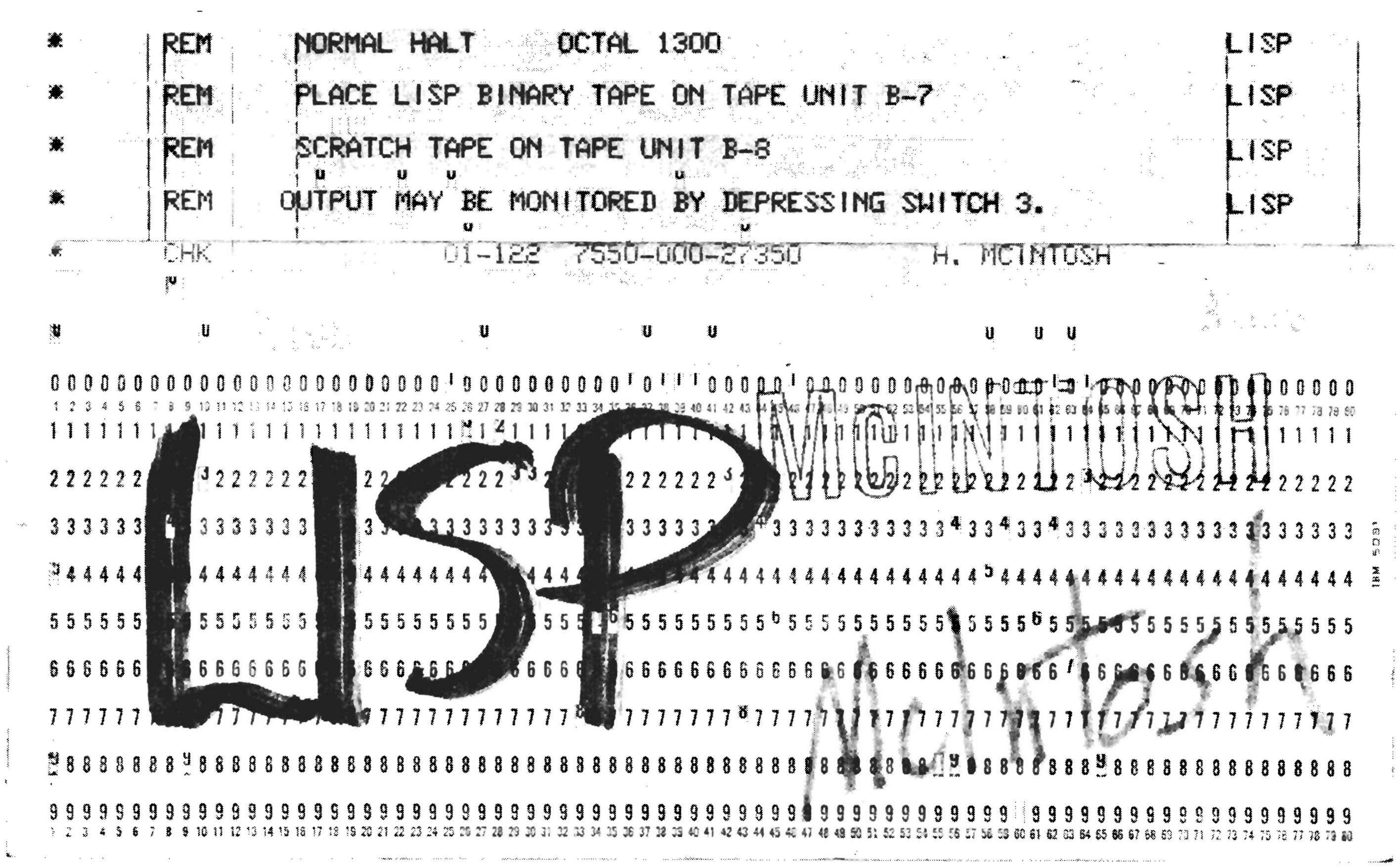
The format of LISP is somewhat inconvenient for the usage of library functions, since one expects to surround the function definitions with DEFINE ((...)). The parentheses so required lend an assymmetry to the first and last functions to be defined. However, this problem is easily remedied by keeping two cards, imprinted with, respectively,

```
DEFINE(((BEGIN (LAMBDA () NIL))
 (END (LAMBDA () NIL))))
```

By then confining all other function definitions to a separate card (or cards), they may be easily selected from a library collection, and inserted along with any freshly defined functions similarly prepared, between the BEGIN and END cards.

SUBMITTING A LISP PROGRAM

The program deck for submitting a LISP run consists of the following cards:

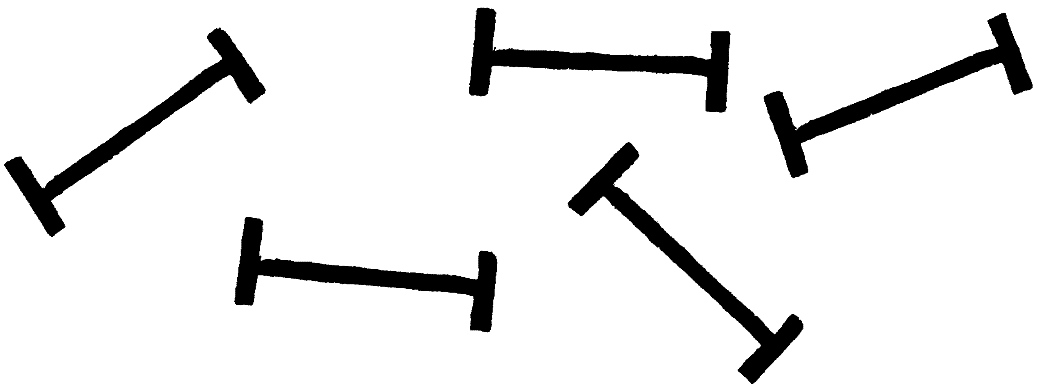


```
*     REM    NORMAL HALT     OCTAL 1300                          LISP
*     REM    PLACE LISP BINARY TAPE ON TAPE UNIT B-7             LISP
*     REM    SCRATCH TAPE ON TAPE UNIT B-8                       LISP
*     REM   OUTPUT MAY BE MONITORED BY DEPRESSING SWITCH 3.      LISP
*     CHK              01-122  7550-000-27350        H. MCINTOSH
```

LISP
MCINTOSH
McIntosh

6. One card load tape button simulator
7. One of the overlord cards; SET, TST, or MLT
8. The body of the source program
9. Stop card STOP))))))))))))))))))
10. Items 4-5-6 may be repeated as often as desired (up to 100)
11. FIN card

The five primitive LISP functions



THE FIVE PRIMITIVE LISP FUNCTIONS

LISP is a programming language, constructed entirely from three functions and two predicates. Out of these, or their composites, all program functions will be made. These five functions, together with examples of their use, are listed below;

1. (ATOM X) is a predicate which takes the value TRUE only when the expression X is an atomic symbol. If X is a list, or anything but a single atom it will assume the value FALSE.

2. (EQ X Y) is defined when one of the two expressions X or Y is an atom, and is TRUE if they are identical; otherwise it is FALSE.

3. (CAR L) If L is a list; L = (A B C D ...), then (CAR L) applied to the list = (A). If L is not a list, (CAR L) is undefined, and will give a page full of commas if attempt is made to print the results.

```
Examples;  If  L = (X Y)              (CAR L) = X
               L = (X Y Z W)          (CAR L) = X
               L = ((X Y) Z W)        (CAR L) = (X Y)
               L = ((X Y))            (CAR L) = (X Y)
               L is atomic            (CAR L) is undefined
```

4. (CDR L) If L is a list; L = (A B C ...), then (CDR L) will yield (B C ...).

```
Examples;  If  L = (X Y Z W)          (CDR L) = (Y Z W)
               L = (X Y)              (CDR L) = (Y)
               L = (X)                (CDR L) = NIL = ()
               L = ((X Y) Z)          (CDR L) = (Z)
               L is atomic            (CDR L) is undefined
```

5. (CONS A B) If B is a list, then (CONS A B) is a list, whose first element is A, and whose remaining elements are the elements of B. If B is not a list, (CONS A B) is technically a list, but will cause bizzare behaviour by the print routine.

```
Examples;      (CONS X NIL) = (X)
               (CONS (X) (Y)) = ((X) Y)
               (CONS (CAR L) (CDR L)) = L     when L is a list.
```

It will be found, upon writing programs in LISP, that composites of CAR and CDR are necessary and can get very lengthy. These can be abbreviated in the following fashion; (CAR (CDR L)) can be written as (CADR L), and similarly; (CAR (CAR (CAR (CDR L)))) becomes (CAAADR L), and (CDR (CAR (CDR L))) becomes (CDADR L). By applying various composite functions to the expression (X (A B C) D) = L, we get the following results:

```
(CAR L) = X
(CDR L) = ((A B C) D)
(CADR L) = (A B C)
(CAADR L) = A
(CDADR L) = (B C)
(CADAR L) = B
(CDDADR L) = (C)
(CAR (CDDADR L)) = C
(CDR (CDDADR L)) = NIL = ()
(CDDR L) = (D)
(CADDR L) = D
(CDDDR L) = NIL = ()
```

The programming functions

THE PROGRAMMING FUNCTIONS

While the basic elements of LISP are the five previously discussed functions, there must also exist a set of programming functions to enable one to write logical programs. These are discussed below:

(DEFINE ((N_1 E_1) (N_2 E_2)...))

DEFINE is one of three ways, in addition to LAMBDA and LABEL, by which a function may be defined in LISP. DEFINE has as its argument a list of pairs, which may be of unspecified length. Each is composed of a name N_i and an expression E_i by which it is to be defined. This allows the function name to be treated as an atomic expression in defining other functions, and its LAMBDA expression need not be continually repeated.

One can use DEFINE to define recursive functions without the use of LABEL, since DEFINE assigns the name of the entire expression properly. It is important to note that the argument of DEFINE is a single list, and to therefore use the proper number of parentheses.

(LAMBDA (X Y ...) E)

LAMBDA is a special form used in defining expressions. It has two arguments, namely a list of arguments, and a sample of the function, E. LAMBDA is necessary when the arguments comprising the example are hopelessly mixed, and we must make unambiguous which of the dummy variables in the argument list corresponds to a given occurence of the variable in the sample.

(LABEL A E)

LABEL is a special form used in the definition of recursive functions. It has two arguments, one the name of the function, and the other the expression defining the function. This is to allow usages of A within E to be interpreted as standing for the function as a whole.

(COND (K_1 F_1) (K_2 F_2) ...)

COND is a special form used in function definitions. It is a function of an indeterminate number of arguments, each of which is itself a pair, consisting of a predicate and a function definition.

A function defined with the aid of COND is evaluated by considering the list of pairs in sequence. For the first predicate K_i which is true, the function is defined as F_i. Thus COND is a compact way of making a conditional function definition. Each F_i may in its turn be defined in terms of COND, and thus one may make explicit use of the conditional. For each K_i, we may assume all its predecessors to be false. At least one predicate must be true, otherwise one will get an error return. Usually K_n is T. and covers the remaining case.

(QUOTE X)

QUOTE is a function of one argument whose value is that argument. Its principle use is to introduce atomic symbols into a source program, which would otherwise be undefined, or possibly be interpreted as further functions if they corresponded to the name of a function already defined.

(LIST X Y ...)

LIST is a function of an unspecified number of arguments whose value is simply the list of its arguments.

```
                         (LIST X Y Z) = (X Y Z)
                         (LIST (X) (Y)) = ((X) (Y))
    whereas              (CONS (X) (Y)) = ((X) Y)

    thus                 (LIST (CAR L) (CADR L)) = L
    whereas              (CONS (CAR L) (CDR L)) = L
    when L = (X Y)
```

(EQUAL X Y)

EQUAL is a predicate which takes the value true if and only if X and Y are identical. It is gotten by applying EQ to all components of X and Y, to see if they are made up of the same atoms, similarly arranged.

(AND X Y ...)

AND is a predicate which is a function of an unspecified number of arguments, one or more. It takes the value true if all its arguments and false otherwise. However, it is not quite the same as the ordinary "and", since its arguments form an ordered set and need not be commutative. This is because the list of arguments is examined in order from left to right; if one is found to be false, AND takes the value false, if none are found false, the value is true. Thus, any argument may assume its predecessors are true, with a consequent economy realizable in specifying the arguments. Thus, a series of more restrictive predicates may be listed, by specifying only the additional conditions in each successive argument. Rearranging such a sequence may allow some of the argument to be undefined.

(OR X Y ...)

OR is a predicate which is a function of an unspecified number of arguments, taking the value true if any one is true, and the value false only if none are true.

Since, like AND, its arguments are examined in order, economy may be achieved in the writing of any one argument by assuming that all its predecessors are false, since otherwise it would never be examined. Since some of the arguments may be undefined in some cases, as well as the possibility of such an assumption, it is not a commutative function of its arguments.

(NOT X)

NOT is a predicate which is true if its argument is false and false if its argument is true.

(NULL L)

NULL is a predicate whose argument is a list. It takes the value if and only if its argument is the atomic symbol NIL. Since NIL is not entered explicitly, but indicated by an address zero, one is safest in assuming that NULL is true when L is empty. An atom does not form an empty list.

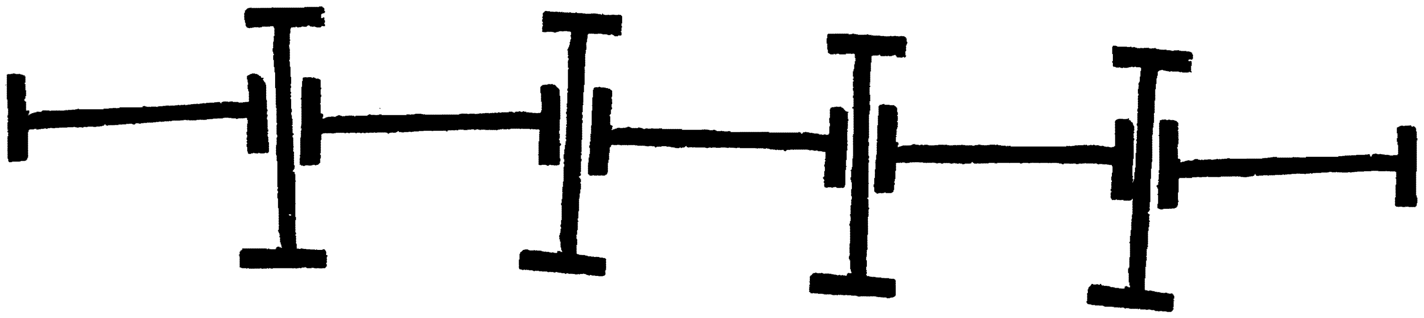
NIL is an atomic expression used to signal the end of a list. It does not appear in output, but is used in the source language to terminate a list. For instance, (CONS X NIL) will produce the list (X); NIL itself if printed, (). The predicate NULL takes the value T when applied to NIL; that is, a list is null when its only element is NIL.

The use of NIL permits the construction of arbitrary lists in the context of a framework which is essentially binary.

T is an atomic symbol, indicating the value true of a predicate. The machine output represents T by a blank.

F is an atomic symbol, indicating the value false of a predicate. The machine output represents F by the symbol ().

The constructive functions



MAPCON

I. MAPCON is a function which applies a given function to every element of a given list and lists the results. MAPCON is predefined in the LISP system and does not need to be redefined.

II. This function has two arguments; the first is a list, and the second is a function. MAPCON creates the new list by using the function APPEND (rather than CONS, as in MAPLIST).

FUNCTION

I. FUNCTION is a function which defines its argument as a function to the computer. FUNCTION is predefined in the LISP system and needs no redefinition.

II. FUNCTION has one argument, which it defines as a function.

APPEND

I. APPEND joins two lists to form a new single list by placing the second at the end of the first. APPEND is part of the LISP system and need not be defined in a program. (If atoms are used, the value of APPEND is the first atom.)

II. Examples.

```
APPEND
((PARITY,X,(X,Y,Z,X)),(SUBSET,(A,B,C),(A,R,B,D,E)))

(PARITY,X,(X,Y,Z,X),SUBSET,(A,B,C),(A,R,B,D,E))

APPEND
( ,(A,B,C))

(A,B,C)

APPEND
((A,B,C),(D,E,F))

(A,B,C,D,E,F)
```

LIST

I. LIST is a function whose resulting value is the list which is its argument. It is a part of the LISP system and need not be defined in a program. Any number of variables may be used.

II. Examples.

```
LIST
((ELEMENT,X,((X,Y))),(SUBSET,((X,Y)),((X,Y,Z))),(PARITY,X,((X,Y))))

((ELEMENT,X,((X,Y))),(SUBSET,((X,Y)),((X,Y,Z))),(PARITY,X,((X,Y))))


LIST
( , , )

( , , )

LIST
((A),(B),(C))

((A),(B),(C))

LIST
(A,B,C,D,E)

(A,B,C,D,E)
```

MAPLIST

I. MAPLIST is a function which applies a given function to every element of a given list and lists the results. This function is a part of the LISP system and does not need to be redefined.

II. This function has two arguments; the first is a list and the second is a function. (The new list is created by the basic function CONS.)

III. Examples.

```
MAPLIST
((FUNCTION.(LAMBDA.(X.L),(ELEMENT.(X.(Y.X.Z))))),(FUNCTION.
(LAMBDA,(P),(TRUTHVALUE,P))))

(TRUE,TRUE)


MAPLIST
(((E,S,E),(H,T,E),(I,B,G)),(FUNCTION,(LAMBDA,(L),(ENGLISH,L))))


(((H,T,E),(E,S,E),(I,B,G)),((I,B,G)), )
```

(FORALL L P)

I. FORALL takes a value of true if property P is true for every element of L and a value of false otherwise.

II. FORALL is a predicate having two arguments; (L P) where
L is a list of any length.
P is a property.
(NOTE) When applying FORALL a special form of writeup is required. This is shown below.

III. Machine definition.

```
(FORALL (LAMBDA (L P)
(COND ((NULL L) T)
((P (CAR L)) (FORALL (CDR L) P))
(T F))))
```

IV. Examples.

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
(LAMBDA,(L),(FORALL,L,(FUNCTION,CDDPLET)))
(((A,B,C),(D,E,F),(G,H,I)))


END OF APPLY, VALUE IS ...
( )         TRUE [INDICATED BY THE PRESENCE OF ( )]
```

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
(LAMBDA,(L),(FORALL,L,(FUNCTION,ODDPLET)))
(((A,B,C),(D,E),(F,G,H)))


END OF APPLY, VALUE IS ...
              FALSE, SINCE THERE APPEARS NO ( )
```

(FORSOME L P)

I. FORSOME takes the value true if a specified property holds true for any element of a given list.

II. FORSOME is a predicate having two arguments; (L P) where
L is a list of any length.
P is a property to be checked for.

III. Machine definition.

```
(FORSOME (LAMBDA(L P)
(COND ((NULL L) F)
((P (CAR L)) T)
(T (FORSOME (CDR L) P)))))
```

IV. Examples.

(FORODD L P)

I. FORODD is a predicator which takes the value true if and only if a given predicate is true for an odd number of elements of a given list.

II. FORODD is a function of two arguments: (L P)

L is a list of lists.

P is a predicate.

III. Machine definition.

```
 (FORODD (LAMBDA (L P)
   (COND ((NULL L) F)
 ((P (CAR L)) (NOT (FORODD (CDR L) P)))
 (T (FORODD (CDR L) P)))))
```

IV. Examples.

```
 FUNCTION   APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FORODD
(((A,B),(A),(A,B,C)),(FUNCTION,(LAMBDA,(L),(ULTRATRIPLET,L))))


END OF APPLY, VALUE IS ...
( )         TRUE  [INDICATED BY THE PRESENCE OF THE ( )]
```

```
 FUNCTION   APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FORODD
(((A,A,C),(B,C),(B,B)),(FUNCTION,(LAMBDA,(L),(EQUAL,(CAR,L),(CADR,L)))))


END OF APPLY, VALUE IS ...
            FALSE, SINCE THERE APPEARED NO ( )
```

```
 FUNCTION   APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FORODD
(((A,B),(A,A),(C,D),(E,F)),(FUNCTION,(LAMBDA,(L),(EQUAL,(CAR,L),(CADR,L)))))


END OF APPLY, VALUE IS ...
( )         TRUE  [INDICATED BY THE PRESENCE OF THE ( )]
```

(ELEMENT X L)

I. ELEMENT checks to see if a specified element is on a list. If it is ELEMENT takes the value true, otherwise false.

II. ELEMENT is a predicate having two arguments; (X L) where X is a specified element.
L is a list of any length which may or may not contain X.

III. Machine definition.

```
(ELEMENT (LAMBDA (X L)
(COND ((NULL L) F)
((EQUAL X (CAR L)) T)
(T (ELEMENT X (CDR L))))))
```

IV. Examples

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ELEMENT
((A,B),(C,(A,B),D))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ()]

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ELEMENT
(A,((A,B),C,D))


END OF APPLY, VALUE IS ...
               FALSE, SINCE THERE APPEARS NO ()

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ELEMENT
(A,(R,S,A,C,A,D))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ()]
```

(SUBSET S L)

I. SUBSET takes the value true if all the elements of the first argument are contained in the second element, false otherwise.

II. SUBSET is a predicate having two arguments; (S L) where S is a list.
L is a list.
(NOTE) If S is empty SUBSET will treat it as being contained in L and take the value true.
SUBSET uses the function ELEMENT.

III. Machine definitions.

```
(SUBSET (LAMBDA (S L)
(COND ((NULL S) T)
((ELEMENT (CAR S) L) (SUBSET (CDR S) L))
(T F))))
```

IV. Examples

```
SUBSET
( ,(A,N,Y,T,H,I,N,G))

( )

SUBSET
((A),(A,B,C))

( )

SUBSET
((A,B),(A,X,Y,Z, ,*))
```

(SUCHTHAT L P)

I. SUCHTHAT is a function which lists the first element in a given list for which a given condition is true.

II. SUCHTHAT is a function of two arguments: (L P)

L is a list of lists

P is a predicate. In examples for proper form to use in designating P.

If no element of L has property P, then there is no result.

III. Machine definition

```
(SUCHTHAT (LAMBDA (L P)
(COND ((NULL L) NIL)
((P (CAR L)) (CAR L))
(T (SUCHTHAT (CDR L) P)))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SUCHTHAT
(((A,B,C),(A,B,C)),(FUNCTION,(LAMBDA,(S),(ULTRATRIPLET,S))))


END OF APPLY, VALUE IS ...
(A,B,C)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SUCHTHAT
(((A,A),(A,B)),(FUNCTION,(LAMBDA,(S),(EQUAL,(CAR,S),(CADR,S)))))


END OF APPLY, VALUE IS ...
(A,A)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SUCHTHAT
(((A,B),(A,B)),(FUNCTION,(LAMBDA,(S),(ULTRATRIPLET,S))))


END OF APPLY, VALUE IS ...
```

(POSSESSING P S)

I. POSSESSING picks out all the elements of a given list with a specified property and puts them in a new list.

II. POSSESSING is a function of two arguments; (P S) where P is a specified property to be checked for.
S is a list of any length.
(NOTE) The value of POSSESSING is an empty list if applied to an empty list.

III. Machine definition.

```
(POSSESSING (LAMBDA (P S)
(COND ((NULL S) NIL)
((P (CAR S)) (CONS (CAR S) (POSSESSING P (CDR S))))
(T (POSSESSING P (CDR S)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POSSESSING
((FUNCTION,(LAMBDA,(S),(EQUAL,(CAR,S),(CADR,S)))),((A,A),(A,B)))


END OF APPLY, VALUE IS ...
((A,A))

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POSSESSING
((FUNCTICN,(LAMBDA,(S),(ULTRATRIPLET,S))),((A, ,B),(A, ,B),(A, ,B)))


END OF APPLY, VALUE IS ...
((A, ,B),(A, ,B),(A, ,B))

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POSSESSING
((FUNCTION,(LAMBDA,(S),(ULTRATRIPLET,S))),((A,B,C,D),(A,B)))


END OF APPLY, VALUE IS ...
((A,B,C,D))
```

(TRUTHVALUE P)

I. TRUTHVALUE is a function which tests whether another function is true or false, and according to the outcome prints TRUE or FALSE.

II. TRUTHVALUE is a function of one argument; (P) where P is the property to be tested.

III. Machine definition.

```
(TRUTHVALUE (LAMBDA (P) (COND (P (QUOTE TRUE)) (T (QUOTE FALSE)))))
```

IV. Examples

```
(LAMBDA,(X,L),(TRUTHVALUE,(PARITY,X,L)))
(V,((A,Q),(V,V), ))

FALSE
```

```
(LAMBDA,(S,L),(TRUTHVALUE,(SUBSET,S,L)))
((V,R,T),(A,( ,X),V,R,T,Z))

TRUE
```

```
(LAMBDA,(X,L),(TRUTHVALUE,(ELEMENT,X,L))).
(B,(A,(Z, ,(R,B,*),Q)))

FALSE
```

(SINGLET L)

I. SINGLET is a predicate which takes the value true if its argument is a list containing one element. Otherwise it takes the value false.

II. SINGLET is a function of one argument: (L)

If L is a list containing one element, SINGLET is true.

If L is nil, or atomic, or contains more than one element, SINGLET is false.

III. Machine definition.

```
(SINGLET (LAMBDA (L) (AND (ULTRASINGLET L) (NULL (CDR L)))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SINGLET
((A))


END OF APPLY, VALUE IS ...
( )
```

TRUE [AS INDICATED BY THE PRESENCE OF ()]

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SINGLET
((A,B))


END OF APPLY, VALUE IS ...
```

FALSE SINCE THERE APPEARED NO ()

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SINGLET
(A)


END OF APPLY, VALUE IS ...
```

FALSE, SINCE THERE APPEARED NO ()

(DOUBLET L)

I. DOUBLET takes a value <u>true</u> if the list has exactly two elements on it. A list on L will be regarded as a single element.

II. DOUBLET is a predicate having one argument; (L) where L is a list of any length.

III. Machine definition.

```
(DOUBLET (LAMBDA (L) (AND (ULTRADOUBLET L) (NULL (CDDR L)))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DOUBLET
((A,B))


END OF APPLY, VALUE IS ...
( )          TRUE [AS INDICATED BY THE PRESENCE OF ( )]


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DOUBLET
(((A,B),(C,D)))


END OF APPLY, VALUE IS ...
( )          TRUE [AS INDICATED BY THE PRESENCE OF ( )]


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DOUBLET
((A,B,C))


END OF APPLY, VALUE IS ...
               FALSE, SINCE THERE APPEARED NO ( )
```

(TRIPLET L)

I. TRIPLET takes a value true if there are exactly three elements on a list, otherwise its value is false.

II. TRIPLET is a predicate having one argument; (L) where L is a list of any length.
(NOTE) A list on L will be regarded as a single element.

III. Machine definition.

```
(TRIPLET (LAMBDA (L) (AND (ULTRATRIPLET L) (NULL (CDDDR L)))))
```

IV. Examples.

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 TRIPLET
 ((A,B,C))
```

```
 END OF APPLY, VALUE IS ...
 ( )         TRUE  [AS INDICATED BY THE PRESENCE OF ( ) ]
```

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 TRIPLET
 ((A,B))
```

```
 END OF APPLY, VALUE IS ...
             FALSE , SINCE THERE APPEARED NO ( )
```

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 TRIPLET
 ((A,(B,C),D))
```

```
 END OF APPLY, VALUE IS ...
 ( )         TRUE  [AS INDICATED BY THE PRESENCE OF ( ) ]
```

(ODDPLET L)

I. ODDPLET takes a value of true if there are an odd number of elements on a list and false otherwise.

II. ODDPLET is a predicate having one argument; (L) where L is a list of any length.

III. Machine definition.

```
(ODDPLET (LAMBDA (L) (NOT (OR (NULL L) (ODDPLET (CDR L))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ODDPLET
((A,(B,C),D))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ( )]

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ODDPLET
((A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,1,2,3))


END OF APPLY, VALUE IS ...
( )          TRUE [AS INDICATED BY THE PRESENCE OF ( )]

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ODDPLET
((A,B))


END OF APPLY, VALUE IS ...
             FALSE SINCE THERE APPEARED NO ( )
```

(ULTRASINGLET L)

I. **ULTRASINGLET is a predicate which takes the value true if a given list contains at least one element.**

II. **ULTRASINGLET is a predicate of one argument, L.**
L is a list. If L is null or an atomic symbol, ULTRASINGLET takes the value false.

III. **Machine definition of ULTRASINGLET.**

```
(ULTRASINGLET (LAMBDA (L) (NOT (OR (ATOM L) (NULL L)))))
```

IV. **Examples.**

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRASINGLET
((A))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ()]

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRASINGLET
(A)


END OF APPLY, VALUE IS ...
           FALSE, SINCE THERE APPEARED NO ()

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRASINGLET
((A,B))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ()]

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRASINGLET
((A,(BC,D)))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ()]
```

(ULTRADOUBLET L)

I. ULTRADOUBLET takes a value of <u>true</u> if a list has two or more items. A list contained in L is counted as a single element.

II. ULTRADOUBLET is a predicate of one argument; (L) where L is a length of any length.

III. Machine definition.

```
(ULTRADOUBLET (LAMBDA (L) (NOT (OR (ATOM L) (NULL L) (NULL (CDR L))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRADOUBLET
((A,B,C,D))


END OF APPLY, VALUE IS ...
( )          TRUE [AS INDICATED BY THE PRESENCE OF ()]


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRADOUBLET
(((A,B),(C,D)))


END OF APPLY, VALUE IS ...
( )          TRUE [AS INDICATED BY THE PRESENCE OF ()]


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRADOUBLET
(((A,B)))


END OF APPLY, VALUE IS ...
             FALSE  SINCE THERE APPEARED NO ()
```

(ULTRATRIPLET L)

I. ULTRATRIPLET takes a value of <u>true</u> if a list contains more than two elements, and a value of <u>false</u> otherwise. A list contained in L will be treated as a single element.

II. ULTRATRIPLET is a predicate having one argument; (L) where L is a list of any length.

III. Machine definition

```
(ULTRATRIPLET (LAMBDA (L) (AND (ULTRADOUBLET L) (NOT (NULL (CDDR L))))))
```

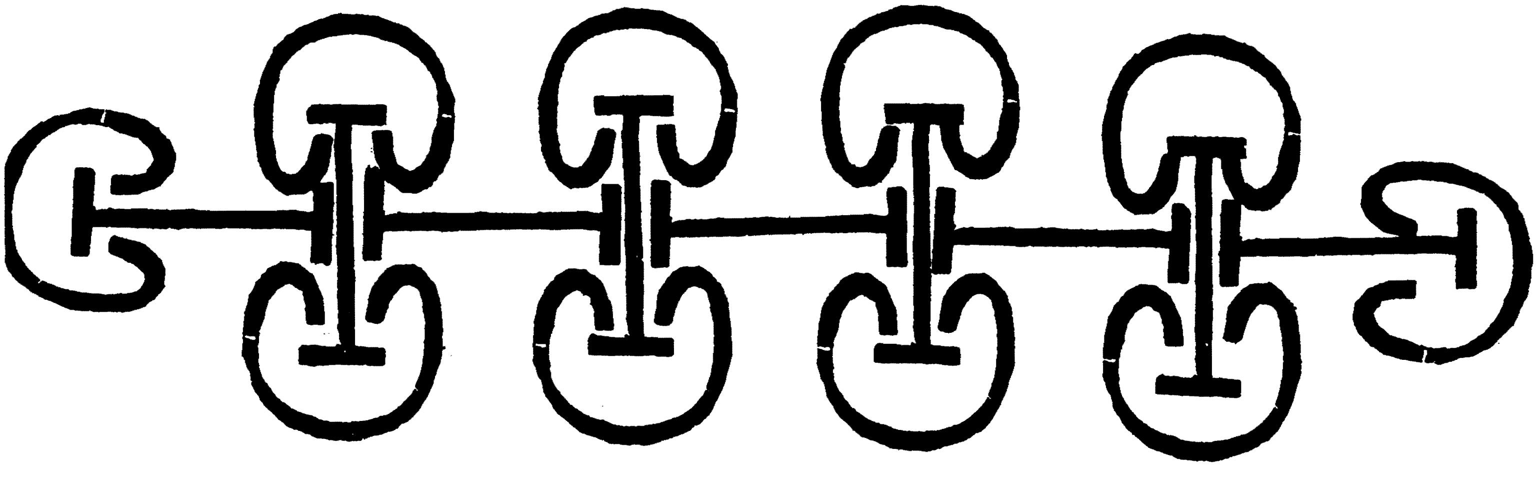
IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRATRIPLET
((A,B,C,D))


END OF APPLY, VALUE IS ...
( )        TRUE [AS IS INDICATED BY THE PRESENCE OF ()]
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRATRIPLET
((A,B))


END OF APPLY, VALUE IS ...
           FALSE, SINCE THERE APPEARED NO ()
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ULTRATRIPLET
((A,(B,C),D))


END OF APPLY, VALUE IS ...
( )        TRUE [AS INDICATED BY THE PRESENCE OF ()]
```

Miscellaneous functions, chosen for convenience or elegance



(INTERSECTION L)

I. INTERSECTION makes a list of all elements common to all the given lists.

II. INTERSECTION is a predicate of one argument; (L) where L is a list of any number of lists.

III. Machine definition.

```
(INTERSECTION (LAMBDA (L)
(COND ((NULL L) (QUOTE UNIVERSAL))
(T (POSSESSING (FUNCTION (LAMBDA (U) (FORALL L (FUNCTION (LAMBDA (V)
(ELEMENT U V)))))) (CAR L))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INTERSECTION
(((A,B,C,A,D),(A,B,A,E)))


END OF APPLY, VALUE IS ...
(A,B,A)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INTERSECTION
( )


END OF APPLY, VALUE IS ...
UNIVERSAL


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INTERSECTION
(( ,(A,B,C)))


END OF APPLY, VALUE IS ...
```

(UNION L)

I. UNION makes a list of any items which are on all the given lists.

II. UNION is a predicate of one argument; (L) where L is a list made up of any number of lists.

III. Machine definition

```
(UNION (LAMBDA (L)
(COND ((NULL L) NIL)
((NULL (CDR L)) (CAR L))
(T (UNION (CONS (ACCUMULATE (CADR L) (CAR L)) (CDDR L))))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
UNION
(((A,C,T,X,U,B,H),(T,E,C,B,X,Z,H,R)))


END OF APPLY, VALUE IS ...
(R,Z,E,A,C,T,X,U,B,H)
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
UNION
(((A,B,C,D,E,F,(R,S)),(Y,A,C,G,B,R,S)))


END OF APPLY, VALUE IS ...
(S,R,G,Y,A,B,C,D,E,F,(R,S))
```

(DELTA L)

I. DELTA makes a list of all items which are on an odd number of lists.

II. DELTA is a predicate of one argument; (L) where L is a list made up of any number of lists.

III. Machine definition.

```
(DELTA (LAMBDA (L) (POSSESSING (FUNCTION (LAMBDA (U)
(FORODD L (FUNCTION (LAMBDA (V) (ELEMENT U V))))))
(UNION L))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DELTA
(((A,B,A,C,D),(B,C,D,E,F,G)))


END OF APPLY, VALUE IS ...
(G,F,E,A,A)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DELTA
(((A,B,A,C,D),(A,B,C,D,E,D)))


END OF APPLY, VALUE IS ...
(E)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DELTA
( )


END OF APPLY, VALUE IS ...
```

(COMPOSITE A B S)

I. COMPOSITE checks to see if there exist doublets (X Y) contained in list A and (Y Z) contained in list B, Y being an element of list S. If there exist such doublets COMPOSITE forms a new list of doublets (X Z) made up of the first element of the doublet contained in B.

II. COMPOSITE is a function of three arguments; (A B S) where
A is a list of doublets
B is a list of doublets
S is a list of atomic symbols

III. Machine definition.

```
(COMPOSITE (LAMBDA (A B S)
(POSSESSING (FUNCTION (LAMBDA (U) (FORSOME S (FUNCTION (LAMBDA (V)
(AND (ELEMENT (LIST (CAR U) V) A) (ELEMENT
(LIST V (CADR U)) B))))))) (KARTESIAN S S))))
```

1V. Examples

```
COMPOSITE
((((1,2),(3,4),(5,6),(7,8)),((9,9),(8,8),(7,7),(6,6))
,(1,2,3,4,5,6,7,8,9))


((5,6),(7,8))
```

```
COMPOSITE
(((A,B),(C,D),(E,F),(G,H)),((H,H),(G,G),(F,F)
,(E,E),(D,D),(C,C),(B,B),(A,A)),(A,B,C,D,E,F,G,H))

((A,B),(C,D),(E,F),(G,H))
```

(COMPOSE L S)

I. COMPOSE forms the composite of all the elements of a list with a second list.

II. COMPOSE is a function of two arguments; (L S) where
L is a list of sublists of doublet sub-sub lists.
S is a list of atoms.
(NOTE) If L is an empty list the value of compose will be an empty list; if a singlet list, the value will be that singlet list.
This function uses the function COMPOSITE.

III. Machine definition.

```
(COMPOSE (LAMBDA (L S)
(COND ((NULL L) NIL)
((NULL (CDR L)) L)
(T (COMPOSE (CONS (COMPOSITE (CAR L) (CADR L) S) (CDDR L)) S)))))
```

(KARTESIAN X Y)

I. KARTESIAN forms all the possible pairs of the elements contained in two lists.

II. KARTESIAN is a function of two arguments; (X Y) where
X is a list of any length.
Y is a list of any length.

III. Machine definition.

```
(KARTESIAN (LAMBDA (X Y)
(MAPCON X (FUNCTION (LAMBDA (U) (MAPLIST Y (FUNCTION (LAMBDA (V)
(LIST (CAR U) (CAR V))))))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
KARTESIAN
((1,1,1),(*,*,*,*,*))


END OF APPLY, VALUE IS ...
 ((1,*),(1,*),(1,*),(1,*),(1,*),(1,*),(1,*),(1,*),(1,*),(1,*),(1,*),
(1,*),(1,*),(1,*),(1,*))

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
KARTESIAN
((A, ,B),(1, ,2, ,3))


END OF APPLY, VALUE IS ...
 ((A,1),(A, ),(A,2),(A, ),(A,3),( ,1),( , ),( ,2),( , ),( ,3),(B,1),
(B, ),(B,2),(B, ),(B,3))

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
KARTESIAN
((1,2,3),(A,B,C,D,E))


END OF APPLY, VALUE IS ...
 ((1,A),(1,B),(1,C),(1,D),(1,E),(2,A),(2,B),(2,C),(2,D),(2,E),(3,A),
(3,B),(3,C),(3,D),(3,E))
```

(KARTESIAN* X Y)

I. KARTESIAN*, when applied to a list, will create a new list, in which each of the elements of a second list be appended, in turn, to each of the elements of the first list. This function would be used to add another element (or set of elements) to each of the pairs created by KARTESIAN.

II. KARTESIAN* is a function of two arguments; (X Y) where
X is a list of any length.
Y is a list of any length.

III. Machine definition.

```
(KARTESIAN* (LAMBDA (X Y)
(MAPCON X (FUNCTION (LAMBDA (U) (MAPLIST Y (FUNCTION (LAMBDA (V)
(APPEND (CAR U) (LIST (CAR V))))))))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 KARTESIAN*
 (((1,2),(3,4),(5,6)),((7,8),(9,10),(11,12)))


 END OF APPLY, VALUE IS ...
 ((1,2,(7,8)),(1,2,(9,10)),(1,2,(11,12)),(3,4,(7,8)),(3,4,(9,10)),
 (3,4,(11,12)),(5,6,(7,8)),(5,6,(9,10)),(5,6,(11,12)))
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 KARTESIAN*
 (((1,2),(3,4),(5,1),(3,3)),(*))


 END OF APPLY, VALUE IS ...
 ((1,2,*),(3,4,*),(5,1,*),(3,3,*))
```

(CARTESIAN* L)

I. CARTESIAN* takes a list of any number of sublists, which in turn may contain sublists, and creates a new list of all possible ordered combinations of sublists where the first element of a resulting sublist is from the first sublist given, the second element of the resulting sublist is from the second sublist given, and so on. The resulting sub-lists contain partially nested sub-sub lists.

II. CARTESIAN* is a function of only one argument; (L) where L is a list divided into sublists which may be divided still further.
(NOTE) If there are no sub-sub lists, the new list is created only of the elements of the first sublist.

III. Machine definition.

```
(CARTESIAN* (LAMBDA (L)
(COND ((NULL L) NIL)
((NULL (CDR L)) L)
(T (CARTESIAN* (CONS (KARTESIAN* (CAR L)(CADR L)) (CDDR L)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
CARTESIAN*
((((A,B),(C,D),(E,F)),((G,H),(I,K),(K,L)),(Q,M,P,V,Z)))


END OF APPLY, VALUE IS ...
(((A,B,(G,H),Q),(A,B,(G,H),M),(A,B,(G,H),P),(A,B,(G,H),V),(A,B,(G,H),Z),
(I,K),V),(A,B,(I,K),Z),(A,B,(K,L),Q),(A,B,(K,L),M),(A,B,(K,L),P),(A,B,(K,L.
),(C,D,(G,H),P),(C,D,(G,H),V),(C,D,(G,H),Z),(C,D,(I,K),Q),(C,D,(I,K),M),(
(K,L),Q),(C,D,(K,L),M),(C,D,(K,L),P),(C,D,(K,L),V),(C,D,(K,L),Z),(E,F.
),(E,F,(G,H),Z),(E,F,(I,K),Q),(E,F,(I,K),M),(E,F,(I,K),P),(E,F,(I,K),V),.
(K,L),P),(E,F,(K,L),V),(E,F,(K,L),Z)))
```

(CARTESIAN L)

I. CARTESIAN takes a list of any number of sublists, which in turn may contain sublists and creates a new list of all possible ordered combinations of sublists where the first element of a resulting sublist is from the first argument sublist, the second from the second argument sublist, and so on. The resulting sublists contain appended sub-sub lists, which differentiates this function from CARTESIAN*.

II. CARTESIAN is a function of only one argument; (L) where L is a list which may be divided into sub-lists which may be divided still further.
CARTESIAN uses the function KARTESIAN and CARTESIAN*.

III. Machine definition.

```
(CARTESIAN (LAMBDA (L)
(COND ((NULL L) NIL)
((NULL (CDR L)) L)
(T (CARTESIAN* (CONS (KARTESIAN (CAR L) (CADR L)) (CDDR L))))))))
```

IV. Examples.

```
CARTESIAN
(((1,2),(3,4),(5,6),(7,8)))


(((1,3,5,7),(1,3,5,8),(1,3,6,7),(1,3,6,8),(1,4,5,7),(1,4,5,8)
,(1,4,6,7),(1,4,6,8),(2,3,5,7),(2,3,5,8),(2,3,6,7),(2,3,6,8)
,(2,4,5,7),(2,4,5,8),(2,4,6,7),(2,4,6,8)))


CARTESIAN
(((A,B),(C,D)))


(((A,C),(A,D),(B,C),(B,D)))
```

(AMONG X L)

I. AMONG is a function that checks a list for a specified element. If it does not find the element on the list it adds it on at the end, creating a new list consisting of the old list plus the new element.

II. AMONG is a function of two arguments; (X L) where
X may be an element or a list.
L is a list of any length.
(NOTE) If the element X is present on L the function takes the value of X.
AMONG uses the function ELEMENT.

III. Machine definition.

```
(AMONG (LAMBDA (X L)
(COND ((ELEMENT X L) L)
(T (CONS X L)))))
```

IV. Examples

```
AMONG
(ATOM,(M,O,L,E,C,U,L,E))

(ATOM,M,O,L,E,C,U,L,E)

AMONG
(X,(X,Y,Z))

(X,Y,Z)

AMONG
((A,B,C),(D,E,F))

((A,B,C),D,E,F)

AMONG
(X, )

(X)
```

(REMOVE I L)

I. REMOVE takes the first occurrence of a specific item off a given list. If I is an element of a list contained in L it will not be removed. If I is not an element on L, L remains unchanged.

II. REMOVE is a function of two arguments; (I L) where I is an item, the first occurrence of which will be removed from L.

L is a list of any length.

III. Machine definition.

```
(REMOVE (LAMBDA (I L)
(COND ((NULL L) NIL)
((EQUAL I (CAR L)) (CDR L))
(T (CONS (CAR L) (REMOVE I (CDR L))))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REMOVE
(X,(A,B,S,(C,D),E,F,D,G,H))


END OF APPLY, VALUE IS ...
(A,B,S,(C,D),E,F,D,G,H)
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REMOVE
(D,(A,B,(C,D),E,F,D,G,H))


END OF APPLY, VALUE IS ...
(A,B,(C,D),E,F,G,H)
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REMOVE
( ,(A,B, ,C,D))


END OF APPLY, VALUE IS ...
(A,B,C,D)
```

(EXPUNGE I L)

I. EXPUNGE will remove every occurrence of a given element from a given list.

II. EXPUNGE is a function of two arguments; (I L) where I is an element which may or may not be included on the list L.
L is a list of any length.
(NOTE) EXPUNGE will not remove an occurrence of I from a list contained in L.

III. Machine definition.

```
(EXPUNGE (LAMBDA (I L) (POSSESSING (FUNCTION (LAMBDA (U)
(NOT (EQUAL I U)))) L)))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
EXPUNGE
(A,(A,B,A,C,A,D))


END OF APPLY, VALUE IS ...
(B,C,D)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
EXPUNGE
(A,((A,B),A,C,A,D))


END OF APPLY, VALUE IS ...
((A,B),C,D)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
EXPUNGE
((A,B),((A,B),A,C,(A,B)))


END OF APPLY, VALUE IS ...
(A,C)
```

(COLLECT L)

I. COLLECT regroups a list so that any reoccurrence of an element is made immediately after its first occurrence.

II. COLLECT is a function of one argument; (L) where L is a list of any length which may or may not have repeated elements.

III. Machine definition.

```
(COLLECT (LAMBDA (L)
(COND ((NULL L) NIL)
((ELEMENT (CAR L) (CDR L)) (CONS (CAR L) (COLLECT
(CONS (CAR L) (REMOVE (CAR L) (CDR L))))))
(T (CONS (CAR L) (COLLECT (CDR L)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
COLLECT
((A,B,D,A,A,C,B,E,C,B))


END OF APPLY, VALUE IS ...
(A,A,A,B,B,B,D,C,C,E)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
COLLECT
((A,B,(A,B),D,A,C,B))


END OF APPLY, VALUE IS ...
(A,A,B,B,(A,B),D,C)
```

(ACCUMULATE L M)

I. ACCUMULATE will make a list of distinct items from two lists, providing one of the original lists is itself a list of distinct items.

II. ACCUMULATE is a function of two arguments; (L M) where L is a list which may contain any number of distinct or non-distinct elements.
M is a list of any number of distinct elements.
(NOTE) Any element on L which is not on M is placed on M.

III. Machine definition.

```
(ACCUMULATE (LAMBDA (L M)
(COND ((NULL L) M)
((ELEMENT (CAR L) M) (ACCUMULATE (CDR L) M))
(T (ACCUMULATE (EXPUNGE (CAR L) (CDR L)) (CONS (CAR L) M))))))
```

IV. Examples

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 ACCUMULATE
 ((A,B,C,D),(D,E,F,A,D))


 END OF APPLY, VALUE IS ...
 (C,B,D,E,F,A,D)


  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 ACCUMULATE
 ((A,B,C,D,B,C),(D,E,F,A))


 END OF APPLY, VALUE IS ...
 (C,B,D,E,F,A)
```

(ORDER X Y L)

I. ORDER takes a value of true if X is on the list and Y is not or if X and Y are both on the list and X comes before Y. It takes a value of false if X is not on the list, or if Y comes before X.

II. ORDER is a predicate having three arguments; (X Y L) where X is an element which may or may not be on the list. Y is an element which may or may not be on the list. L is a list of any length, in general ordered from smallest to largest. However, this is merely a convention and is not necessary.

III. Machine definition.

```
(ORDER (LAMBDA (X Y L)
(COND ((NULL L) F)
((EQUAL X (CAR L)) T)
((EQUAL Y (CAR L)) F)
(T (ORDER X Y (CDR L))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ORDER
(A,B,(M,P,A,S,T,B,R))


END OF APPLY, VALUE IS ...
( )             TRUE [AS INDICATED BY THE PRESENCE OF ()]


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ORDER
(A,B,(M,P,B,S,A,T))


END OF APPLY, VALUE IS ...
          FALSE, SINCE THERE APPEARED NO ()
```

(LEXORDER X Y L)

I. LEXORDER is a predicate which takes the value true if, for the first pair of non-equal corresponding elements in two given lists, the element of the first list precedes the corresponding element of the second list in the corresponding element of a third given list. Otherwise it takes the value false.

LEXORDER utilizes the lisp predicate ORDER.

II. LEXORDER is a predicate of three arguments: (X Y L)

X is a list

Y is a list

L is a list of lists.

If X or Y or L runs out of elements before a value true has been given, the predicate takes the value false.

III. Machine definition of predicate LEXORDER

```
(LEXORDER (LAMBDA (X Y L)
(COND ((NULL L) F)
((NOT (EQUAL (CAR X) (CAR Y))) (ORDER (CAR X) (CAR Y) (CAR L)))
(T (LEXORDER (CDR X) (CDR Y) (CDR L))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
LEXORDER
(( ,A),( ,B),((A, ),(A,B),(C,D,Z),Q,P))


END OF APPLY, VALUE IS ...
( )         TRUE  [AS INDICATED BY THE PRESENCE OF ()]
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
LEXORDER
(((A,B),C,D,E),((A,B),C,D,P),((A,B),C,D,E))


END OF APPLY, VALUE IS ...
          FALSE  SINCE THERE APPEARED NO ()
```

(MINIMUM L O)

I. MINIMUM selects the smallest element on a given list according to a second list.

II. MINIMUM is a predicate of two arguments; (L O) where L is a list from which the smallest item is to be chosen O is an ordered list of any length of the form:
$(X_1, X_2, X_3, \ldots X_n)$ such that $X_1 \leq X_2 \leq X_3 \leq \ldots X_n$.
(NOTE) If no element of L is on the list O, MINIMUM takes as its value the last element of L.

III. Machine definition

```
(MINIMUM (LAMBDA (L O)
(COND ((NULL L) NIL)  ((NULL (CDR L)) L)
((ORDER (CAR L) (CADR L) O) (MINIMUM (CONS (CAR L) (CDDR L)) O))
(T (MINIMUM (CDR L) O)))))
```

(RANK L O)

I. RANK rewrites a list so that its elements are ordered from smallest to largest, according to a second list.

II. RANK is a function of two arguments; (L O) where L is a list of any length and not necessarily ordered. O is an ordered list of the form $(X_1, X_2, X_3, \ldots X_n)$ such that $X_1 < X_2 < X_3 < \ldots X_n$.

(NOTE) List O need not contain all elements of L. If it does not, those elements of L which are not listed on O are considered larger.

III. Machine definition.

```
(RANK (LAMBDA (L O) (COND ((NULL L) NIL) (T (CONS (MINIMUM L O)
(RANK (REMOVE (MINIMUM L O) (CDR L)) O))))))
```

(EQUIVALENT X Y L)

I. EQUIVALENT takes the value <u>true</u>, if two specified elements are contained in a sublist of a given list, <u>false</u> otherwise.

II. EQUIVALENT is a predicate of three arguments; (X Y L) where
X is an element
Y is an element
L is a list which may have sublists.
(NOTE) EQUIVALENT uses the function ELEMENT.

III. Machine definition.

```
(EQUIVALENT (LAMBDA (X Y L)
(COND ((NULL L) F)
((ELEMENT X (CAR L)) (ELEMENT Y (CAR L)))
(T (EQUIVALENT X Y (CDR L))))))
```

IV. Examples

```
EQUIVALENT .
(1,2,(3,4,(1,2,5),  ,6))

( )

EQUIVALENT
( ,  ,(( ,  ),A,B,Q))

( )

EQUIVALENT
((X),(Y),(A,X,Y,C))


EQUIVALENT
(X,Y,(X,A,B,C))
```

I. TRANSPOSE takes a list in the form as is produced by REASSOCIATE and makes a list whose elements are in the inverse order of the elements of the given list.

II. TRANSPOSE is a function of one argument: (L)

L is a list of the form REASSOCIATE for which the CDR of each CAR is atomic. Otherwise an error results.

III. Machine definition

```
(TRANSPOSE (LAMBDA (L)
(COND ((NULL (CDR L)) L)
(T (CONS (CADR L) (TRANSPOSE (CAR L)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
TRANSPOSE
(((((A),B),C),D))


END OF APPLY, VALUE IS ...
(D,C,B,A)
```

(PREPARE L)

I. PREPARE makes the first element of a given list a singlet sublist of that list.

II. PREPARE is a function of only one argument; (L) where L is a list of any length, which may or may not contain other lists.
(NOTE) If the given list is an atom, empty or a singlet its value will be an empty list.

III. Machine definition.

```
(PREPARE (LAMBDA (L)
(COND ((OR (ATOM L) (NULL L) (NULL (CDR L))) NIL)
(T (CONS (CONS (CAR L) NIL) (CDR L))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
PREPARE
((A, ,B))


END OF APPLY, VALUE IS ...
((A), ,B)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
PREPARE
(((A,B),C,D,E))


END OF APPLY, VALUE IS ...
(((A,B)),C,D,E)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
PREPARE
(( ,*))


END OF APPLY, VALUE IS ...
(( ),*)
```

(REASSOCIATE L)

I. REASSOCIATE is a function which makes a list having two elements by consecutively grouping the elements of a given list into lists.

II. REASSOCIATE is a function of one argument: (L)
L is a list. If L contains two or less elements, the result is L. Otherwise see examples.

III. Machine definition

```
(REASSOCIATE (LAMBDA (L)
(COND ((NULL (CDDR L)) L)
(T (REASSOCIATE (CONS (LIST (CAR L) (CADR L)) (CDDR L)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REASSOCIATE
(((A,B),C,D,E))


END OF APPLY, VALUE IS ...
((((A,B),C),D),E)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REASSOCIATE
((*,A,B))


END OF APPLY, VALUE IS ...
((*,A),B)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REASSOCIATE
(((A,Z),(P,Q)))


END OF APPLY, VALUE IS ...
((A,Z),(P,Q))
```

(INVERT L)

I. INVERT is a function which rearranges a given list to the reverse order.

INVERT uses functions PREPARE, REASSOCIATE, TRANSPOSE, and ELEMENT.

II. INVERT is a function of one argument: (L)

L is a list.

III. Machine definition.

```
(INVERT (LAMBDA (L)
(COND ((NULL (PREPARE L)) L)
(T (TRANSPOSE (REASSOCIATE (PREPARE L)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INVERT
(( ,*))


END OF APPLY, VALUE IS ...
(*, )


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INVERT
((A, ,B))


END OF APPLY, VALUE IS ...
(B, ,A)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INVERT
(((A,B),C,D,E))


END OF APPLY, VALUE IS ...
(E,D,C,(A,B))
```

(UNIFORM L M)

I. UNIFORM takes the value <u>true</u> if all the even terms of a list are the same as a given term. The list must have an odd number of elements and at least three. Otherwise it takes the value <u>false</u>.

UNIFORM uses the predicates TRIPLET and ULTRATRIPLET.

II. UNIFORM is a predicate of two arguments; (LM) where

L is an odd list
M is the given term to be tested against the list.

III. Machine definition.

```
(UNIFORM (LAMBDA (L M) (OR (AND (TRIPLET L) (EQUAL (CADR L) M))
(AND (ULTRATRIPLET L) (EQUAL (CADR L) M) (UNIFORM (CDDR L) M)))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
UNIFORM
((A,-,B,-,C,-,D,-,E,-,F),-)


END OF APPLY, VALUE IS ...
( )          TRUE (INDICATED BY THE PRESENCE OF THE ())
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
UNIFORM
((A,-,B,*,C,-,D,*,E,-,F),-)


END OF APPLY, VALUE IS ...
           FALSE  SINCE THERE APPEARED NO ( )
```

(ALTERNATE L)

I. ALTERNATE makes a list consisting of all the odd terms of the given list.

II. ALTERNATE is a predicate of one argument; (L) where L is a list of any length.

III. Machine definition.

```
(ALTERNATE (LAMBDA (L)
(COND ((NULL L) NIL)
((NULL (CDR L)) NIL)
(T (CONS (CAR L) (ALTERNATE (CDDR L)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ALTERNATE
((A,B,C,D,E,F,G,H))


END OF APPLY, VALUE IS ...
(A,C,E,G)



 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ALTERNATE
((A,B,C,D,E,F,G))


END OF APPLY, VALUE IS ...
(A,C,E,G)
```

(REALTERNATE L M)

I. REALTERNATE places one atom (element) or one list after each of the members of another list.

II. REALTERNATE is a function of two arguments; (L M) where L is a list with any member of elements M may be either an element or a list. (NOTE) If L is a list of only one element the new list will contain only that element. If L is an atom the function will not be carried out.

III. Machine definition.

```
(REALTERNATE (LAMBDA (L M)
(COND ((NULL L) NIL) ((NULL (CDR L)) L)
(T (CONS (CAR L) (CONS M (REALTERNATE (CDR L) M)))))))
```

IV. Examples

```
REALTERNATE
((R,I,A,S),M)

(R,M,I,M,A,M,S)


REALTERNATE
((R,I,A,S),(I,B,M))

(R,(I,B,M),I,(I,B,M),A,(I,B,M),S)
```

(ENGLISH L)

I. ENGLISH is a function which rearranges a list so the first element of the original list will appear, in a new list, following each of the other elements of the list.

II. ENGLISH is a function of one argument; (L) where L is a list of any length, the first member of which is to appear after each of the other members.
(NOTE) ENGLISH will not work if applied to a list containing less than two elements.

ENGLISH uses the function REALTERNATE.

III. Machine definition.

```
(ENGLISH (LAMBDA (L) (REALTERNATE (CDR L) (CAR L))))
```

IV. Examples

```
ENGLISH
((A,B,N,N,S))

(B,A,N,A,N,A,S)
```

(POLISH L)

I. POLISH takes an algebraic expression in the ordinary English notation with only one operation, and rewrites it in POLISH notation (with the operation first, followed by its operands).

II. POLISH is a function of only one argument; (L) where L is a list of any length (at least three terms with an odd number of terms. If this holds true, and all the even terms are the same, POLISH places one of the even terms at the front and lists the odd terms following. If the requirements are not met POLISH just lists L.

III. Machine definition.

```
(POLISH (LAMBDA (L)
(COND ((AND (ULTRATRIPLET L) (ODDPLET L) (UNIFORM L (CADR L)))
(CONS (CADR L) (ALTERNATE L)))   (T L))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POLISH
((A,-,B,-,C,-,D,-,E))


END OF APPLY, VALUE IS ...
(-,A,B,C,D,E)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POLISH
((A,-,B,*,C,-,D,-,E))


END OF APPLY, VALUE IS ...
(A,-,B,*,C,-,D,-,E)
```

(POSTPOLISH L)

I. POSTPOLISH will change an algebraic expression in english notation containing only one operation to postpolish form, with the operation following all its operands.

II. POSTPOLISH is a predicate of one argument; (L) where L is a list of any length having an odd number of terms with the even terms the same.
(NOTE) If the list L is even, or if the even terms are not identical, POSTPOLISH just lists L.

III. Machine definition.

```
(POSTPOLISH (LAMBDA (L)
(COND ((AND (ULTRATRIPLET L) (ODDPLET L) (UNIFORM L (CADR L)))
(APPEND (ALTERNATE L) (LIST (CADR L))))     (T L))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POSTPOLISH
((A,-,B,-,C,-,D,-,E))


END OF APPLY, VALUE IS ...
(A,B,C,D,E,-)
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
POSTPOLISH
((A,-,B,*,C,-,D,-,E))


END OF APPLY, VALUE IS ...
(A,-,B,*,C,-,D,-,E)
```

(HOLLERITH)

I. HOLLERITH is a function which lists the hollerith characters available in lisp.

II. HOLLERITH is a function of <u>no</u> arguments: ( )

III. Machine definition

```
 (HOLLERITH (LAMBDA () (LIST
(QUOTE 0) (QUOTE 1) (QUOTE 2) (QUOTE 3) (QUOTE 4) (QUOTE 5) (QUOTE 6)
(QUOTE 7) (QUOTE 8) (QUOTE 9) (QUOTE A) (QUOTE B) (QUOTE C) (QUOTE D)
(QUOTE E) (QUOTE F) (QUOTE G) (QUOTE H) (QUOTE I) (QUOTE J) (QUOTE K)
(QUOTE L) (QUOTE M) (QUOTE N) (QUOTE O) (QUOTE P) (QUOTE Q) (QUOTE R)
(QUOTE S) (QUOTE T) (QUOTE U) (QUOTE V) (QUOTE W) (QUOTE X) (QUOTE Y)
(QUOTE Z) (QUOTE -) (QUOTE () (QUOTE *) (QUOTE )) (QUOTE =) (QUOTE ,)
(QUOTE $) (QUOTE .) (QUOTE -) (QUOTE +) (QUOTE ())))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 HOLLERITH



 END OF APPLY, VALUE IS ...
 (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,
U,V,W,X,Y,Z,-, ,=,( ),$,.,-,+, )
```

(INTEGERS)

I. **INTEGERS is a function which lists the decimal numbers, from 0 to 60**

II. **INTEGERS is a function of <u>no</u> arguments**

III. **Machine definition**

```
(INTEGERS (LAMBDA () (LIST
(QUOTE 0) (QUOTE 1) (QUOTE 2) (QUOTE 3) (QUOTE 4) (QUOTE 5) (QUOTE 6)
(QUOTE 7) (QUOTE 8) (QUOTE 9) (QUOTE 10) (QUOTE 11) (QUOTE 12)
(QUOTE 13) (QUOTE 14) (QUOTE 15) (QUOTE 16) (QUOTE 17) (QUOTE 18)
(QUOTE 19) (QUOTE 20) (QUOTE 21) (QUOTE 22) (QUOTE 23) (QUOTE 24)
(QUOTE 25) (QUOTE 26) (QUOTE 27) (QUOTE 28) (QUOTE 29) (QUOTE 30)
(QUOTE 31) (QUOTE 32) (QUOTE 33) (QUOTE 34) (QUOTE 35) (QUOTE 36)
(QUOTE 37) (QUOTE 38) (QUOTE 39) (QUOTE 40) (QUOTE 41) (QUOTE 42)
(QUOTE 43) (QUOTE 44) (QUOTE 45) (QUOTE 46) (QUOTE 47) (QUOTE 48)
(QUOTE 49) (QUOTE 50) (QUOTE 51) (QUOTE 52) (QUOTE 53) (QUOTE 54)
(QUOTE 55) (QUOTE 56) (QUOTE 57) (QUOTE 58) (QUOTE 59) (QUOTE 60))))
```

IV. **Examples**

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INTEGERS


 END OF APPLY, VALUE IS ...
 (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,
41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60)
```

(BINARY L)

I. BINARY is a function which changes an octal number (that is, a number to the base 8) to a binary number.

II. BINARY is a function of one argument: L is a list of octal digits; that is, L is an octal number in list form.

Notice that each octal digit takes up three binary digits. This is, of course, because 8 is the cube.

III. Machine definition of BINARY.

```
(BINARY (LAMBDA (L)
(COND ((NULL L) NIL)
((EQUAL (CAR L) (QUOTE 7)) (CONS (QUOTE 1) (CONS (QUOTE 1)
(CONS (QUOTE 1) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 6)) (CONS (QUOTE 1) (CONS (QUOTE 1)
(CONS (QUOTE 0) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 5)) (CONS (QUOTE 1) (CONS (QUOTE 0)
(CONS (QUOTE 1) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 4)) (CONS (QUOTE 1) (CONS (QUOTE 0)
(CONS (QUOTE 0) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 3)) (CONS (QUOTE 0) (CONS (QUOTE 1)
(CONS (QUOTE 1) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 2)) (CONS (QUOTE 0) (CONS (QUOTE 1)
(CONS (QUOTE 0) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 1)) (CONS (QUOTE 0) (CONS (QUOTE 0)
(CONS (QUOTE 1) (BINARY (CDR L))))))
((EQUAL (CAR L) (QUOTE 0)) (CONS (QUOTE 0) (CONS (QUOTE 0)
(CONS (QUOTE 0) (BINARY (CDR L))))))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 BINARY
 ((1))


 END OF APPLY, VALUE IS ...
 (0,0,1)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 BINARY
 ((1,3,7,2))


 END OF APPLY, VALUE IS ...
 (0,0,1,0,1,1,1,1,1,0,1,0)
```

(TALLY L M)

I. TALLY is a function which "counts" a binary number, by forming a list which contains the same number of NILs as are in the given binary number.

II. TALLY is a function of two arguments: (L M)

A. L is a list whose elements are the digits of a binary number. In other words, L is a binary number in list form.

B. M is a list. It may contain only an atomic symbol and NIL for simplicity. However, if M is an atomic symbol, wallpaper will be the result of TALLY. (See examples)

M is interspaced between the nils of the resulting list of TALLY. (See examples)

III. Machine definition of TALLY.

```
(TALLY (LAMBDA (L M)
(COND ((NULL L) M)
((EQUAL (CAR L) (QUOTE 0)) (TALLY (CDR L) (APPEND M M)))
((EQUAL (CAR L) (QUOTE 1)) (TALLY (CDR L)
(CONS NIL (APPEND M M)))))))
```

IV. Examples.

(TALLYCOPY L M)

I. TALLYCOPY is a function which lists as many elements from a given list, starting at the left, as there are elements in a second given list.

II. TALLYCOPY is a function of two arguments: (L M)

M is a list, and

L is a list, where list M contains at least as many elements as list L.

If list L contains more elements than M, TALLYCOPY will print out list M and then add NIL's until the new list contains as many elements as are in list L.

If either M or L is an atomic symbol, TALLYCOPY will take the value of M.

III. Machine definition of TALLYCOPY.

```
(TALLYCOPY (LAMBDA (L M)
(COND ((NULL L) NIL)
(T (CONS (CAR M) (TALLYCOPY (CDR L) (CDR M)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 TALLYCOPY
 ((((A,B,C),(C,E,P)),G,H,I),(A,B))


 END OF APPLY, VALUE IS ...
 (A,B,( ), )

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
TALLYCOPY
(A,((B,C),D,E))


END OF APPLY, VALUE IS ...
((B,C),D,E)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
TALLYCOPY
((A,(B,C),E),Q)


END OF APPLY, VALUE IS ...
Q
```

(TALLYCOMPLEMENT L M)

I. TALLYCOMPLEMENT is a function which eliminates as many elements from a given list as are contained in a second given list, and prints a list containing those elements <u>remaining</u> in the <u>first</u> list.

II. TALLYCOMPLEMENT is a function of two arguments:

M is a list, and

L is a list, where the number of elements in M is greater than the number of elements in L.

If L ontains more elements than M, a list of nils, equal to the difference in the number of elements of L and M, will result.

If L or M is an atom, there is no result.

If L and M contain the same number of elements, there is no result.

III. Machine Definition of TALLYCOMPLEMENT

```
(TALLYCOMPLEMENT (LAMBDA (L M)
(COND ((NULL L) M)
(T (TALLYCOMPLEMENT (CDR L) (CDR M))))))
```

IV. Examples.

```
  FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 TALLYCOMPLEMENT
 ((A,B,C,D,E),(E,F,G,H))


 END OF APPLY, VALUE IS ...
 ( )


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS.
TALLYCOMPLEMENT
((A,B,C),(E,G,H,J,K))


END OF APPLY, VALUE IS ...
(J,K)
```

(ORDINAL L)

I. ORDINAL is a function which makes a list of nils, the number of nils in the list is equal to the value of a given binary number.

ORDINAL is useful where some sort of counting device is necessary, as in INTERVAL.

II. ORDINAL is a function of one list, (L).
L is a binary number in list form.

ORDINAL takes advantage of the fact that TALLY of a binary number with NIL yields only the total number of nils <u>added by the definition</u>; the nils of the APPEND function reduce to nothing.
So for each zero preceding the binary number, nothing is the result; for each one in the binary number, one nil is added to the list and the preceding value of the list is doubled -- that is, literally multiplied by two.

III. Machine definition of ORDINAL.

```
(ORDINAL (LAMBDA (L) (TALLY L NIL)))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ORDINAL
((1,0,1))


END OF APPLY, VALUE IS ...
( , , , , )


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
ORDINAL
((1,0,1,0))


END OF APPLY, VALUE IS ...
( , , , , , , , , , )
```

(INTERVAL X Y)

I. INTERVAL is a function which lists the decimal numbers which lie between the decimal counterparts of two octal numbers, starting with the smaller number and ending with one less than the larger number.

II. INTERVAL is a function of two arguments: (X Y)

X is an octal number in list form and

Y is an octal number in list form.

The value of Y is greater than the value of X. If X is greater than Y, there is no result.

The values of Y and X are limited to less than or equal to 60_{10}, or 74_8, due to INTEGERS.

III. Machine definition of INTERVAL

```
(INTERVAL (LAMBDA (X Y) (TALLYCOMPLEMENT (ORDINAL (BINARY X))
(TALLYCOPY (ORDINAL (BINARY Y)) (INTEGERS)))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INTERVAL
((1,1),(2,1))


END OF APPLY, VALUE IS ...
(9,10,11,12,13,14,15,16)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
INTERVAL
((7,2),(6,2))


END OF APPLY, VALUE IS ...
```

(REPEAT X I)

I. REPEAT is a function which makes a list containing a given list repeated as many times as there are elements in a second given list.

II. REPEAT is a function of two arguments: (X I)
X may be anything - a nil, an atomic symbol, or a list.
I is a list

III. Machine definition of REPEAT.

```
(REPEAT (LAMBDA (X I)
(COND ((NULL I) NIL)
(T (CONS X (REPEAT X (CDR I)))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REPEAT
(Z,(1,1,1,1,1,1))


END OF APPLY, VALUE IS ...
(Z,Z,Z,Z,Z,Z)

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REPEAT
( ,(A,B,C,D,E,F))


END OF APPLY, VALUE IS ...
( , , , , , )

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REPEAT
((1,0),(Z,X,P,Q,R,T,Z))


END OF APPLY, VALUE IS ...
((1,0),(1,0),(1,0),(1,0),(1,0),(1,0),(1,0))

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
REPEAT
(( , ),(A,B,(C,D),E,F))


END OF APPLY, VALUE IS ...
(( , ),( , ),( , ),( , ),( , ))
```

(SUCHTHAT* L P A1 A2)

I. SUCHTHAT* is a function which takes a given value (or applies a given function) if there is an element in a given list for which a given property is true. Otherwise SUCHTHAT* takes a second given value (or function).

II. SUCHTHAT* is a function of four arguments: (L P A1 A2)

L	is a list of lists.
P	is a predicate to be tested on each element of L.
A1	is the value SUCHTHAT* takes if P is not true for any element of L. A1 may be a list or it may apply a given function to L or any of its parts.
A2	is the value SUCHTHAT* takes if P is true for some element of L. It, too, may be a list or function.

See lisp function DETIZE 1 for a good example of how SUCHTHAT* can have functions for arguments A1 and A2.

III. Machine definition

```
(SUCHTHAT* (LAMBDA (L P A1 A2)
(COND ((NULL L) A1)
((P (CAR L)) A2)
(T (SUCHTHAT* (CDR L) P A1 A2)))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 SUCHTHAT*
 (((A,B,C),(A,B,C)),(FUNCTION,(LAMBDA,(S),(ULTRATRIPLET,S))),
 (PREDICATE-FALSE),(OK))

 END OF APPLY, VALUE IS ...
 (OK)


 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
 SUCHTHAT*
 ((((A,C),D),((A,C),D)),(FUNCTION,(LAMBDA,(S),(EQUAL,(CAR,S),(CADR,S)))),
 (PREDICATE-FALSE),(O

 END OF APPLY, VALUE IS ...
 (PREDICATE-FALSE)
```

(SUCHTHAT** L P PUNCTION)

I. SUCHTHAT** is a function which takes a given value (or function) if and only if there exists an element in a given list for which a given property is true.

II. SUCHTHAT** is a function of three arguments: (L P PUNCTION)

L is a list of lists.

P is a predicate to be tested on each element of L.

PUNCTION is a list or function which is the value of SUCHTHAT** if predicate P is true for some element of L.

III. Machine definition

```
(SUCHTHAT** (LAMBDA (L P PUNCTION)
(COND ((NULL L) NIL)
((P (CAR L)) (PUNCTION L))
(T (SUCHTHAT** (CDR L) P PUNCTION)))))
```

IV. Examples.

```
FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SUCHTHAT**
(((A,B,C,D),(A,B,C,D)),(FUNCTION,(LAMBDA,(S),(ULTRATRIPLET,S))),
(FUNCTION,(LAMBDA,(S),(LIST,(QUOTE,**),S))))

END OF APPLY, VALUE IS ...
(**,((A,B,C,D),(A,B,C,D)))


FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SUCHTHAT**
(((A,A),(A,B)),(FUNCTION,(LAMBDA,(S),(EQUAL,(CAR,S),(CADR,S)))),
(FUNCTION,(LAMBDA,(S),(LIST,(QUOTE,****),S))))

END OF APPLY, VALUE IS ...
(****,((A,A),(A,B)))


FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SUCHTHAT**
(((A,B),(A,B)),(FUNCTION,(LAMBDA,(S),(ULTRATRIPLET,S))),
(FUNCTION,(LAMBDA,(S),(LIST,(QUOTE,**),S))))

END OF APPLY, VALUE IS ...
```

(DETERMINANTIZE L DIMENSION)

I. DETERMINANTIZE is a function which sets up a square matrix of a given size, and assigns values of 0 or 1 to each matrix element; a value of 1 is assigned only to those elements designated in a given list.

DETERMINANTIZE utilizes the lisp functions DETIZE, TALLYCOPY, ORDINAL, BINARY, and INTEGERS.

II. DETERMINANTIZE is a function of two arguments: (L DIMENSION)

L is a list of lists. The elements of the elements of L are decimal numbers, whose values are limited to equal to or less than 60_{10}, due to INTEGERS. The first element of each element of L indicates the column and the second indicates the row for each value of 1 in the matrix or determinant.

DIMENSION is an octal number in list form. It designates the size of the matrix.

The first row and the first column of the matrix are the zero th row and column, so the element of L which gives a 1 in the top left hand matrix element is (0,0).

III. Machine definition of the function

```
(DETERMINANTIZE (LAMBDA (L DIMENSION)
(DETIZE L (TALLYCOPY (ORDINAL (BINARY DIMENSION)) (INTEGERS))
(TALLYCOPY (ORDINAL (BINARY DIMENSION)) (INTEGERS)))))
```

IV. Examples.

```
FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANTIZE
(((1,1,0),(2,2,0),(3,3,0),(4,4,0),(5,5,0),(6,6)),(7))


END OF APPLY, VALUE IS ...
((0,0,0,C,0,0,0),(0,1,0,0,0,0,0),(0,0,1,0,0,0,0),(0,0,0,1,0,0,0),
(0,0,0,0,1,0,0),(0,0,0,0,0,1,0),(0,0,0,0,0,0,1))
```

(DETIZE L X Y)

I. DETIZE is a function which applies the function DETIZE 1 to a given list for each element in another given list. SEE DEFINITION OF DETIZE 1.

DETIZE utilizes the function DETIZE 1.

II. DETIZE is a function of three arguments: (L X Y)

- L is a list whose elements are ultra-doublets. DETIZE is concerned only with the first two elements of each element of L.
- X is a list. Each element of the product has as many elements as does X.
- Y is a list. There are as many elements in the product as there are in Y.

III. Machine definition of DETIZE

```
(DETIZE (LAMBDA (L X Y)
(COND ((NULL Y) NIL)
(T (CONS (DETIZE1 L X (CAR Y)) (DETIZE L X (CDR Y))))))))
```

IV. Examples.

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETIZE
(((O,N,E),(T,W,O),(T,R,I),(F,O,R)),(O,T,T,F),(N,W,R,O))


END OF APPLY, VALUE IS ...
((1,0,0,0),(0,1,1,0),(0,1,1,0),(0,0,0,1))



 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETIZE
(((A,B,C),(D,E,F),(G,H,I)),(A,B,G),(B))


END OF APPLY, VALUE IS ...
((1,0,0))
```

(DETIZE1 L X B)

I. DETIZE1 is a function which checks to see if the first two elements of any of the elements of a given list are equal to the corresponding element of a second list and a given element, respectively. A value is assigned, according to whether the elements are equal or not.

DETIZE1 utilizes the function SUCHTHAT*

II. DETIZE1 is a function of three arguments: (L X B)

L is a list of lists. No matter how long the lists of the elements of L are, DETIZE1 considers only the first two elements of each element of L.

X is a list.

B may be an atomic symbol, or a list, according to whether or not the second element of each element of L is a list or an atomic symbol.

A value of 1 is assigned to each element of X for which there exists in L an element whose first element is equal to the element of X and whose second element is equal to B.

Otherwise a value 0 is assigned to the element of X. The number of elements in X determine the number of values assigned in the product.

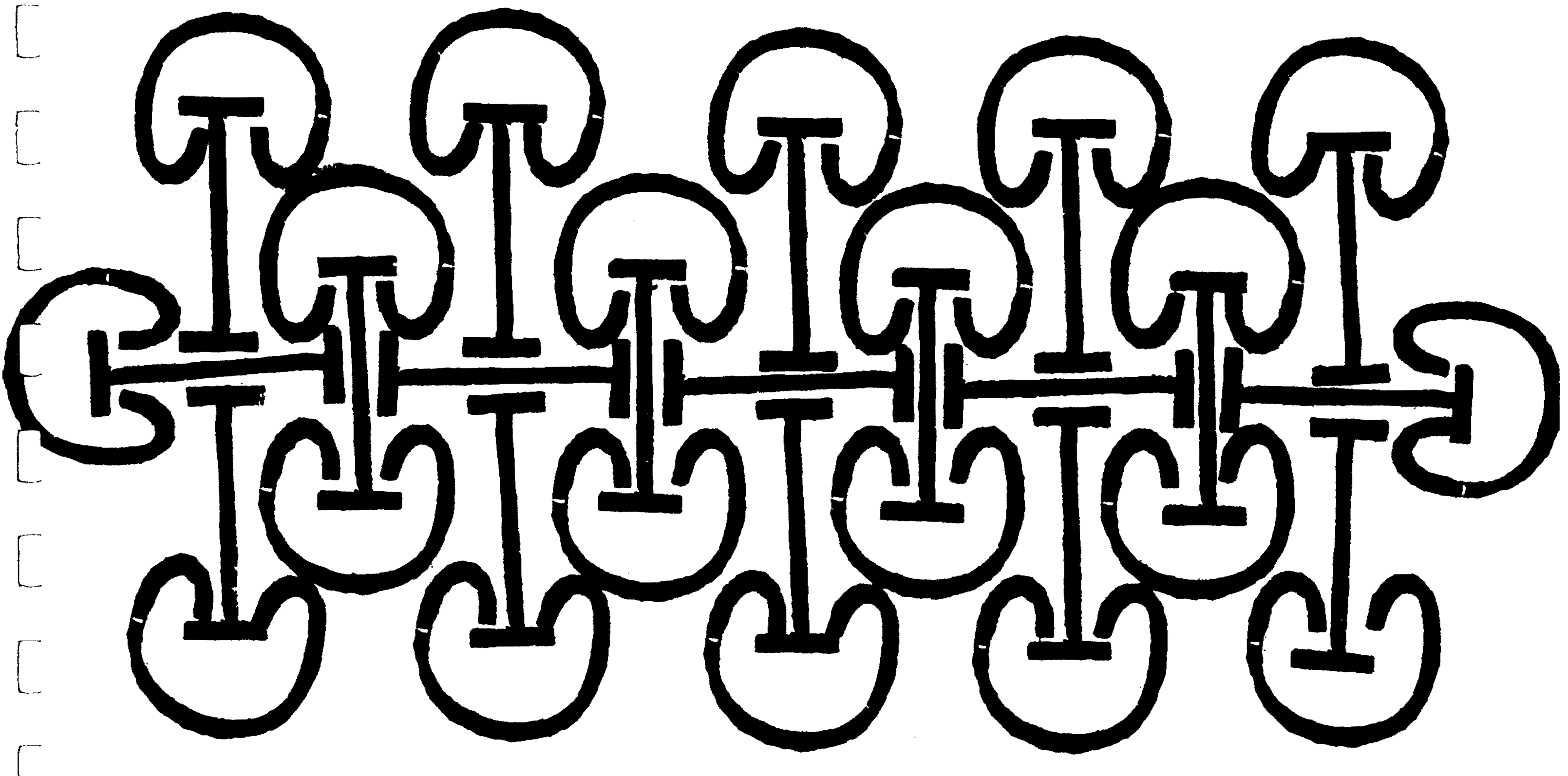
III. Machine definition.

```
(DETIZE1 (LAMBDA (L X B)
(COND ((NULL X) NIL)
(T (CONS (SUCHTHAT* L (FUNCTION (LAMBDA (L) (AND
EQUAL (CAR L) (CAR X)) (EQUAL (CADR L) B)))) (QUOTE 0) (QUOTE 1))
(DETIZE1 L (CDR X) B))))))
```

IV. Examples

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETIZE1
(((A,B),(C,D),(A,B),(C,B),(D,B),(E,F),(J,B)),(A,Z,Y,C,D,E,J),B)


END OF APPLY, VALUE IS ...
(1,0,0,1,1,0,1)
```

Specialized functions and their satellites



THE LISP FUNCTION DISPLAY

DISPLAY is a function of three arguments:

```
(DISPLAY L X Y)
```

The first of these, L, is a list of triplets:

$$L = ((x_1,y_1,*_1), (x_2,y_2,*_2), \ldots)$$

where the members of the triplet have the following significance:

x_i is the x-coordinate of a point to be displayed

y_i is the y-coordinate of a point to be displayed

$*_i$ is a character to be displayed.

x_i is measured across the page, and is a decimal number. A page will accomodate 57 characters across its width, since the margin is reserved for the printer, and alternate characters are commas. In addition, a special symbol is used at the end of a line to correct for the odd number of spaces available.

y_i is measured down from the top of the page; as many lines may be displayed as the memory capacity of the computer allows.

Both x_i and y_i are measured from zero.

A character may be attached to a list of coordinates by using the LISP function KARTESIAN*, viz:

```
(KARTESIAN* L (LIST (QUOTE *))).
```

The second argument X is an octal number giving the length of the rows. The digits of this octal number must be listed, viz:

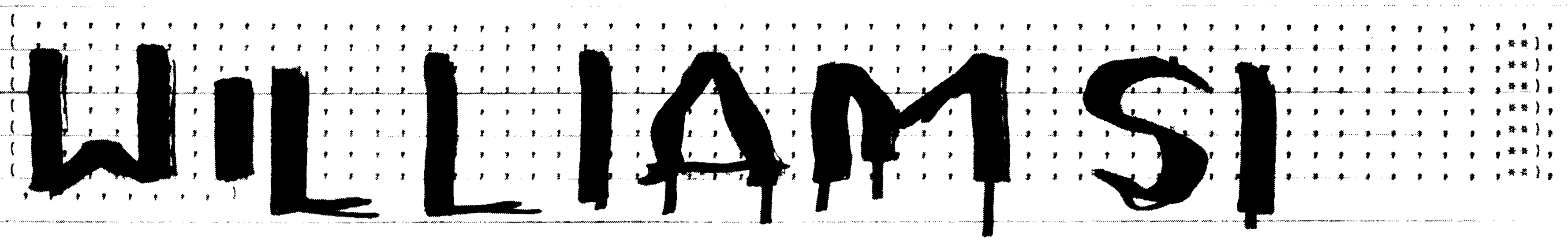
$$X = (3,7).$$

The length of a row may range from 1_8 to 70_8.

In a similar fashion the third argument Y gives, in octal form the number of rows which the display is to contain.

```
(DISPLAY (LAMBDA (L X Y)
(APPEND (ORDINAL (BINARY (LIST (QUOTE 7) (QUOTE 3))))
(DISPLAY1
L
(TALLYCOPY (ORDINAL (BINARY X)) (INTEGERS))
(TALLYCOPY (ORDINAL (BINARY Y)) (INTEGERS))
(TALLYCOMPLEMENT (ORDINAL (BINARY X)) (ORDINAL  (BINARY (LIST
(QUOTE 7) (QUOTE 1))))))
(TALLYCOMPLEMENT (ORDINAL (BINARY X)) (ORDINAL  (BINARY (LIST
(QUOTE 7) (QUOTE 1)))))))))))


(DISPLAY1 (LAMBDA (L X Y ROWEND PAGEEND)
(COND ((NULL Y) PAGEEND)
(T (CONS (APPEND
(DISPLAY2 (POSSESSING (FUNCTION (LAMBDA (X)
(EQUAL (CADR X) (CAR Y)))) L) X Y)
(APPEND ROWEND (LIST (QUOTE **))))
(DISPLAY1 L X (CDR Y) ROWEND PAGEEND))))))

(DISPLAY2 (LAMBDA (L X Y)
(COND ((NULL X) NIL)
(T (CONS (SUCHTHAT** L (FUNCTION (LAMBDA (L)
(AND (EQUAL (CAR L) (CAR X))
(EQUAL (CADR L) (CAR Y)))))
(FUNCTION (LAMBDA (L) (CADDAR L))))
                                    (DISPLAY2 L (CDR X) Y))))))
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DISPLAY
((((1,1,*),(1,2,*),(2,1,*),(2,2,*),(3,3,*),(4,4,*),(5,5,*),(6,6,*),(3,4,=)),(1,0),(1,0))


GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=    37 IR4 ON PDL=     11    GARBAGE= 12634

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   129 IR4 ON PDL=     51    GARBAGE= 12161

END OF APPLY, VALUE IS ...
( , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,
( , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( ,*,*, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( ,*,*, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( , , ,*, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( , , ,=,*, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( , , , , ,*, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( , , , , , ,*, , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
( , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , ,**),
 , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , , )



 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
(LAMBDA,(L,M,N),(DISPLAY,(KARTESIAN*,L,(LIST,(QUOTE,*))),M,N))
((((1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),(2,6),(3,5),(4,6),(5,1),(5,2),(5,3),(5,4),(5,5),(5,6),(5,7),(8,2),(8,2),(8,
3),(8,4),(8,5),(3,6),(8,7),(10,2),(10,3),(10,4),(10,5),(10,6),(10,7),(11,7),(12,7),(13,7),(14,7),(16,1),(16,2),(16,3),(
16,4),(16,5),(16,5),(16,6),(16,7),(17,7),(18,7),(19,7),(20,7),(22,1),(22,2),(22,3),(22,4),(22,5),(22,6),(22,7),(24,6),(
24,7),(25,3),(25,4),(25,5),(25,6),(26,1),(26,2),(26,6),(27,1),(27,2),(27,6),(28,3),(28,4),(28,5),(28,6),(29,6),(29,7),(
31,1),(31,2),(31,3),(31,4),(31,5),(31,6),(31,7),(32,2),(33,3),(33,4),(33,5),(36,2),(37,1),(37,2),(37,3),(37,4),(37,5),(
37,5),(37,7),(41,2),(41,3),(41,6),(42,1),(42,3),(42,7);(43,1),(43,4),(43,7),(44,1),(44,4),(44,7),(45,2),(45,5),(45,6),(
47,2),(47,1),(47,3),(64,6),(64,7),(65,2),(66,3),(66,4),(66,5),(67,6),(68,1),(68,1)),(6,0),(7))
```



DETERMINANT

DETERMINANT is a function of one argument, which is a list of the rows of the determinant which is proposed to be evaluated. Matrix elements of the determinant should be given in polish notation to be consistent with the evaluated form of the determinant. The reduction is done by the method of Gaussian elimination, and no attempt is made to recognize a zero matrix element occuring as a divisor unless it is explicitly zero. Some rudimentary simplification is made when a zero or a one occurs explicitly. The method is not recommended without some additional simplification for complicated determinants.

```
OBJECT LIST NOW IS ...
(AL,++++,++,QUXL,X*,EXPGO2,NEGATIVE,MINUSGO1,MINUSGO2,MINUSGOR,OP,MINUSGOL,ZEROGO1,ZEROGO2,ZEROGO,EXPGO,MINUSGO,DERIVAT
IVE,LN,COS,SIN,DERFUN,DIFFO,ZEROBEGONE,DIFFERENTIATE,ILLEGAL,DIFF1,DIFF2,**,+,DIFF,DETERMINANT,ABSDET,/,COREDUCE,*,MULT
IPLIER,-,DIFFERENCE,RELSIGN,REDUCE,NONZERO,ENGLISH,POLISH,DOUBLET,TRIPLET,PF,VAR,FORM,SEARCHF,T1,UF,FF,DF,COMPSRCH,BNDV
,SELECT,CSETQ,CONC,CDDDDR,CDDDAR,CDDADR,CDDAAR,CDADDR,CDADAR,CDAADR,CDAAAR,CADDDR,CADDAR,CADADR,CADAAR,CAADDR,CAADAR,CA
AADR,CAAAAR,VALUE,ARGUMENTS,TERRIBLYUNIQUEPROGRAMVARIABLE,UNTRACLIS,TRACLIS,PROPERTY,ITS,HAS,TRAC3,TRACLISED,UNTRAC1,PR
OPER,IN,DEFINITION,TRAC2,NAME,FN,TRAC1,M,B,SELECT2,REMP1,OF,PROPERTIES,PRINTPROP,PROG2,PRINTP1,D,U,MAPCAR,WAY,J,MAKCBLR
,PICK,R,EQUALS,NO,ATOMIC,FORMATQ,FORMATP,P,TPXD0303A,S,FTQZ01R,N,FORMAT,AS,TO,GIVEN,WAS,V,O,CSET,CONC1,DEFINE,DEFLIS1,D
EFLIS1,PRO,L,OB,DEF1,A,AND,APPEND,APPLY,APVAL,APVAL1,ATOM,ATTRIB,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,CAR,CDR,,,COM
PTRAC,COND,CONS,CONST,CP1,COPY,COUNT,DESC,EQ,EQ1,EQUAL,ERROR,EVAL,EVLIS,EXPR,EXPT,F,FEXPR,FIX,FLO,FSUBR,FUNARG,FUNCTION
,GENSYM,GO,INST,INT,INTERN,LABEL,LAMBDA,LIST,LOAD,LOC,LOCQ,(,MAKENU,MAKEOB,MAP,MAPCON,MAPLIST,-1,-2,MINUS,NCONC,NFLVAL,
NIL,NFWL,NOT,NULL,NUMBER,OBLIST,OR,PAIR,PAUSE,PFWL,PLB,PLUS,PNAME,POWER,PRDCT,PRINT,PRIN2,PROG,PROP,QUOTE,READ,RECIP,RE
CLAIM,RPLACA,RPLACD,RPLACW,RETURN,),SASSOC,SEARCH,SET,SETQ,SPEAK,STOP,SUB,SUBR,SUBLIS,SUBST,SUM,SYMBOL,T,TEST1,TEST2,TE
ST3,TEST4,TIMES,TRACE,TSFLOT,UNCOUNT,ADD,ALS,ARS,BSS,CLA,COM,LDQ,LXA,LXD,PAX,PDX,PXD,STA,STO,STD,STQ,SXD,TIX,TNX,TNZ,TR
A,TSX,TXH,TXI,TXL,TZE,ATOM1,EQ11,NULL1,BIN,EXP,FUNC,TEMF,TEMP,AC,MQ,$ARG2,$ARG3,$ARG4,$ARG5,$ARG6,$ARG7,$ARG8,*+1,*+2,*
+3,*+4,*+5,CAAR,CDAR,CADR,CDDR,CAAAR,CAADR,CADAR,CADDR,CDAAR,CDADR,CDDAR,CDDDR,$CPPI,$ENPDL,$NOPDL+1,$FREE,$FROUT,$ONE,
$ZERO,X,Y,Z,COMPILE,COMP2,COMPAT,DEFF3,GET,NOMAP,SAP,REMPROP, INTEGRATED 709 COMPILER-INTERPRETER LISP 20 OCT 60   )    .

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((TRIPLET,(LAMBDA,(L),(COND,((OR,(ATOM,L),(NULL,L),(NULL,(CDR,L)),(NULL,(CDDR,L))),F),(T,(COND,((NULL,(CDDDR,L)),T),(T
,F))))))),(DOUBLET,(LAMBDA,(L),(COND,((OR,(ATOM,L),(NULL,L),(NULL,(CDR,L))),F),(T,(COND,((NULL,(CDDR,L)),T),(T,F))))))),(
POLISH,(LAMBDA,(L),(COND,((TRIPLET,L),(LIST,(CADR,L),(POLISH,(CAR,L)),(POLISH,(CADDR,L)))),((DOUBLET,L),(LIST,(CAR,L),(
POLISH,(CADR,L)))),(T,L)))),(ENGLISH,(LAMBDA,(L),(COND,((TRIPLET,L),(LIST,(ENGLISH,(CADR,L)),(CAR,L),(ENGLISH,(CADDR,L)
))),((DOUBLET,L),(LIST,(CAR,L),(ENGLISH,(CADR,L)))),(T,L))))))


END OF APPLY, VALUE IS ...
(TRIPLET,DOUBLET,POLISH,ENGLISH)



 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((NONZERO,(LAMBDA,(L),(COND,((NULL,(CDR,L)),(CAR,L)),((NOT,(EQUAL,(CAAR,L),(QUOTE,0))),(CAR,L)),(T,(NONZERO,(CDR,L))))
)),(REDUCE,(LAMBDA,(L,M),(COND,((NOT,(EQUAL,(CAR,L),M)),(CONS,(CAR,L),(REDUCE,(CDR,L),M))),(T,(CDR,L))))),(RELSIGN,(LAM
BDA,(L,M),(COND,((EQUAL,(CAR,L),M),T),(T,(NOT,(RELSIGN,(CDR,L),M)))))),(DIFFERENCE,(LAMBDA,(L,M),(COND,((NULL,L),NIL),(
(AND,(EQUAL,(CAR,L),(QUOTE,0)),(EQUAL,(CAR,M),(QUOTE,0))),(CONS,(QUOTE,0),(DIFFERENCE,(CDR,L),(CDR,M)))),((EQUAL,(CAR,L
),(QUOTE,0)),(CONS,(LIST,(QUOTE,-),(CAR,M)),(DIFFERENCE,(CDR,L),(CDR,M)))),((EQUAL,(CAR,M),(QUOTE,0)),(CONS,(CAR,L),(DI
FFERENCE,(CDR,L),(CDR,M)))),((EQUAL,(CAR,L),(CAR,M)),(CONS,(QUOTE,0),(DIFFERENCE,(CDR,L),(CDR,M)))),(T,(CONS,(LIST,(QUO
TE,-),(CAR,L),(CAR,M)),(DIFFERENCE,(CDR,L),(CDR,M))))))),(MULTIPLIER,(LAMBDA,(L,A),(COND,((NULL,L),NIL),((OR,(EQUAL,A,(
QUOTE,0)),(EQUAL,(CAR,L),(QUOTE,0)))√(CONS,(QUOTE,0),(MULTIPLIER,(CDR,L),A))),((EQUAL,A,(QUOTE,1)),L),((EQUAL,(CAR,L),(
QUOTE,1))√(CONS,A,(MULTIPLIER,(CDR,L),A))),(T,(CONS,(LIST,(QUOTE,*),A,(CAR,L)),(MULTIPLIER,(CDR,L),A)))))),(COREDUCE,(L
AMBDA,(L,R),(COND,((NULL,L),NIL),((EQUAL,(CAAR,L),(QUOTE,0)),(CONS,(CDAR,L),(COREDUCE,(CDR,L),R))),((EQUAL,(CAAR,L),(CA
R,R)),(CONS,(DIFFERENCE,(CDAR,L),(CDR,R)),(COREDUCE,(CDR,L),R))),(T,(CONS,(DIFFERENCE,(CDAR,L),(MULTIPLIER,(CDR,R),(LIS
T,(QUOTE,/),(CAAR,L),(CAR,R)))),(COREDUCE,(CDR,L),R)))))),(ABSDET,(LAMBDA,(L),(LIST,(QUOTE,*),(CAR,(NONZERO,L)),(DETERM
INANT,(COREDUCE,(REDUCE,L,(NONZERO,L)),(NONZERO,L)))))),(DETERMINANT,(LAMBDA,(L),(COND,((NULL,L),NIL),((NULL,(CDR,L)),(
CAAR,L)),((EQUAL,(CAR,(NONZERO,L)),(QUOTE,0)),(QUOTE,0)),((RELSIGN,L,(NONZERO,L)),(ABSDET,L)),(T,(LIST,(QUOTE,-),(ABSDE
T,L))))))))


END OF APPLY, VALUE IS ...
(NONZERO,REDUCE,RELSIGN,DIFFERENCE,MULTIPLIER,COREDUCE,ABSDET,DETERMINANT)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
(((0,A,B),(0,C,D),(0,E,F)))


END OF APPLY, VALUE IS ...
0
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
(((0,0,1),(0,1,0),(1,0,0)))


END OF APPLY, VALUE IS ...
(*,1,(-,(*,1,1)))
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
(((K,0,0,0),(K,K,K,K),(M,0,0,J),(A,B,C,D)))


END OF APPLY, VALUE IS ...
(*,K,(*,K,(-,(*,(-,C,(*,(/,B,K),K)),J))))
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
(((K,0,0),(0,A,C),(0,C,D)))


END OF APPLY, VALUE IS ...
(*,K,(*,A,(-,D,(*,(/,C,A),C))))
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
(((A,A,A),(A,A,A),(A,A,A)))


END OF APPLY, VALUE IS ...
(*,A,0)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
(((A,B,C),(0,0,0),(D,E,F)))


END OF APPLY, VALUE IS ...
(*,A,(-,(*,(-,E,(*,(/,D,A),B)),0)))
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DETERMINANT
((((-,AB,LAMBDA),B,0,0,0,B),(B,(-,AN,LAMBDA),B,0,0,0),(0,B,(-,AB,LAMBDA),B,0,0),(0,0,B,(-,AN,LAMBDA),B,0),(0,0,0,B,(-,A
B,LAMBDA),B),(B,0,0,0,B,(-,AN,LAMBDA))))


END OF APPLY, VALUE IS ...
(*,(-,AB,LAMBDA),(*,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B)),(*,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-
,AB,LAMBDA)),B))),B)),(*,(-,(-,AN,LAMBDA),(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B)))
,B))),B)),(*,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,
LAMBDA)),B))),B))),B))),B)),(-,(-,(-,(-,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B)),(*,(/,(-,(*,(/,B,(-,AB,LAMBDA)),B))
,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),(-,(*,(/,B,(-,AB,LAMBDA)),B))))),(*,(/,(-,(*,(/,(-,(*,(/,B,(-,AB,LAMBDA)),
B)),(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B)),(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B
))),B)))),(-,(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),(-,(*,(/,B,(-,AB,LAMBDA)),B))))))),(*,(/,(-,(*,(/,(-,(*
,(/,(-,(*,(/,B,(-,AB,LAMBDA)),B)),(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B)),(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LA
MBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B))),B)),(-,(-,AN,LAMBDA),(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(
-,AB,LAMBDA)),B))),B))),B))),(-,(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B))),(-,(
*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),(-,(*,(/,B,(-,AB,LAMBDA)),B)))))))),(*,(/,(-,B,(*,(/,(-,(*,(/,(-,(*
,(/,(-,(*,(/,B,(-,AB,LAMBDA)),B)),(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B)),(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LA
MBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B))),B)),(-,(-,AN,LAMBDA),(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(
-,AB,LAMBDA)),B))),B))),B))),B)),(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LA
MBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B))),B))),B))),(-,B,(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,A
N,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),B))),B))),(-,(*,(/,B,(-,(-,AB,LAMBDA),(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMB
DA)),B))),B))),(-,(*,(/,B,(-,(-,AN,LAMBDA),(*,(/,B,(-,AB,LAMBDA)),B))),(-,(*,(/,B,(-,AB,LAMBDA)),B)))))))))))))))
```

DIFFERENTIATE

DIFFERENTIATE is a function of two arguments. One of them, the first, is a function to be differentiated, and the second, is the variable with respect to which differentiation shall take place. The first argument must be given in the form of a list, with parentheses and multiplication indicated explicitly in ordinary english notation,,no more than two symbols connected by an arithmetic sign. The result is simplified slightly in that redundant zeros, factors of one, and the like are suppressed.

```
APPLY OPERATOR AS OF FEBRUARY 10, 1960

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

READ IN LISTS ...
DEFINE
(((TRIPLET,(LAMBDA,(L),(COND,((OR,(ATOM,L),(NULL,L),(NULL,(CDR,L)),(NULL,(CDDR,L))),F),(T,(COND,((NULL,(CDDDR,L)),T),(T
,F)))))),(DOUBLET,(LAMBDA,(L),(COND,((OR,(ATOM,L),(NULL,L),(NULL,(CDR,L))),F),(T,(COND,((NULL,(CDDR,L)),T),(T,F)))))),(
POLISH,(LAMBDA,(L),(COND,((TRIPLET,L),(LIST,(CADR,L),(POLISH,(CAR,L)),(POLISH,(CADDR,L)))),((DOUBLET,L),(LIST,(CAR,L),(
POLISH,(CADR,L)))),(T,L)))),(ENGLISH,(LAMBDA,(L),(COND,((TRIPLET,L),(LIST,(ENGLISH,(CADR,L)),(CAR,L),(ENGLISH,(CADDR,L)
))),((DOUBLET,L),(LIST,(CAR,L),(ENGLISH,(CADR,L)))),(T,L))))))

DEFINE
(((NONZERO,(LAMBDA,(L),(COND,((NULL,(CDR,L)),(CAR,L)),((NOT,(EQUAL,(CAAR,L),(QUOTE,0))),(CAR,L)),(T,(NONZERO,(CDR,L))))
)),(REDUCE,(LAMBDA,(L,M),(COND,((NOT,(EQUAL,(CAR,L),M)),(CONS,(CAR,L),(REDUCE,(CDR,L),M))),(T,(CDR,L))))),(RELSIGN,(LAM
BDA,(L,M),(COND,((EQUAL,(CAR,L),M),T),(T,(NOT,(RELSIGN,(CDR,L),M)))))),(DIFFERENCE,(LAMBDA,(L,M),(COND,((NULL,L),NIL),(
(AND,(EQUAL,(CAR,L),(QUOTE,0)),(EQUAL,(CAR,M),(QUOTE,0))),(CONS,(QUOTE,0),(DIFFERENCE,(CDR,L),(CDR,M)))),((EQUAL,(CAR,L
),(QUOTE,0)),(CONS,(LIST,(QUOTE,-),(CAR,M)),(DIFFERENCE,(CDR,L),(CDR,M)))),((EQUAL,(CAR,M),(QUOTE,0)),(CONS,(CAR,L),(DI
FFERENCE,(CDR,L),(CDR,M)))),((EQUAL,(CAR,L),(CAR,M)),(CONS,(QUOTE,0),(DIFFERENCE,(CDR,L),(CDR,M)))),(T,(CONS,(LIST,(QUO
TE,-),(CAR,L),(CAR,M)),(DIFFERENCE,(CDR,L),(CDR,M))))))),(MULTIPLIER,(LAMBDA,(L,A),(COND,((NULL,L),NIL),((OR,(EQUAL,A,(
QUOTE,0)),(EQUAL,(CAR,L),(QUOTE,0))),(CONS,(QUOTE,0),(MULTIPLIER,(CDR,L),A))),((EQUAL,A,(QUOTE,1)),L),((EQUAL,(CAR,L),(
QUOTE,1)),(CONS,A,(MULTIPLIER,(CDR,L),A))),(T,(CONS,(LIST,(QUOTE,*),A,(CAR,L)),(MULTIPLIER,(CDR,L),A)))))),(COREDUCE,(L
AMBDA,(L,R),(COND,((NULL,L),NIL),((EQUAL,(CAAR,L),(QUOTE,0)),(CONS,(CDAR,L),(COREDUCE,(CDR,L),R))),((EQUAL,(CAAR,L),(CA
R,R)),(CONS,(DIFFERENCE,(CDAR,L),(CDR,R)),(COREDUCE,(CDR,L),R))),(T,(CONS,(DIFFERENCE,(CDAR,L),(MULTIPLIER,(CDR,R),(LIS
T,(QUOTE,/),(CAAR,L),(CAR,R)))),(COREDUCE,(CDR,L),R)))))),(ABSDET,(LAMBDA,(L),(LIST,(QUOTE,*),(CAR,(NONZERO,L)),(DETERM
INANT,(COREDUCE,(REDUCE,L,(NONZERO,L)),(NONZERO,L)))))),(DETERMINANT,(LAMBDA,(L),(COND,((NULL,L),NIL),((NULL,(CDR,L)),(
CAAR,L)),((EQUAL,(CAR,(NONZERO,L)),(QUOTE,0)),(QUOTE,0)),((RELSIGN,L,(NONZERO,L)),(ABSDET,L)),(T,(LIST,(QUOTE,-),(ABSDE
T,L))))))))

DEFINE
(((DIFF,(LAMBDA,(L,X),(COND,((ATOM,L),(COND,((EQUAL,L,X),(QUOTE,1)),(T,(QUOTE,0)))),((TRIPLET,L),(COND,((EQUAL,(CAR,L),
(QUOTE,+)),(LIST,(QUOTE,+),(DIFF,(CADR,L),X),(DIFF,(CADDR,L),X))),((EQUAL,(CAR,L),(QUOTE,-)),(LIST,(QUOTE,-),(DIFF,(CAD
R,L),X),(DIFF,(CADDR,L),X))),((EQUAL,(CAR,L),(QUOTE,*)),(LIST,(QUOTE,+),(LIST,(QUOTE,*),(CADR,L),(DIFF,(CADDR,L),X)),(L
IST,(QUOTE,*),(DIFF,(CADR,L),X),(CADDR,L)))),((EQUAL,(CAR,L),(QUOTE,/)),(LIST,(QUOTE,/),(LIST,(QUOTE,-),(LIST,(QUOTE,*)
,(DIFF,(CADR,L),X),(CADDR,L)),(LIST,(QUOTE,*),(CADR,L),(DIFF,(CADDR,L),X))),(LIST,(QUOTE,**),(CADDR,L),(QUOTE,2)))),((E
QUAL,(CAR,L),(QUOTE,**)),(LIST,(QUOTE,*),(CADDR,L),(LIST,(QUOTE,**),(CADR,L),(LIST,(QUOTE,-),(CADDR,L),(QUOTE,1))))),(T
,(DIFF2,L,X)))),((DOUBLET,L),(COND,((EQUAL,(CAR,L),(QUOTE,-)),(LIST,(QUOTE,-),(DIFF,(CADR,L),X))),(T,(DIFF1,L,X)))),(T,
(DIFF2,L,X))))),(DIFF2,(LAMBDA,(L,X),(LIST,(QUOTE,ILLEGAL,FORMAT),L))),(DIFFERENTIATE,(LAMBDA,(L,X),(ENGLISH,(ZEROBEGON
E,(DIFF,(POLISH,L),X)))))))

DEFINE
(((DIFFO,(LAMBDA,(L,X,DERFUN),(LIST,(QUOTE,*),DERFUN,(DIFF,(CADR,L),X)))),(DIFF1,(LAMBDA,(L,X),(COND,((EQUAL,(CAR,L),(Q
UOTE,EXP)),(DIFFO,L,X,(LIST,(QUOTE,EXP),(CADR,L)))),((EQUAL,(CAR,L),(QUOTE,SIN)),(DIFFO,L,X,(LIST,(QUOTE,COS),(CADR,L))
)),((EQUAL,(CAR,L),(QUOTE,COS)),(DIFFO,L,X,(LIST,(QUOTE,-),(LIST,(QUOTE,SIN),(CADR,L))))),((EQUAL,(CAR,L),(QUOTE,LN)),(
DIFFO,L,X,(LIST,(QUOTE,/),(QUOTE,1),(CADR,L)))),(T,(DIFFO,L,X,(LIST,(LIST,(QUOTE,DERIVATIVE),(CAR,L)),(CADR,L)))))))))

DEFINE
(((ZEROBEGONE,(LAMBDA,(L),(COND,((EQUAL,(MINUSGO,(EXPGO,(ZEROGO,L))),L),L),(T,(ZEROBEGONE,(MINUSGO,(EXPGO,(ZEROGO,L))))
))))))

DEFINE
(((ZEROGO2,(LAMBDA,(L),(LIST,(CAR,L),(ZEROGO,(CADR,L)),(ZEROGO,(CADDR,L))))),(ZEROGO1,(LAMBDA,(L),(LIST,(CAR,L),(ZEROGO
,(CADR,L))))),(ZEROGO,(LAMBDA,(L),(COND,((TRIPLET,L),(COND,((EQUAL,(CAR,L),(QUOTE,*)),(COND,((EQUAL,(CADR,L),(QUOTE,0))
,(QUOTE,0)),((EQUAL,(CADDR,L),(QUOTE,0)),(QUOTE,0)),((EQUAL,(CADR,L),(QUOTE,1)),(ZEROGO,(CADDR,L))),((EQUAL,(CADDR,L),(
QUOTE,1)),(ZEROGO,(CADR,L))),(T,(ZEROGO2,L)))),((EQUAL,(CAR,L),(QUOTE,/)),(COND,((EQUAL,(CADR,L),(QUOTE,0)),(QUOTE,0)),
((EQUAL,(CADDR,L),(QUOTE,1)),(ZEROGO,(CADR,L))),(T,(ZEROGO2,L)))),((EQUAL,(CAR,L),(QUOTE,+)),(COND,((EQUAL,(CADR,L),(QU
OTE,0)),(ZEROGO,(CADDR,L))),((EQUAL,(CADDR,L),(QUOTE,0)),(ZEROGO,(CADR,L))),(T,(ZEROGO2,L)))),((EQUAL,(CAR,L),(QUOTE,-)
),(COND,((EQUAL,(CADR,L),(QUOTE,0)),(LIST,(QUOTE,-),(ZEROGO,(CADDR,L)))),((EQUAL,(CADDR,L),(QUOTE,0)),(ZEROGO,(CADR,L))
),(T,(ZEROGO2,L)))),(T,(ZEROGO2,L)))),((DOUBLET,L),(COND,((AND,(EQUAL,(CAR,L),(QUOTE,-)),(EQUAL,(CADR,L),(QUOTE,0))),(Q
UOTE,0)),(T,(ZEROGO1,L)))),(T,L))))))
```

```
DEFINE
(((MINUSGOL,(LAMBDA,(L,OP),(LIST,OP,(MINUSGO,(CADADR,L)),(MINUSGO,(CADDR,L))))),(MINUSGOR,(LAMBDA,(L,OP),(LIST,OP,(MINU
SGO,(CADR,L)),(MINUSGO,(CAR,(CDADDR,L)))))),(MINUSGO2,(LAMBDA,(L),(LIST,(CAR,L),(MINUSGO,(CADR,L)),(MINUSGO,(CADDR,L)))
)),(MINUSGO1,(LAMBDA,(L),(LIST,(CAR,L),(MINUSGO,(CADR,L))))),(NEGATIVE,(LAMBDA,(L),(COND,((EQUAL,L,(QUOTE,+)),(QUOTE,-)
),((EQUAL,L,(QUOTE,-)),(QUOTE,+)),(T,L))))),(MINUSGO,(LAMBDA,(L),(COND,((TRIPLET,L),(COND,((OR,(EQUAL,(CAR,L),(QUOTE,*))
,(EQUAL,(CAR,L),(QUOTE,/))),(COND,((AND,(DOUBLET,(CADR,L)),(EQUAL,(CAADR,L),(QUOTE,-))),(LIST,(QUOTE,-),(MINUSGOL,L,(CA
R,L)))),((AND,(DOUBLET,(CADDR,L)),(EQUAL,(CAADDR,L),(QUOTE,-))),(LIST,(QUOTE,-),(MINUSGOR,L,(CAR,L)))),(T,(MINUSGO2,L))
)),((OR,(EQUAL,(CAR,L),(QUOTE,+)),(EQUAL,(CAR,L),(QUOTE,-))),(COND,((AND,(DOUBLET,(CADR,L)),(EQUAL,(CAADR,L),(QUOTE,-))
),(LIST,(QUOTE,-),(MINUSGOL,L,(NEGATIVE,(CAR,L))))),((AND,(DOUBLET,(CADDR,L)),(EQUAL,(CAADDR,L),(QUOTE,-))),(MINUSGOR,L
,(NEGATIVE,(CAR,L)))),(T,(MINUSGO2,L)))),((AND,(EQUAL,(CAR,L),(QUOTE,**)),(DOUBLET,(CADDR,L)),(EQUAL,(CAADDR,L),(QUOTE,
-))),(LIST,(QUOTE,/),(QUOTE,1),(MINUSGOR,L,(CAR,L)))),(T,(MINUSGO2,L)))),((DOUBLET,L),(COND,((AND,(EQUAL,(CAR,L),(QUOTE
,-)),(DOUBLET,(CADR,L)),(EQUAL,(CAADR,L),(QUOTE,-))),(MINUSGO,(CADADR,L))),(T,(MINUSGO1,L)))),(T,L))))))

DEFINE
(((EXPGO,(LAMBDA,(L),(COND,((TRIPLET,L),(COND,((AND,(EQUAL,(CAR,L),(QUOTE,-)),(EQUAL,(CADDR,L),(QUOTE,1))),(COND,((EQUA
L,(CADR,L),(QUOTE,10)),(QUOTE,9)),((EQUAL,(CADR,L),(QUOTE,9)),(QUOTE,8)),((EQUAL,(CADR,L),(QUOTE,8)),(QUOTE,7)),((EQUAL
,(CADR,L),(QUOTE,7)),(QUOTE,6)),((EQUAL,(CADR,L),(QUOTE,6)),(QUOTE,5)),((EQUAL,(CADR,L),(QUOTE,5)),(QUOTE,4)),((EQUAL,(
CADR,L),(QUOTE,4)),(QUOTE,3)),((EQUAL,(CADR,L),(QUOTE,3)),(QUOTE,2)),((EQUAL,(CADR,L),(QUOTE,2)),(QUOTE,1)),((EQUAL,(CA
DR,L),(QUOTE,1)),(QUOTE,0)),((EQUAL,(CADR,L),(QUOTE,0)),(QUOTE,(-,1))),(T,(CONS,(CAR,L),(CONS,(EXPGO,(CADR,L)),(CDDR,L)
))))),((EQUAL,(CAR,L),(QUOTE,**)),(COND,((EQUAL,(CADR,L),(QUOTE,1)),(QUOTE,1)),((EQUAL,(CADDR,L),(QUOTE,1)),(EXPGO,(CAD
R,L))),((EQUAL,(CADR,L),(QUOTE,0)),(QUOTE,0)),((EQUAL,(CADDR,L),(QUOTE,0)),(QUOTE,1)),(T,(EXPGO2,L)))),(T,(EXPGO2,L))))
,((DOUBLET,L),(LIST,(CAR,L),(EXPGO,(CADR,L)))),(T,L)))),(EXPGO2,(LAMBDA,(L),(LIST,(CAR,L),(EXPGO,(CADR,L)),(EXPGO,(CADD
R,L)))))))

DIFFERENTIATE
((X,-,Y),X)

DIFFERENTIATE
((X,-,Y),Y)

DIFFERENTIATE
((X,+,Y),X)

DIFFERENTIATE
((X,*,Y),X)

DIFFERENTIATE
((X,/,Y),Y)

DIFFERENTIATE
((X,**,3),X)

DIFFERENTIATE
((SIN,(X*,*,Y)),Y)

DIFFERENTIATE
((QUXL,X),X)

DIFFERENTIATE
(((+,+,+,+,+,+),(++),(++++),(+)),+)

DIFFERENTIATE
((((R,*,(X,**,3)),*,Y),+,Z),/,(((R,-,Y),-,X),*,(X,+,Y))),X)

(LAMBDA,(L),(ENGLISH,(ZEROBEGONE,(DETERMINANT,L))))
((((-,A,L),1,0,0),(1,(-,A,L),1,0),(0,1,(-,A,L),1),(0,0,1,(-,AL,L))))

STOP

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((DIFF,(LAMBDA,(L,X),(COND,((ATOM,L),(COND,((EQUAL,L,X),(QUOTE,1)),(T,(QUOTE,0)))),((TRIPLET,L),(COND,((EQUAL,(CAR,L),
(QUOTE,+)),(LIST,(QUOTE,+),(DIFF,(CADR,L),X),(DIFF,(CADDR,L),X))),((EQUAL,(CAR,L),(QUOTE,-)),(LIST,(QUOTE,-),(DIFF,(CAD
R,L),X),(DIFF,(CADDR,L),X))),((EQUAL,(CAR,L),(QUOTE,*)),(LIST,(QUOTE,+),(LIST,(QUOTE,*),(CADR,L),(DIFF,(CADDR,L),X)),(L
IST,(QUOTE,*),(DIFF,(CADR,L),X),(CADDR,L)))),((EQUAL,(CAR,L),(QUOTE,/)),(LIST,(QUOTE,/),(LIST,(QUOTE,-),(LIST,(QUOTE,*)
,(DIFF,(CADR,L),X),(CADDR,L)),(LIST,(QUOTE,*),(CADR,L),(DIFF,(CADDR,L),X))),(LIST,(QUOTE,**),(CADDR,L),(QUOTE,2)))),((E
QUAL,(CAR,L),(QUOTE,**)),(LIST,(QUOTE,*),(CADDR,L),(LIST,(QUOTE,**),(CADR,L),(LIST,(QUOTE,-),(CADDR,L),(QUOTE,1))))),(T
,(DIFF2,L,X)))),((DOUBLET,L),(COND,((EQUAL,(CAR,L),(QUOTE,-)),(LIST,(QUOTE,-),(DIFF,(CADR,L),X))),(T,(DIFF1,L,X)))),(T,
(DIFF2,L,X))))),(DIFF2,(LAMBDA,(L,X),(LIST,(QUOTE,ILLEGAL,FORMAT),L))),(DIFFERENTIATE,(LAMBDA,(L,X),(ENGLISH,(ZEROBEGON
E,(DIFF,(POLISH,L),X)))))))


END OF APPLY, VALUE IS ...
(DIFF,DIFF2,DIFFERENTIATE)



 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((DIFF0,(LAMBDA,(L,X,DERFUN),(LIST,(QUOTE,*),DERFUN,(DIFF,(CADR,L),X)))),(DIFF1,(LAMBDA,(L,X),(COND,((EQUAL,(CAR,L),(Q
UOTE,EXP)),(DIFF0,L,X,(LIST,(QUOTE,EXP),(CADR,L)))),((EQUAL,(CAR,L),(QUOTE,SIN)),(DIFF0,L,X,(LIST,(QUOTE,COS),(CADR,L))
)),((EQUAL,(CAR,L),(QUOTE,COS)),(DIFF0,L,X,(LIST,(QUOTE,-),(LIST,(QUOTE,SIN),(CADR,L))))),((EQUAL,(CAR,L),(QUOTE,LN)),(
DIFF0,L,X,(LIST,(QUOTE,/),(QUOTE,1),(CADR,L)))),(T,(DIFF0,L,X,(LIST,(LIST,(QUOTE,DERIVATIVE),(CAR,L)),(CADR,L)))))))))


END OF APPLY, VALUE IS ...
(DIFF0,DIFF1)



 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((ZEROBEGONE,(LAMBDA,(L),(COND,((EQUAL,(MINUSGO,(EXPGO,(ZEROGO,L))),L),L),(T,(ZEROBEGONE,(MINUSGO,(EXPGO,(ZEROGO,L))))
))))))


END OF APPLY, VALUE IS ...
(ZEROBEGONE)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((ZEROGO2,(LAMBDA,(L),(LIST,(CAR,L),(ZEROGO,(CADR,L)),(ZEROGO,(CADDR,L))))),(ZEROGO1,(LAMBDA,(L),(LIST,(CAR,L),(ZEROGO
,(CADR,L))))),(ZEROGO,(LAMBDA,(L),(COND,((TRIPLET,L),(COND,((EQUAL,(CAR,L),(QUOTE,*)),(COND,((EQUAL,(CADR,L),(QUOTE,0))
,(QUOTE,0)),((EQUAL,(CADDR,L),(QUOTE,0)),(QUOTE,0)),((EQUAL,(CADR,L),(QUOTE,1)),(ZEROGO,(CADDR,L))),((EQUAL,(CADDR,L),(
QUOTE,1)),(ZEROGO,(CADR,L))),(T,(ZEROGO2,L)))),((EQUAL,(CAR,L),(QUOTE,/)),(COND,((EQUAL,(CADR,L),(QUOTE,0)),(QUOTE,0)),
((EQUAL,(CADDR,L),(QUOTE,1)),(ZEROGO,(CADR,L))),(T,(ZEROGO2,L)))),((EQUAL,(CAR,L),(QUOTE,+)),(COND,((EQUAL,(CADR,L),(QU
OTE,0)),(ZEROGO,(CADDR,L))),((EQUAL,(CADDR,L),(QUOTE,0)),(ZEROGO,(CADR,L))),(T,(ZEROGO2,L)))),((EQUAL,(CAR,L),(QUOTE,-)
),(COND,((EQUAL,(CADR,L),(QUOTE,0)),(LIST,(QUOTE,-),(ZEROGO,(CADDR,L)))),((EQUAL,(CADDR,L),(QUOTE,0)),(ZEROGO,(CADR,L))
),(T,(ZEROGO2,L)))),(T,(ZEROGO2,L)))),((DOUBLET,L),(COND,((AND,(EQUAL,(CAR,L),(QUOTE,-)),(EQUAL,(CADR,L),(QUOTE,0))),(Q
UOTE,0)),(T,(ZEROGO1,L)))),(T,L))))))


END OF APPLY, VALUE IS ...
(ZEROGO2,ZEROGO1,ZEROGO)




 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((MINUSGOL,(LAMBDA,(L,OP),(LIST,OP,(MINUSGO,(CADADR,L)),(MINUSGO,(CADDR,L))))),(MINUSGOR,(LAMBDA,(L,OP),(LIST,OP,(MINU
SGO,(CADR,L)),(MINUSGO,(CAR,(CDADDR,L)))))),(MINUSGO2,(LAMBDA,(L),(LIST,(CAR,L),(MINUSGO,(CADR,L)),(MINUSGO,(CADDR,L)))
)),(MINUSGO1,(LAMBDA,(L),(LIST,(CAR,L),(MINUSGO,(CADR,L))))),(NEGATIVE,(LAMBDA,(L),(COND,((EQUAL,L,(QUOTE,+)),(QUOTE,-)
),((EQUAL,L,(QUOTE,-)),(QUOTE,+)),(T,L)))),(MINUSGO,(LAMBDA,(L),(COND,((TRIPLET,L),(COND,((OR,(EQUAL,(CAR,L),(QUOTE,*))
,(EQUAL,(CAR,L),(QUOTE,/))),(COND,((AND,(DOUBLET,(CADR,L)),(EQUAL,(CAADR,L),(QUOTE,-))),(LIST,(QUOTE,-),(MINUSGOL,L,(CA
R,L)))),((AND,(DOUBLET,(CADDR,L)),(EQUAL,(CAADDR,L),(QUOTE,-))),(LIST,(QUOTE,-),(MINUSGOR,L,(CAR,L)))),(T,(MINUSGO2,L))
)),((OR,(EQUAL,(CAR,L),(QUOTE,+)),(EQUAL,(CAR,L),(QUOTE,-))),(COND,((AND,(DOUBLET,(CADR,L)),(EQUAL,(CAADR,L),(QUOTE,-))
),(LIST,(QUOTE,-),(MINUSGOL,L,(NEGATIVE,(CAR,L))))),((AND,(DOUBLET,(CADDR,L)),(EQUAL,(CAADDR,L),(QUOTE,-))),(MINUSGOR,L
,(NEGATIVE,(CAR,L)))),(T,(MINUSGO2,L)))),((AND,(EQUAL,(CAR,L),(QUOTE,**)),(DOUBLET,(CADDR,L)),(EQUAL,(CAADDR,L),(QUOTE,
-))),(LIST,(QUOTE,/),(QUOTE,1),(MINUSGOR,L,(CAR,L)))),(T,(MINUSGO2,L)))),((DOUBLET,L),(COND,((AND,(EQUAL,(CAR,L),(QUOTE
,-)),(DOUBLET,(CADR,L)),(EQUAL,(CAADR,L),(QUOTE,-))),(MINUSGO,(CADADR,L))),(T,(MINUSGO1,L)))),(T,L))))))


END OF APPLY, VALUE IS ...
(MINUSGOL,MINUSGOR,MINUSGO2,MINUSGO1,NEGATIVE,MINUSGO)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((EXPGO,(LAMBDA,(L),(COND,((TRIPLET,L),(COND,((AND,(EQUAL,(CAR,L),(QUOTE,-)),(EQUAL,(CADDR,L),(QUOTE,1))),(COND,((EQUA
L,(CADR,L),(QUOTE,10)),(QUOTE,9)),((EQUAL,(CADR,L),(QUOTE,9)),(QUOTE,8)),((EQUAL,(CADR,L),(QUOTE,8)),(QUOTE,7)),((EQUAL
,(CADR,L),(QUOTE,7)),(QUOTE,6)),((EQUAL,(CADR,L),(QUOTE,6)),(QUOTE,5)),((EQUAL,(CADR,L),(QUOTE,5)),(QUOTE,4)),((EQUAL,(
CADR,L),(QUOTE,4)),(QUOTE,3)),((EQUAL,(CADR,L),(QUOTE,3)),(QUOTE,2)),((EQUAL,(CADR,L),(QUOTE,2)),(QUOTE,1)),((EQUAL,(CA
DR,L),(QUOTE,1)),(QUOTE,0)),((EQUAL,(CADR,L),(QUOTE,0)),(QUOTE,(-,1))),(T,(CONS,(CAR,L),(CONS,(EXPGO,(CADR,L)),(CDDR,L)
))))),((EQUAL,(CAR,L),(QUOTE,**)),(COND,((EQUAL,(CADR,L),(QUOTE,1)),(QUOTE,1)),((EQUAL,(CADDR,L),(QUOTE,1)),(EXPGO,(CAD
R,L))),((EQUAL,(CADR,L),(QUOTE,0)),(QUOTE,0)),((EQUAL,(CADDR,L),(QUOTE,0)),(QUOTE,1)),(T,(EXPGO2,L)))),(T,(EXPGO2,L))))
,((DOUBLET,L),(LIST,(CAR,L),(EXPGO,(CADR,L)))),(T,L)))),(EXPGO2,(LAMBDA,(L),(LIST,(CAR,L),(EXPGO,(CADR,L)),(EXPGO,(CADD
R,L)))))))


END OF APPLY, VALUE IS ...
(EXPGO,EXPGO2)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((X,-,Y),X)


END OF APPLY, VALUE IS ...
1
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((X,-,Y),Y)


END OF APPLY, VALUE IS ...
(-,1)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((X,+,Y),X)


END OF APPLY, VALUE IS ...
1
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((X,*,Y),X)


END OF APPLY, VALUE IS ...
Y
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((X,/,Y),Y)


END OF APPLY, VALUE IS ...
(-,(X,/,(Y,**,2)))
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((X,**,3),X)


END OF APPLY, VALUE IS ...
(3,*,(X,**,2))
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((SIN,(X*,*,Y)),Y)


END OF APPLY, VALUE IS ...
((COS,(X*,*,Y)),*,X*)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
((QUXL,X),X)


END OF APPLY, VALUE IS ...
((DERIVATIVE,QUXL),X)
```

```
 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
(((((R,*,(X,**,3)),*,Y),+,Z),/,(((R,-,Y),-,X),*,(X,+,Y))),X)


GC. STAT.    ENT. TIME           MARK TIME           EXIT TIME           PDL DEPTH=   101 IR4 ON PDL=    29    GARBAGE= 13305

GC. STAT.    ENT. TIME           MARK TIME  000.0    EXIT TIME  000.0    PDL DEPTH=    59 IR4 ON PDL=    17    GARBAGE= 13309

GC. STAT.    ENT. TIME  000.0    MARK TIME           EXIT TIME           PDL DEPTH=    90 IR4 ON PDL=    26    GARBAGE= 13239

GC. STAT.    ENT. TIME           MARK TIME           EXIT TIME  000.0    PDL DEPTH=    79 IR4 ON PDL=    23    GARBAGE= 13237

GC. STAT.    ENT. TIME  000.0    MARK TIME  000.0    EXIT TIME           PDL DEPTH=    80 IR4 ON PDL=    23    GARBAGE= 13048

END OF APPLY, VALUE IS ...
(((((R,*,(3,*,(X,**,2))),*,Y),*,(((R,-,Y),-,X),*,(X,+,Y))),-,((((R,*,(X,**,3)),*,Y),+,Z),*,(((R,-,Y),-,X),-,(X,+,Y)))),
/,((((R,-,Y),-,X),*,(X,+,Y)),**,2))


 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DIFFERENTIATE
(((+,+,+,+,+,+),(++),(++++),(+)),+)

END OF APPLY, VALUE IS ...
(ILLEGAL,((+,+,+,+,+,+),(++),(++++),(+)))
```

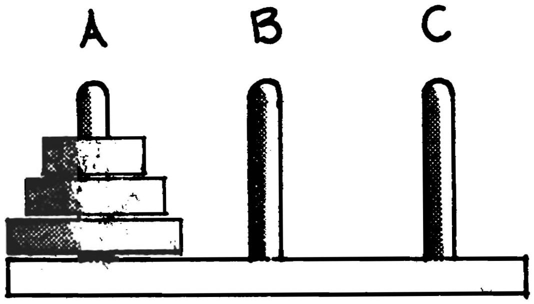
THE TOWERS OF HANOI

The Tower of Hanoi is a classical puzzle in which one has a set of discs, whose radii are progressively larger, and having holes in the center by which they may be stacked onto a peg. Given three pegs, A, B, C, with all the discs stacked on one of them,A, say according to their increasing size, the object of the puzzle is to transfer all the discs to peg B. However, only one may be moved at a time, but it may rest at any peg, so long as a larger disc is never placed above a smaller disc.

Although it would be a more challenging problem to design a LISP function which would seek out a means of solution, we shall content ourselves with an analysis of the solution, and describe a LISP function which will effect it.

The problem is rather easily solved for a small number n of discs. For n=1, the disc is moved directly from peg A to peg B. For n=2, the small disc is stored on peg C, permitting the large one to move to peg B, and the small disc is then placed on top of it. For n=3, the sequence of moves is

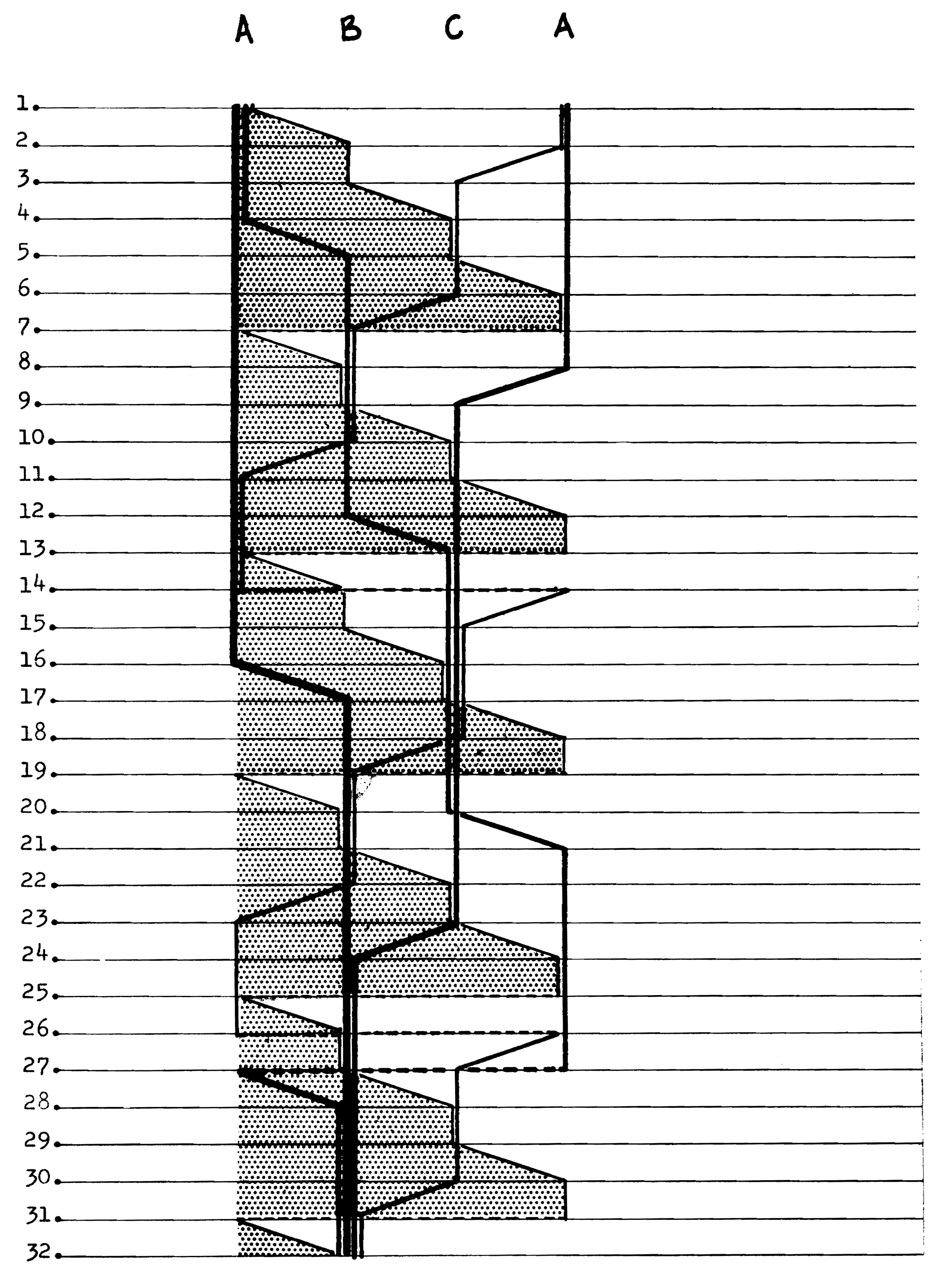
small	→	B
middle	→	C
small	→	C
large	→	B
small	→	A
middle	→	B
small	→	B



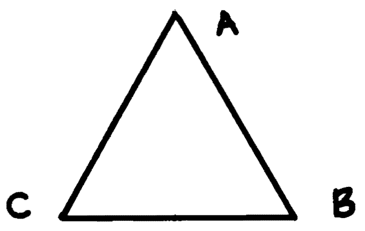
However, for larger n, the problem deserves more careful thought. Let us suppose that there is a solution to the problem, and that we have progressed to some stage in this solution. There will be at most three disces exposed on the top of any pile, and of these one will be larger than the other two, and hence cannot be moved without violating the rule. Of the two remaining the larger can move to the top of the largest disc, while the smaller has free choice of the other two pegs. Thus there are never more than three moves available.

If the smallest disc was moved on the previous turn, there is no need to move it again, for otherwise one turn could be saved by moving it directly to its final resting place. In this way we see that the small and the large disc must alternate turns, for otherwise the moves would undo the move of the previous turn.

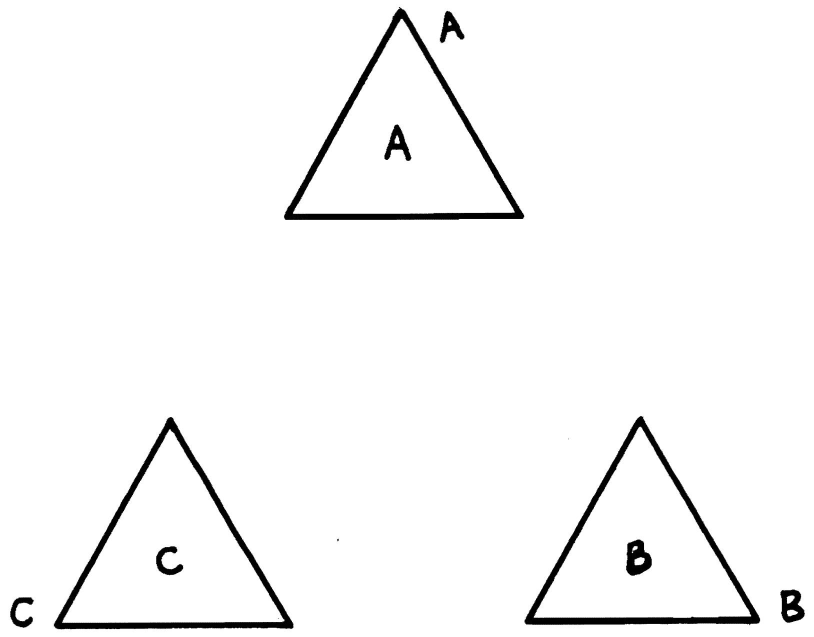
In fact, we see that if we regard the three pegs A, B, C, as forming a circle, the smallest disc will consistently move in one direction, say counterclockwise, every other turn. In a similar fashion the second disc will always move clockwise one place each fourth turn, the third disc will move counterclockwise every eighth turn, and so on. We may then summarize the solution in a chart - a sort of braid diagram in which the turns are shown vertically and the pegs horizontally. A line for each disc informs us of its location at each turn:


A
B
C
A
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.
28.
29.
30.
31.
32.

A somewhat different series of diagrams may be used to show all the possible states of the puzzle and not just its most direct line of solution. For one disc, we draw a triangle:

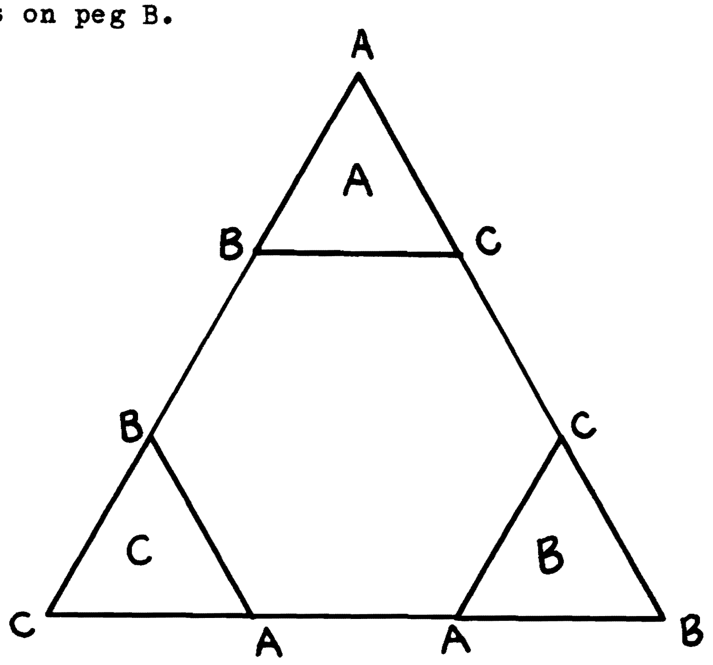


The vertices represent peg which may contain the disc, while the edges represent legal moves. For two discs, the problem reduces to that of one disc when the large disc is not moved, and thus we set up three triangles to represent the states of the small

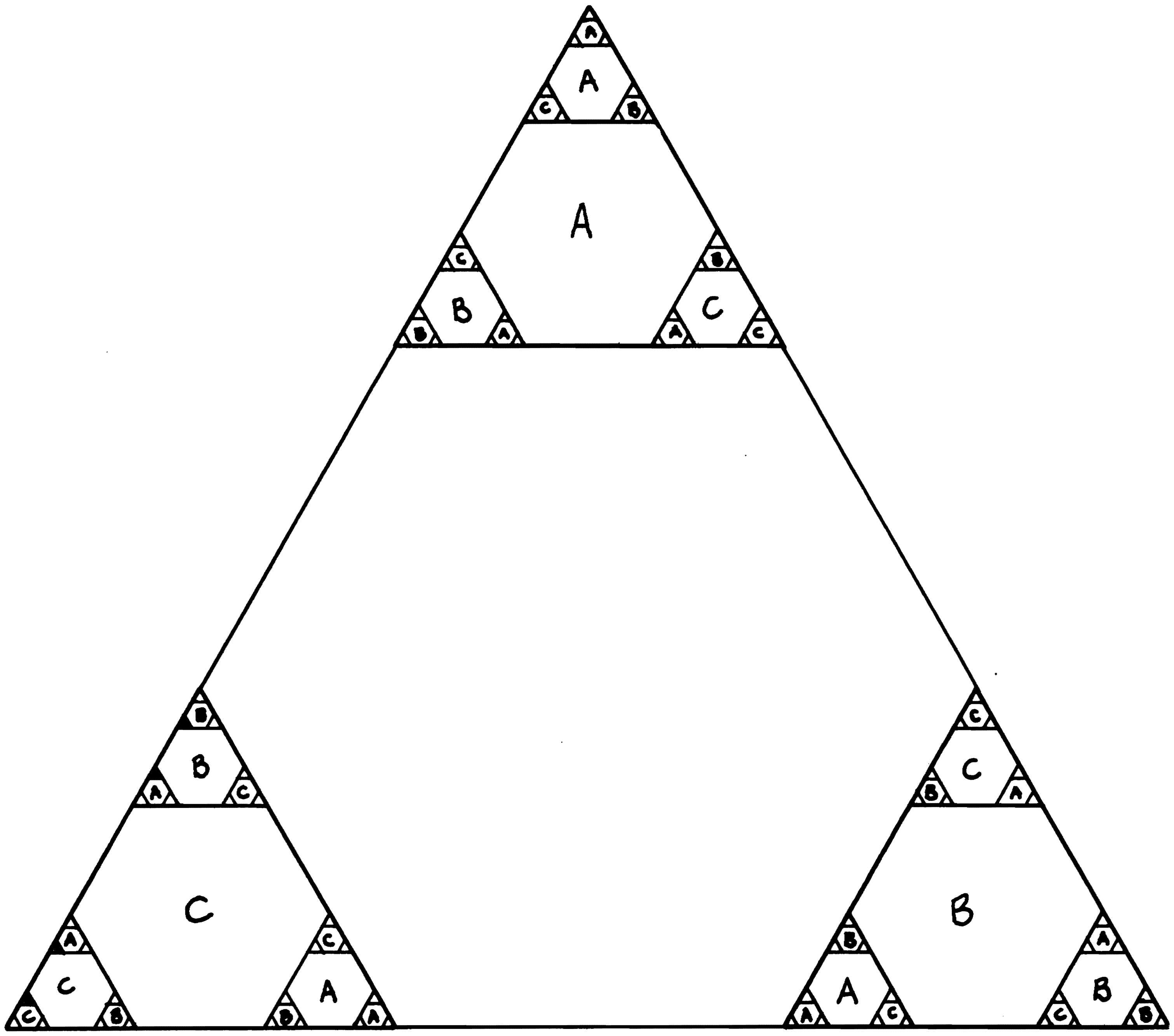


disc when the larger is fixed. We name the triangle by the location of the larger disc, the vertices by the location of the smaller disc.

We may connect these triangles to show the legal moves possible for the large disc. For example, the large disc may move from peg A to peg C when the smaller is on peg B.



The process may now repeat, as we add more and more discs to the problem. In each case we take three diagrams for n-1 discs. When all but the largest discs are on any one peg, the largest disc can move between the remaining pegs, which shows how the three diagrams may be joined at their corners.


A
A
C
B
B
A
C
B
A
C
B
C
A
B
C
A
C
B
A
B
C
C
A
B
C
A
C
B
C
A
A
B
A
C
B
A
C
B
B

The Tower of Hanoi is solved by the LISP functions HANOI AND HANOI*, together with certain of their satellites. The satellites are used to determine which peg the smallest and next smallest occupy, as well as whether the move is odd or even and whether the problem is solved. HANOI AND HANOI* themselves cause the alternate moves of the smallest and intermediate disc, and the function is recursive because HANOI is defined in terms of HANOI* of suitably rearranged arguments, and conversely. The recursion terminates when HANOI* has an acceptable set of arguments, and the steps of the solution may be presented by TRACLIS. The puzzle itself is represented as a list of three lists, each of which enumerate the discs on one of the three pegs.

HAROLD V. McINTOSH

```
APPLY OPERATOR AS OF FEBRUARY 10, 1960

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

READ IN LISTS ...
DEFINE
(((ORDER,(LAMBDA,(X,Y,L),(COND,((NULL,L),F),((EQUAL,X,(CAR,L)),T),((EQUAL,Y,(CAR,L)),F),(T,(ORDER,X,Y,(CDR,L)))))),(EXP
UNGE,(LAMBDA,(I,L),(COND,((NULL,L),NIL),((EQUAL,I,(CAR,L)),(EXPUNGE,I,(CDR,L))),(T,(CONS,(CAR,L),(EXPUNGE,I,(CDR,L)))))
)),(ACCUMULATE,(LAMBDA,(L,M),(COND,((NULL,L),M),(T,(ACCUMULATE,(EXPUNGE,(CAR,L),(CDR,L)),(CONS,(CAR,L),(EXPUNGE,(CAR,L)
,M))))))),(ELEMENT,(LAMBDA,(X,S),(COND,((NULL,S),F),((EQUAL,X,(CAR,S)),T),(T,(ELEMENT,X,(CDR,S)))))),(COLLECT,(LAMBDA,(
L),(COND,((NULL,L),NIL),((NULL,(CDR,L)),L),((ELEMENT,(CAR,L),(CDR,L)),(CONS,(CAR,L),(COLLECT,(CONS,(CAR,L),(REMOVE,(CAR
,L),(CDR,L)))))),(T,(CONS,(CAR,L),(COLLECT,(CDR,L))))))),(REMOVE,(LAMBDA,(X,L),(COND,((NULL,L),NIL),((EQUAL,X,(CAR,L)),
(CDR,L)),(T,(CONS,(CAR,L),(REMOVE,X,(CDR,L)))))))))

DEFINE
(((TERMINAL,(LAMBDA,(L),(COND,((OR,(AND,(NULL,(CAR,L)),(NULL,(CADR,L))),(AND,(NULL,(CADR,L)),(NULL,(CADDR,L))),(AND,(NU
LL,(CADDR,L)),(NULL,(CAR,L)))),T),(T,F)))),(MEDIAN,(LAMBDA,(A,B,C,M),(COND,((NULL,A),(COND,((ORDER,B,C,M),B),(T,C))),((
NULL,B),(COND,((ORDER,C,A,M),C),(T,A))),((NULL,C),(COND,((ORDER,A,B,M),A),(T,B))),((ORDER,A,B,M),(COND,((ORDER,B,C,M),B
),((ORDER,A,C,M),C),(T,A))),(T,(COND,((ORDER,A,C,M),A),((ORDER,B,C,M),C),(T,B))))))),(ODD,(LAMBDA,(X,M),(COND,((EQUAL,X,
(CAR,M)),T),(T,(NOT,(ODD,X,(CDR,M))))))),(STACK,(LAMBDA,(L),(COND,((NOT,(NULL,(CAR,L))),(CAR,L)),((NOT,(NULL,(CADR,L)))
,(CADR,L)),(T,(CADDR,L)))))),(HANOI,(LAMBDA,(L),(COND,((EQUAL,(CAR,(STACK,Q)),(CAAR,L)),(HANOI*,(LIST,(CDAR,L),(CADR,L),
(CONS,(CAAR,L),(CADDR,L))))),((EQUAL,(CAR,(STACK,Q)),(CAADR,L)),(HANOI*,(LIST,(CONS,(CAADR,L),(CAR,L)),(CDADR,L),(CADDR
,L)))),((EQUAL,(CAR,(STACK,Q)),(CAADDR,L)),(HANOI*,(LIST,(CAR,L),(CONS,(CAADDR,L),(CADR,L)),(CDADDR,L)))))))),(HANOI*,(L
AMBDA,(L),(COND,((TERMINAL,L),L),((EQUAL,(MEDIAN,(CAAR,L),(CAADR,L),(CAADDR,L),(STACK,Q)),(CAAR,L)),(COND,((ODD,(CAAR,L
),(STACK,Q)),(HANOI,(LIST,(CDAR,L),(CADR,L),(CONS,(CAAR,L),(CADDR,L))))),(T,(HANOI,(LIST,(CDAR,L),(CONS,(CAAR,L),(CADR,
   ),(CADDR,L)))))),((EQUAL,(MEDIAN,(CAAR,L),(CAADR,L),(CAADDR,L),(STACK,Q)),(CAADR,L)),(COND,((ODD,(CAADR,L),(STACK,Q))
,(HANOI,(LIST,(CONS,(CAADR,L),(CAR,L)),(CDADR,L),(CADDR,L)))),(T,(HANOI,(LIST,(CAR,L),(CDADR,L),(CONS,(CAADR,L),(CADDR,
L)))))))),((EQUAL,(MEDIAN,(CAAR,L),(CAADR,L),(CAADDR,L),(STACK,Q)),(CAADDR,L)),(COND,((ODD,(CAADDR,L),(STACK,Q)),(HANOI,
(LIST,(CAR,L),(CONS,(CAADDR,L),(CADR,L)),(CDADDR,L)))),(T,(HANOI,(LIST,(CONS,(CAADDR,L),(CAR,L)),(CADR,L),(CDADDR,L))))
)))))))

TRACLIS
((HANOI,HANOI*))

HANOI
(( , ,(A,B,C,D,E)))
((Q,( , ,(A,B,C,D,E))))




STOP

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...
```

```
FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
HANOI
(( , ,(A,B,C,D,E)))
((Q,( , ,(A,B,C,D,E))))
```

```
(ARGUMENTS,OF,HANOI)
( , ,(A,B,C,D,E))

(ARGUMENTS,OF,HANOI*)
( ,(A),(B,C,D,E))

(ARGUMENTS,OF,HANOI)
((B),(A),(C,D,E))

(ARGUMENTS,OF,HANOI*)
((A,B), ,(C,D,E))

(ARGUMENTS,OF,HANOI)
((A,B),(C),(D,E))

(ARGUMENTS,OF,HANOI*)
((B),(C),(A,D,E))

(ARGUMENTS,OF,HANOI)
( ,(B,C),(A,D,E))

(ARGUMENTS,OF,HANOI*)
( ,(A,B,C),(D,E))

(ARGUMENTS,OF,HANOI)
((D),(A,B,C),(E))

(ARGUMENTS,OF,HANOI*)
((A,D),(B,C),(E))

(ARGUMENTS,OF,HANOI)
((A,D),(C),(B,E))

(ARGUMENTS,OF,HANOI*)
((D),(C),(A,B,E))

(ARGUMENTS,OF,HANOI)
((C,D), ,(A,B,E))

(ARGUMENTS,OF,HANOI*)
((C,D),(A),(B,E))

(ARGUMENTS,OF,HANOI)
((B,C,D),(A),(E))

(ARGUMENTS;OF,HANOI*)
((A,B,C,D), ,(E))

(ARGUMENTS,OF,HANOI)
((A,B,C,D),(E), )
```

```
(ARGUMENTS,OF,HANOI*)
((B,C,D),(E),(A))

(ARGUMENTS,OF,HANOI)
((C,D),(B,E),(A))

(ARGUMENTS,OF,HANOI*)
((C,D),(A,B,E), )

(ARGUMENTS,OF,HANOI)
((D),(A,B,E),(C))

(ARGUMENTS,OF,HANOI*)
((A,D),(B,E),(C))

(ARGUMENTS,OF,HANOI)
((A,D),(E),(B,C))

(ARGUMENTS,OF,HANOI*)
((D),(E),(A,B,C))

(ARGUMENTS,OF,HANOI)
( ,(D,E),(A,B,C))

(ARGUMENTS,OF,HANOI*)
( ,(A,D,E),(B,C))

(ARGUMENTS,OF,HANOI)
((B),(A,D,E),(C))

(ARGUMENTS,OF,HANOI*)
((A,B),(D,E),(C))

(ARGUMENTS,OF,HANOI)
((A,B),(C,D,E), )

(ARGUMENTS,OF,HANOI*)
((B),(C,D,E),(A))

(ARGUMENTS,OF,HANOI)
( ,(B,C,D,E),(A))

(ARGUMENTS,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )
```

```
(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )
```

```
(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI*)
( ,(A,B,C,D,E), )

(VALUE,OF,HANOI)
( ,(A,B,C,D,E), )

END OF APPLY, VALUE IS ..
( ,(A,B,C,D,E), )
```

COFLEXLIST

A flexagon may for our purposes be considered as a list of length two; a special case of a pat, which is a list of length two or greater or an atomic symbol. Lists of length zero can not be handled by COFLEXLIST; list of length one are reduced to atoms during the course of operation; and are therefore more consistently not used as pats. A typical flexagon then has the structure (A, B) and a typical pat the structure A or $(B_1,\ldots,B_n)$, where A and B are pats (lists or atoms).

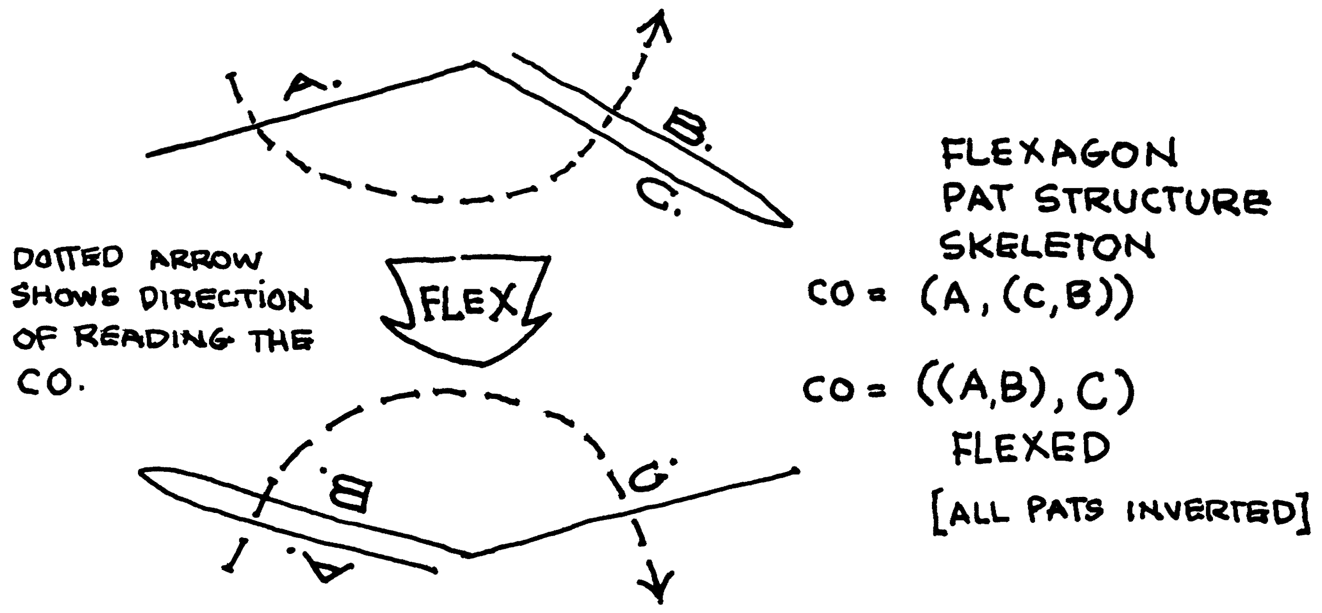
We define three distinct operations which are performed upon flexagons, called ROTATE, FLEXR, and FLEXL. These operations are essentially characterized by:

ROTATE	:	(A, B)	$\rightarrow$	(B, A)
FLEXR	:	$(A, (B_1, B_2,\ldots,B_n))$	$\rightarrow$	$((B_2,\ldots B_n,A), B_1)$
FLEXL	:	$(A, (B_1,\ldots,B_{n-1}, B_n))$	$\rightarrow$	$((A_1,\ldots,B_{n-1})\ B_n)$.

These operations all maintain a constant cyclic order of the atomic symbols (which are called "leaves" in flexagons) of which the flexagon is ultimately composed, independent of the list groupings indicated by parentheses. The representation of the flexagon structure is consequently called the flexagon's constant order (CO).

The convenience which accompanies the constant order representation is somewhat offset by the fact that it introduces an ambiguity in the order of the sides showing in the flexagon.

When the flexing process is examined in the actual flexagon, it is seen that each flex reverses the direction of the constant order relative to the left-hand pat, as indicated by the following diagram:



A.
B.
C.
FLEX
DOTTED ARROW SHOWS DIRECTION OF READING THE CO.
FLEXAGON PAT STRUCTURE SKELETON
CO = (A, (C, B))
CO = ((A, B), C)
FLEXED
[ALL PATS INVERTED]

The direction of reading the CO will clearly also be reversed by a rotation as well.

It can be seen that FLEXR and FLEXL are dual in this same respect; one counts the leaves in one direction, the other in the other direction. Essentially, the parity involved is between reading <u>up</u> or <u>down</u> through the first of the two given pats. This parity is reversed each time the flexagon is flexed or rotated, and in the program COFLEXLIST is recorded by an auxiliary atomic symbol, T or F, which is paired with the current flexagon structure. The initial value is given as that corresponding to FLEXR. As FLEXR is performed, the parity indication is changed to that for FLEXL, etc. The coordination of the three operations is handled by the operation FLEX. It will be noted that FLEXR and FLEXL always move as many subpats as possible away from the second pat; this type of flexing opens up the way to consequent flexing deeper within the flexagon subpat structure, so to speak; that is, B_n will always immediately become a pat in its own right. This type of flexing has been called O-cut flexing or following O-cut cycles. It is also a part of the operation of the generalized Tuckerman traverse, which exhibits all of the flexagons O-cut faces with maximum efficiency and gives significant insight into the problem of building paper flexagons. It is this traverse of a flexagon which is given to be COFLEXLIST. To perform the traverse, we simply flex whenever possible, then rotate if flexing is impossible. The function FLEX prescribes this order of events, as well as making parity changes.

This program CO-FLEX is a LIST of constant-order-represented flexagons. The list is surrounded by an additional pair of parenthesis and followed by two parenthesis, so: ( ), which complete the apply triplet for COFLEXLIST. The function COFLEXLIST itself only checks to see if any flexagons remain to be flexed, and adds each finished flexagon. When all the flexagons have been flexed, COFLEXLIST terminates its list with the word "FINISHED".

The function OPERATE stores the given initial value of the CO for each flexagon and checks after each operation to see whether the new CO structure is the same as the first. If so, the traverse has been completed and OPERATE gives as its value the function FAKE, whose value is the words "NEXT, FLEXAGON", listed with the last CO structure.

The functions LEFT and RIGHT compute $(B_1,\ldots,B_{n-1})$ and (B_n), respectively. The value of each function is a list, splitting the old list just before its last term.

FLEXR and FLEXL actually do somewhat more than we have indicated; they list the new CO structures computed along with a list of the flexagon sides actually showing at the beginning of the operation just completed. The current sides showing are not indicated, since there can scarcely be much ambiguity here and to give them would require undue complexity. The sides shown are the value of the function SIDES, which takes the first leaves to the left of each pat, which are given by the function 1E, and orders them according to the current parity indication. The sides showing are then assumed to be given by these first leaves; that is, each leaf is assumed to bear the name which, in the actual paper flexagon, would be carried by what is, in the CO, the left-hand surface of that leaf as we view the CO on the page. At any rate, this gives a quick characterization of the faces which are revealed.

As far as output, we receive the final value of the function COFLEXLIST at the program's end. Furthermore, we trace (using TRACLIS) the values of the function's FAKE, ROTATE, FLEXR, and FLEXL, to indicate what happens when flexagons are changed and within each flexagon. Each of these functions includes in both argument and value the current CO structure. Furthermore, arguments of all four and the value of FAKE end with the current parity value, though this is printed out as a blank space, since it is indicated by one of the special characters T and F. Values of FLEXR, FLEXL, and ROTATE end with the list of sides showing at their commencement, in order from top to bottom.

ANTHONY CONRAD

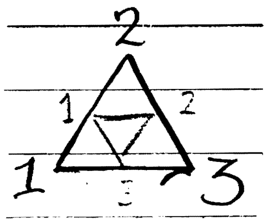
```
START TIME  17.41 DATE  07/05/61 ID CARD *      CHK              01-128  7550-000-27350          MCINTOSH
        REM      A CONRAD
        REM       PLACE LISP BINARY TAPE ON TAPE UNIT B-9
        REM      SCRATCH TAPE ON TAPE UNIT B-8
        REM     OUTPUT MAY BE MONITORED BY DEPRESSING SWITCH 3.
        REM      NORMAL HALT    OCTAL 1300
        REM     NOTE... DONT INCLUDE REMARK CARDS IN LISP SECTION OF DECK.
        REM     THE FLEXAGONS ARE ENTERED IN A LIST AT THE END OF THE PROG
        REM     RAM DECK IN CONSTANT ORDER FORM, WITH PARENTHESES INDICATI
        REM     NG THE THUMBHOLE STRUCTURE. THIS LIST IS SURROUNDED BY A
        REM     PAIR OF ADDITIONAL PARENTHESES AND FOLLOWED BY AN EXTRA
        REM     PAIR, THUS... (). LEAVES MAY BE REPRESENTED BY ANY ATOMIC
        REM     SYMBOLS. EACH WILL BE INTERPRETED AS THE LEFTHAND SIDE OF
        REM     THAT LEAF AS IT APPEARS IN THE CONSTANT ORDER. INDIVIDUAL
        REM     LEAVES ARE NOT SURROUNDED BY PARENTHESES.  OUTPUT FORMAT..
        REM     (CURRENT C.O.STRUCTURE , SIDES PREVIOUSLY SHOWING ).
```

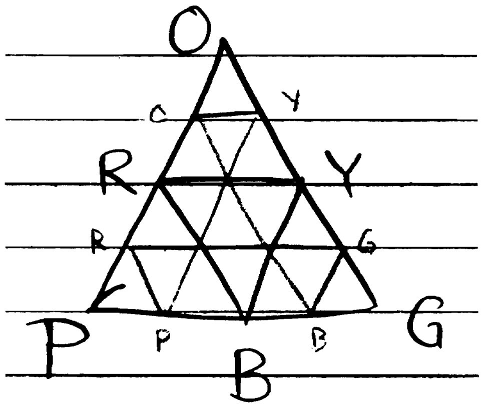
```
APPLY OPERATOR AS OF FEBRUARY 10, 1960

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

READ IN LISTS ...
DEFINE
(((LEFT,(LAMBDA,(L),(COND,((NULL,L),NIL),((ATOM,L),NIL),((NULL,(CDR,L)),NIL),(T,(CONS,(CAR,L),(LEFT,(CDR,L))))))),(RIGH
T,(LAMBDA,(L),(COND,((NULL,L),L),((ATOM,L),L),((NULL,(CDR,L)),L),(T,(RIGHT,(CDR,L)))))),(SIDES,(LAMBDA,(X),(COND,((CADR
,X),(LIST,(1E,(CADAR,X)),(1E,X))),(T,(LIST,(1E,X),(1E,(CADAR,X))))))),(ROTATE,(LAMBDA,(X),(LIST,(LIST,(CADAR,X),(CAAR,X
)),(SIDES,X)))),(FLEXR,(LAMBDA,(X),(LIST,(LIST,(APPEND,(CDADAR,X),(LIST,(CAAR,X))),(CAADAR,X)),(SIDES,X)))),(FLEXL,(LAM
BDA,(X),(LIST,(LIST,(CONS,(CAAR,X),(LEFT,(CADAR,X))),(CAR,(RIGHT,(CADAR,X)))),(SIDES,X)))),(FLEX,(LAMBDA,(X),(COND,((AT
OM,(CADAR,X)),(LIST,(CAR,(ROTATE,X)),(COND,((CADR,X),F),(T,T)))),((CADR,X),(LIST,(CAR,(FLEXR,X)),F)),(T,(LIST,(CAR,(FLE
XL,X)),T))))),(FAKE,(LAMBDA,(X),(LIST,X,(QUOTE,NEXT),(QUOTE,FLEXAGON)))),(OPERATE,(LAMBDA,(CO,STO),(COND,((EQUAL,CO,(CA
R,STO)),(FAKE,STO)),(T,(OPERATE,CO,(FLEX,STO)))))),(1E,(LAMBDA,(X),(COND,((ATOM,X),X),(T,(1E,(CAR,X)))))),(COFLEXLIST,(
LAMBDA,(L),(COND,((NULL,L),(QUOTE,FINISHED)),(T,(LIST,(OPERATE,(CAR,L),(FLEX,(LIST,(CAR,L),T))),(COFLEXLIST,(CDR,L)))))
))))

TRACLIS
((FLEXR,FLEXL,ROTATE,FAKE))

COFLEXLIST
((((1,2),3),(R,(P,((B,G),(Y,0)))),(((7,6),(9,8)),(5,((2,1),(4,3)))),((A,B,C),((T,U,V),(G,((N,O,(R,Q,P)),M,(H,(K,J,I),L)
),S),(D,E,F))),(((1,2,3,4,5,6,7),26,(13,14,15,16,(23,22,21,20,19,18,17),24,25),12,11,10,9),8),((1,2,3,4),((14,((21,22),
(16,17,18,19,20),15)),(7,8,((11,12,13),(9,10))),(5,6))),((2,AAARGH),(3,((HELP,1),(1,HELP),(2,HELP))))))

STOP

 AT THE TONE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...
```

```
OBJECT LIST NOW IS ...
(HELP,AAARGH,25,24,17,19,20,21,22,23,16,26,E,I,K,H,Q,C,G,FINISHED,COFLEXLIST,CO,OPERATE,FLEXAGON,NEXT,FAKE,FLEX,FLEXL,F
LEXR,ROTATE,1E,SIDES,RIGHT,LEFT,PF,VAR,FORM,SEARCHF,T1,UF,FF,DF,CCMPSRCH,BNDV,SELECT,CSETQ,CONC,CDDDDR,CDDDAR,CDDADR,CD
DAAR,CDADDR,CDADAR,CDAADR,CDAAAR,CADDDR,CADDAR,CADADR,CADAAR,CAADDR,CAADAR,CAAADR,CAAAAR,VALUE,ARGUMENTS,TERRIBLYUNIQUE
PROGRAMVARIABLE,UNTRACLIS,TRACLIS,PROPERTY,ITS,HAS,TRAC3,TRACLISED,UNTRAC1,PROPER,IN,DEFINITION,TRAC2,NAME,FN,TRAC1,M,B
,SELECT2,REMP1,OF,PROPERTIES,PRINTPROP,PROG2,PRINTP1,D,U,MAPCAR,WAY,J,MAKCBLR,PICK,R,EQUALS,NO,ATOMIC,FORMATQ,FORMATP,P
,TPXD0303A,S,FTQZ01R,N,FORMAT,AS,TO,GIVEN,WAS,V,0,CSET,CONC1,DEFINE,DEFLIST,DEFLIS1,PRO,L,OB,DEF1,A,AND,APPEND,APPLY,AP
VAL,APVAL1,ATOM,ATTRIB,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,CAR,CDR,,,COMPTRAC,COND,CONS,CONST,CP1,COPY,COUNT,DESC,
EQ,EQ1,EQUAL,ERROR,EVAL,EVLIS,EXPR,EXPT,F,FEXPR,FIX,FLO,FSUBR,FUNARG,FUNCTION,GENSYM,GO,INST,INT,INTERN,LABEL,LAMBDA,LI
ST,LOAD,LOC,LOCQ,(,MAKENU,MAKEOB,MAP,MAPCON,MAPLIST,-1,-2,MINUS,NCONC,NFLVAL,NIL,NFWL,NOT,NULL,NUMBER,OBLIST,OR,PAIR,PA
USE,PFWL,PLB,PLUS,PNAME,POWER,PRDCT,PRINT,PRIN2,PROG,PROP,QUOTE,READ,RECIP,RECLAIM,RPLACA,RPLACD,RPLACW,RETURN,),SASSOC
,SEARCH,SET,SETQ,SPEAK,STOP,SUB,SUBR,SUBLIS,SUBST,SUM,SYMBOL,T,TEST1,TEST2,TEST3,TEST4,TIMES,TRACE,TSFLOT,UNCOUNT,ADD,A
LS,ARS,BSS,CLA,COM,LDQ,LXA,LXD,PAX,PDX,PXD,STA,STO,STD,STQ,SXD,TIX,TNX,TNZ,TRA,TSX,TXH,TXI,TXL,TZE,ATOM1,EQ11,NULL1,BIN
,EXP,FUNC,TEMF,TEMP,AC,MQ,$ARG2,$ARG3,$ARG4,$ARG5,$ARG6,$ARG7,$ARG8,*+1,*+2,*+3,*+4,*+5,CAAR,CDAR,CADR,CDDR,CAAAR,CAADR
,CADAR,CACDR,CDAAR,CDADR,CDDAR,CDDDR,$CPPI,$ENPDL,$NOPDL+1,$FREE,$FROUT,$ONE,$ZERO,X,Y,Z,COMPILE,COMP2,COMPAT,DEFF3,GET
,NOMAP,SAP,REMPROP, INTEGRATED 709 COMPILER-INTERPRETER LISP 20 OCT 60    )

 AT THE TCNE THE TIME WILL BE ....  0/ 0  000.0 ... BEEP ...

 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
DEFINE
(((LEFT,(LAMBDA,(L),(COND,((NULL,L),NIL),((ATOM,L),NIL),((NULL,(CDR,L)),NIL),(T,(CONS,(CAR,L),(LEFT,(CDR,L))))))),(RIGH
T,(LAMBDA,(L),(COND,((NULL,L),L),((ATOM,L),L),((NULL,(CDR,L)),L),(T,(RIGHT,(CDR,L)))))),(SIDES,(LAMBDA,(X),(COND,((CADR
,X),(LIST,(1E,(CADAR,X)),(1E,X))),(T,(LIST,(1E,X),(1E,(CADAR,X)))))))),(ROTATE,(LAMBDA,(X),(LIST,(LIST,(CADAR,X),(CAAR,X
)),(SIDES,X)))),(FLEXR,(LAMBDA,(X),(LIST,(LIST,(APPEND,(CDADAR,X),(LIST,(CAAR,X))),(CAADAR,X)),(SIDES,X)))),(FLEXL,(LAM
BDA,(X),(LIST,(LIST,(CONS,(CAAR,X),(LEFT,(CADAR,X))),(CAR,(RIGHT,(CADAR,X)))),(SIDES,X)))),(FLEX,(LAMBDA,(X),(COND,((AT
OM,(CADAR,X)),(LIST,(CAR,(ROTATE,X)),(COND,((CADR,X),F),(T,T)))),((CADR,X),(LIST,(CAR,(FLEXR,X)),F)),(T,(LIST,(CAR,(FLE
XL,X)),T)))))),(FAKE,(LAMBDA,(X),(LIST,X,(QUOTE,NEXT),(QUOTE,FLEXAGON)))),(OPERATE,(LAMBDA,(CO,STO),(COND,((EQUAL,CO,(CA
R,STO)),(FAKE,STO)),(T,(OPERATE,CO,(FLEX,STO)))))),(1E,(LAMBDA,(X),(COND,((ATOM,X),X),(T,(1E,(CAR,X)))))),(COFLEXLIST,(
LAMBDA,(L),(COND,((NULL,L),(QUOTE,FINISHED)),(T,(LIST,(OPERATE,(CAR,L),(FLEX,(LIST,(CAR,L),T))),(COFLEXLIST,(CDR,L)))))
))))

END OF APPLY, VALUE IS ...
(LEFT,RIGHT,SIDES,ROTATE,FLEXR,FLEXL,FLEX,FAKE,OPERATE,1E,COFLEXLIST)
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
COFLEXLIST
 ((((1,2),3),(R,(P,((B,G),(Y,O))))),(((7,6),(9,8)),(5,((2,1),(4,3)))),
((A,B,C),((T,U,V),(G,((N,O,(R,Q,P)),M,(H,(K,J,I),L) ),S),(D,E,F))),
,((((1,2,3,4,5,6,7),26,(13,14,15,16,(23,22,21,20,19,18,17)
,24,25),12,11,10,9),8),((1,2,3,4),((14,((21,22), (16,17,18,19,20),
15)),(7,8,((11,12,13),(9,10))),(5,6))),((2,AAARGH),
(3,((HELP,1),(1,HELP),(2,HELP))))))
```

```
(ARGUMENTS,OF,ROTATE)
(((1,2),3),( ))

(VALUE,OF,ROTATE)
((3,(1,2)),(3,1))

(ARGUMENTS,OF,FLEXL)
((3,(1,2)), )

(VALUE,OF,FLEXL)
(((3,1),2),(3,1))

(ARGUMENTS,OF,ROTATE)
(((3,1),2),( ))

(VALUE,OF,ROTATE)
((2,(3,1)),(2,3))

(ARGUMENTS,OF,FLEXL)
((2,(3,1)), )

(VALUE,OF,FLEXL)
(((2,3),1),(2,3))

(ARGUMENTS,OF,ROTATE)
(((2,3),1),( ))

(VALUE,OF,ROTATE)
((1,(2,3)),(1,2))

(ARGUMENTS,OF,FLEXL)
((1,(2,3)), )

(VALUE,OF,FLEXL)
(((1,2),3),(1,2))

(ARGUMENTS,OF,FAKE)
(((1,2),3),( ))

(VALUE,OF,FAKE)
((((1,2),3),( )),NEXT,FLEXAGON)
```





```
(ARGUMENTS,OF,FLEXR)
((R,(P,((B,G),(Y,O)))),( ))

(VALUE,OF,FLEXR)
(((((B,G),(Y,O)),R),P),(P,R))

(ARGUMENTS,OF,ROTATE)
(((((B,G),(Y,O)),R),P), )

(VALUE,OF,ROTATE)
((P,(((B,G),(Y,O)),R)),(B,P))

(ARGUMENTS,OF,FLEXR)
((P,(((B,G),(Y,O)),R)),( ))

(VALUE,OF,FLEXR)
(((R,P),((B,G),(Y,O))),(B,P))

(ARGUMENTS,OF,FLEXL)
(((R,P),((B,G),(Y,O))), )

(VALUE,OF,FLEXL)
((((R,P),(B,G)),(Y,O)),(R,B))

(ARGUMENTS,OF,FLEXR)
((((R,P),(B,G)),(Y,O)),( ))

(VALUE,OF,FLEXR)
(((O,((R,P),(B,G))),Y),(Y,R))

(ARGUMENTS,OF,ROTATE)
(((O,((R,P),(B,G))),Y), )

(VALUE,OF,ROTATE)
((Y,(O,((R,P),(B,G)))),(O,Y))

(ARGUMENTS,OF,FLEXR)
((Y,(O,((R,P),(B,G)))),( ))

(VALUE,OF,FLEXR)
(((((R,P),(B,G)),Y),O),(O,Y))

(ARGUMENTS,OF,ROTATE)
(((((R,P),(B,G)),Y),O), )

(VALUE,OF,ROTATE)
((O,(((R,P),(B,G)),Y)),(R,O))

(ARGUMENTS,OF,FLEXR)
((O,(((R,P),(B,G)),Y)),( ))

(VALUE,OF,FLEXR)
(((Y,O),((R,P),(B,G))),(R,O))

(ARGUMENTS,OF,FLEXL)
(((Y,O),((R,P),(B,G))), )
```

```
(VALUE,OF,FLEXL)
((((Y,O),(R,P)),(B,G)),(Y,R))

(ARGUMENTS,OF,FLEXR)
((((Y,O),(R,P)),(B,G)),( ))

(VALUE,OF,FLEXR)
(((G,((Y,O),(R,P))),B),(B,Y))

(ARGUMENTS,OF,ROTATE)
(((G,((Y,O),(R,P))),B), )

(VALUE,OF,ROTATE)
((B,(G,((Y,O),(R,P)))),(G,B))

(ARGUMENTS,OF,FLEXR)
((B,(G,((Y,O),(R,P)))),( ))

(VALUE,OF,FLEXR)
(((((Y,O),(R,P)),B),G),(G,B))

(ARGUMENTS,OF,ROTATE)
(((((Y,O),(R,P)),B),G), )

(VALUE,OF,ROTATE)
((G,(((Y,O),(R,P)),B)),(Y,G))

(ARGUMENTS,OF,FLEXR)
((G,(((Y,O),(R,P)),B)),( ))

(VALUE,OF,FLEXR)
(((B,G),((Y,O),(R,P))),(Y,G))

(ARGUMENTS,OF,FLEXL)
(((B,G),((Y,O),(R,P))), )

(VALUE,OF,FLEXL)
((((B,G),(Y,O)),(R,P)),(B,Y))
```

```
(ARGUMENTS,OF,FLEXR)
((((B,G),(Y,O)),(R,P)),( ))

(VALUE,OF,FLEXR)
(((P,((B,G),(Y,O))),R),(R,B))

(ARGUMENTS,OF,ROTATE)
(((P,((B,G),(Y,O))),R), )

(VALUE,OF,ROTATE)
((R,(P,((B,G),(Y,O)))),(P,R))

(ARGUMENTS,OF,FAKE)
((R,(P,((B,G),(Y,O)))),( ))
```

CANNIBAL PROBLEM

There are three types of logical problems solvable by a computer. The first of these involves the programming of a direct, step-by-step solution such as the Tower of Hanoi and Chinese Ring Puzzle. The second is to examine all the possiblilities at a given step and find the one that is correct. The last involves making a decision between several probable solutions and requires a more complicated scheme.

The well-known Cannibal Puzzle is in the second class of problems. It is as follows:

On the bank of the river are situated three man-eating cannibals, three white hunters, and a small boat. The hunters hold the cannibals as captives and wish to transport them across the river. However, there are several stipulations. First, the boat holds only two people. Secondly, only one cannibal can row (in addition to the hunter), and lastly, at no time can there be more cannibals with the absence of hunters) since the cannibals would then eat the white men. The problem is to get the six men across the river in a finite number of steps.

Solution: Let "A" stand for the white hunters
Let "C" stand for the cannibals unable to row
Let "R" stand for the cannibal who can row
Let "B" stand for the boat

Move	Left Bank	Boat	Right Bank
	A A A B C C R		
1)	A A C R	B $\overrightarrow{A}$ C	
2)	A A C R	B $\overleftarrow{A}$	C
3)	A A A	B $\overrightarrow{R}$ C	C
4)	A A A	B $\overleftarrow{R}$	C C
5)	A R	B $\overrightarrow{A}$ A	C C
6)	A R	B $\overleftarrow{A}$ C	A C
7)	A C	B $\overrightarrow{A}$ R	A C
8)	A C	B $\overleftarrow{A}$ C	A R
9)	C C	B $\overrightarrow{A}$ A	A R
10)	C C	B $\overleftarrow{R}$	A A A
11)	C	B $\overrightarrow{R}$ C	A A A
12)	C	B $\overleftarrow{R}$	C A A A
13)		B $\overrightarrow{R}$ C	C A A A
14)			A A A B R C C

For a computer to solve the puzzle, it would have to know precisely the same amount of data as a person plus an ability to recognize patterns. The essence of the whole problem centers about those two things.

Since the problem can be solved recursively, it is well adapted for LISP.

The various functions defined are as follows:

1)	Element	-	to determine if an atomic is a member of a list or not (predicate)
2)	Subset	-	to determine if a list is composed of a larger list (prediaate)
3)	Same*	-	a predicate to determine if two lists are identical-regardless of permutations
4)	Same	-	a predicate to determine if two lists of lists are the same regardless of permutations
5)	Member	-	a predicate to determine if a list of lists is an element of another list of lists
6)	Doubletrip	-	a predi ate to determine if a list has exactly two or three members
7)	Remove	-	takes the first occurance of an atomic symbol of a list
8)	Erase	-	removes each element of one list from another
9)	Count	-	counts the number of times a particular symbol is used in a list (function)
10)	Divide	-	function to prepare a list of all possible pairs and singletons from a given list
11)	Divide	-	a particular function that adds "B", the boat to each of the lists of divide
12)	Pare	-	a function that takes a given element and pairs it with each element in a list
13)	Reverse	-	a function to put the last item on front a list
14)	Reverse	-	a function to interchange a list of lists of two members ((A B) (C D) (D F)) → ((B A) (D C) (F E))
15)	Eggreat	-	a predicate to tell if a given list has as many or more members as another list
16)	Transfer	-	a predicate to determine whether or not a boat-load is legal or not by the rules of the problem

17) Sail - makes one boatload from the left side of the river to the right.

18) Sail* - makes one boatload from the right side of the river to the left.

The problem works as follows:
Given the list (A A A B C C R) on the left side of the river, the function "Divide" makes a list of all the possible moves. They are gone through one-by-one to see if they are legal ("Transfer"), then tested to see if, given this particular situation, the same move has been made before. It does this by examining ("Member") a list made of all previous moves (4th argument of Sail and Sail*. If the move is legal and has not been made, then Sail and Sail* make the move and the other takes over. This process repeats until the one entire sextet is moved across the river.

Bob Yates

DEFINE

```
(SAIL* (LAMBDA (L M N P)
(COND ((NULL N) NIL)
((NULL M) (SAIL (CADADR P)
(DIVIDE (CADADR P)) (CAADR P) (REVERSE P)))
((OR (NOT (TRANSFER (ERASE (CAR M) L) (CAR M) N))
(MEMBER (LIST (ERASE (CAR M) L) (APPEND (CAR M) N)) P))
(SAIL* L (CDR M) N P))
(T (SAIL (APPEND (CAR M) N)
(DIVIDE (APPEND (CAR M) N))
(ERASE (CAR M) L)
(REVERSE (CONS (LIST (ERASE (CAR M) L) (APPEND (CAR M) N)) P)))))))
(SAIL (LAMBDA (L M N P)
(COND ((NULL L) NIL)
((NULL M) (SAIL* (CADADR P)
(DIVIDE (CADADR P))
(CAADR p) (REVERSE P)))
((OR (NOT (TRANSFER (ERASE (CAR M) L) (CAR M) N))
(MEMBER (LIST (ERASE (CAR M) L) (APPEND (CAR M) N)) P))
(SAIL L (CDR M) N P))
(T (SAIL* (APPEND (CAR M) N)
(DIVIDE (APPEND (CAR M) N))
(ERASE (CAR M) L)
(REVERSE (CONS (LIST (ERASE (CAR M) L) (APPEND (CAR M) N)) p)))))))
(EQGREAT (LAMBDA (L M)
(COND ((EQUAL L M) T)
((NULL M) T) ((NULL L) F)
(T (EQGREAT (CDR L) (CDR M))))))
(TRANSFER (LAMBDA (L M N)
(AND (ELEMENT (QUOTE B) M)
(DOUBLETRIP M)
(OR (ELEMENT (QUOTE A) M)
(ELEMENT (QUOTE R) M))
(OR (NULL (KOUNT (QUOTE A) L))
(EQGREAT (KOUNT (QUOTE A) L)
(APPEND (KOUNT (QUOTE C) L)
(KOUNT (QUOTE R) L))))
(OR (NULL (KOUNT (QUOTE A) N))
(EQGREAT (KOUNT (QUOTE A) N)
(APPEND (KOUNT (QUOTE C) N)
(KOUNT (QUOTE R) N))))
(OR (NULL (KOUNT (QUOTE A) (APPEND L M)))
(EQGREAT (KOUNT (QUOTE A) (APPEND L M))
(APPEND (KOUNT (QUOTE C) (APPEND L M))
(KOUNT (QUOTE R) (APPEND L M)))))
(OR (NULL (KOUNT (QUOTE A) (APPEND M N)))
(EQGREAT (KOUNT (QUOTE A) (APPEND M N))
(APPEND (KOUNT (QUOTE C) (APPEND M N))
(KOUNT (QUOTE R) (APPEND M N))))))))
(ERASE (LAMBDA (L M)
(COND ((NULL M) NIL) ((NULL L) M)
(T (ERASE (CDR L) (REMOVE (CAR L) M))))))
(DIVIDE (LAMBDA (L)
(COND ((NULL L) NIL)
((ATOM L) (LIST L (QUOTE B)))
(T (MAPLIST (DIVIDE* (REMOVE (QUOTE B) L))
(FUNCTION (LAMBDA (L)
(CONS (QUOTE B) (CAR L)))))))))
(DIVIDE* (LAMBDA (L)
(COND ((NULL L) NIL)
((ATOM L) (LIST (LIST L)))
(T (APPEND (PARE (CAR L) (CDR L))
(DIVIDE* (CDR L)))))))
(PARE (LAMBDA (X L)
(COND ((NULL L) (LIST (LIST X)))
((ATOM L) (LIST (LIST X L)))
(T (CONS (LIST X (CAR L))
(PARE X (CDR L)))))))
(ELEMENT (LAMBDA (X L)
(COND ((NULL L) F)
((EQUAL X (CAR L)) T)
(T (ELEMENT X (CDR L))))))
(REVERSE (LAMBDA (L)
(COND ((NULL L) NIL)
((ATOM L) L)
(T (CONS (REVERSE* (CAR L))
(REVERSE (CDR L)))))))
(REVERSE* (LAMBDA (L)
(COND ((NULL L) NIL)
((ATOM L) L)
(T (APPEND (CDR L) (LIST (CAR L)))))))
(SAME (LAMBDA (L M)
(COND ((AND (NULL L) (NULL M)) T)
((OR (NULL L) (NULL M)) F)
((SAME* (CAR L) (CAR M))
(SAME (CDR L) (CDR M)))
(T F))))
(SAME* (LAMBDA (L M)
(AND (SUBSET L M) (SUBSET M L))))
(SUBSET (LAMBDA (S L)
(COND ((NULL S) T)
((ELEMENT (CAR S) L)
(SUBSET (CDR S) (REMOVE (CAR S) L)))
(T F))))
(REMOVE (LAMBDA (X L) (COND ((NULL L) NIL)
((EQUAL X (CAR L)) (CDR L))
(T (CONS (CAR L) (REMOVE X (CDR L)))))))
(ULTRADOUBLET (LAMBDA (L)
(COND ((NULL L) F) ((ATOM L) F) ((NULL (CDR L)) F) (T T))))
(DOUBLETRIP (LAMBDA (L)
(COND ((OR (ATOM L) (NULL L) (NULL (CDR L))) F)
((OR (NULL (CDDR L)) (NULL (CDDDR L))) T) (T F))))
(MEMBER (LAMBDA (L M)
(COND ((NULL L) T) ((NULL M) F) ((SAME L (CAR M)) T)
(T (MEMBER L (CDR M))))))
(KOUNT (LAMBDA (X L)
(COND ((NULL L) NIL)
((EQUAL X (CAR L))
(CONS (QUOTE 1) (KOUNT X (CDR L))))
(T (KOUNT X (CDR L))))))
(END (LAMBDA () NIL))))    ()
TRACLIS ((SAIL)) ()
TRACLIS ((SAIL*)) ()
SAIL ((A A A B C C R) ((B A A) (B A C) (B A R) (B A) (B C C)
(B C R) (B C) (B R)) () (((A A A B C C R) ())))) ()
        STOP    )))))))))))))))
        FIN
```

```
APPLY OPERATOR AS OF FEBRUARY 10, 1960

READ IN LISTS ...
DEFINE
(((SAIL*,(LAMBDA,(L,M,N,P),(COND,((NULL,N),NIL),((NULL,M),(SAIL,(CADADR,P),(DIVIDE,(CADADR,P)),(CAADR,P),P)),((OR,(MEMB
ER,(LIST,(ERASE,(CAR,M),L),(APPEND,(CAR,M),N)),P),(NOT,(TRANSFER,(ERASE,(CAR,M),L),(CAR,M),N))),(SAIL*,L,(CDR,M),N,P)),
(T,(SAIL,(APPEND,(CAR,M),N),(DIVIDE,(APPEND,(CAR,M),N)),(ERASE,(CAR,M),L),(REVERSE,(CONS,(LIST,(ERASE,(CAR,M),L),(APPEN
D,(CAR,M),N)),P)))))))),(SAIL,(LAMBDA,(L,M,N,P),(COND,((NULL,L),NIL),((NULL,M),(SAIL*,(CADADR,P),(DIVIDE,(CADADR,P)),(CA
ADR,P),P)),((OR,(MEMBER,(LIST,(ERASE,(CAR,M),L),(APPEND,(CAR,M),N)),P),(NOT,(TRANSFER,(ERASE,(CAR,M),L),(CAR,M),N))),(S
AIL,L,(CDR,M),N,P)),(T,(SAIL*,(APPEND,(CAR,M),N),(DIVIDE,(APPEND,(CAR,M),N)),(ERASE,(CAR,M),L),(REVERSE,(CONS,(LIST,(ER
ASE,(CAR,M),L),(APPEND,(CAR,M),N)),P)))))))),(EQGREAT,(LAMBDA,(L,M),(COND,((EQUAL,L,M),T),((NULL,M),T),((NULL,L),F),(T,(
EQGREAT,(CDR,L),(CDR,M)))))),(TRANSFER,(LAMBDA,(L,M,N),(AND,(ELEMENT,(QUOTE,B),M),(DOUBLETRIP,M),(OR,(ELEMENT,(QUOTE,A)
,M),(ELEMENT,(QUOTE,R),M)),(OR,(NULL,(KOUNT,(QUOTE,A),L)),(EQGREAT,(KOUNT,(QUOTE,A),L),(APPEND,(KOUNT,(QUOTE,C),L),(KOU
NT,(QUOTE,R),L)))),(OR,(NULL,(KOUNT,(QUOTE,A),N)),(EQGREAT,(KOUNT,(QUOTE,A),N),(APPEND,(KOUNT,(QUOTE,C),N),(KOUNT,(QUOT
E,R),N)))),(OR,(NULL,(KOUNT,(QUOTE,A),(APPEND,L,M))),(EQGREAT,(KOUNT,(QUOTE,A),(APPEND,L,M)),(APPEND,(KOUNT,(QUOTE,C),(
APPEND,L,M)),(KOUNT,(QUOTE,R),(APPEND,L,M))))),(OR,(NULL,(KOUNT,(QUOTE,A),(APPEND,M,N))),(EQGREAT,(KOUNT,(QUOTE,A),(APP
END,M,N)),(APPEND,(KOUNT,(QUOTE,C),(APPEND,M,N)),(KOUNT,(QUOTE,R),(APPEND,M,N)))))))),(ERASE,(LAMBDA,(L,M),(COND,((NULL
,M),NIL),((NULL,L),M),(T,(ERASE,(CDR,L),(REMOVE,(CAR,L),M)))))),(DIVIDE,(LAMBDA,(L),(COND,((NULL,L),NIL),((ATOM,L),(LIS
T,L,(QUOTE,B))),(T,(MAPLIST,(DIVIDE*,(REMOVE,(QUOTE,B),L)),(FUNCTION,(LAMBDA,(L),(CONS,(QUOTE,B),(CAR,L))))))))),(DIVID
E*,(LAMBDA,(L),(COND,((NULL,L),NIL),((ATOM,L),(LIST,(LIST,L))),(T,(APPEND,(PARE,(CAR,L),(CDR,L)),(DIVIDE*,(CDR,L)))))))
,(PARE,(LAMBDA,(X,L),(COND,((NULL,L),(LIST,(LIST,X))),((ATOM,L),(LIST,(LIST,X,L))),(T,(CONS,(LIST,X,(CAR,L)),(PARE,X,(C
DR,L))))))),(ELEMENT,(LAMBDA,(X,L),(COND,((NULL,L),F),((EQUAL,X,(CAR,L)),T),(T,(ELEMENT,X,(CDR,L)))))),(REVERSE,(LAMBDA
,(L),(COND,((NULL,L),NIL),((ATOM,L),L),(T,(CONS,(REVERSE*,(CAR,L)),(REVERSE,(CDR,L))))))),(REVERSE*,(LAMBDA,(L),(COND,(
(NULL,L),NIL),((ATOM,L),L),(T,(APPEND,(CDR,L),(LIST,(CAR,L))))))),(SAME,(LAMBDA,(L,M),(COND,((AND,(NULL,L),(NULL,M)),T)
,((OR,(NULL,L),(NULL,M)),F),((SAME*,(CAR,L),(CAR,M)),(SAME,(CDR,L),(CDR,M))),(T,F)))),(SAME*,(LAMBDA,(L,M),(AND,(SUBSET
,L,M),(SUBSET,M,L)))),(SUBSET,(LAMBDA,(S,L),(COND,((NULL,S),T),((ELEMENT,(CAR,S),L),(SUBSET,(CDR,S),(REMOVE,(CAR,S),L))
),(T,F)))),(REMOVE,(LAMBDA,(X,L),(COND,((NULL,L),NIL),((EQUAL,X,(CAR,L)),(CDR,L)),(T,(CONS,(CAR,L),(REMOVE,X,(CDR,L))))
))),(ULTRADOUBLET,(LAMBDA,(L),(COND,((NULL,L),F),((ATOM,L),F),((NULL,(CDR,L)),F),(T,T)))),(DOUBLETRIP,(LAMBDA,(L),(COND
,((OR,(ATOM,L),(NULL,L),(NULL,(CDR,L))),F),((OR,(NULL,(CDDR,L)),(NULL,(CDDDR,L))),T),(T,F)))),(MEMBER,(LAMBDA,(L,M),(CO
ND,((NULL,L),T),((NULL,M),F),((SAME,L,(CAR,M)),T),(T,(MEMBER,L,(CDR,M)))))),(KOUNT,(LAMBDA,(X,L),(COND,((NULL,L),NIL),(
(EQUAL,X,(CAR,L)),(CONS,(QUOTE,1),(KOUNT,X,(CDR,L)))),(T,(KOUNT,X,(CDR,L)))))),(END,(LAMBDA, ,NIL))))

TRACLIS
((SAIL))

TRACLIS
((SAIL*))

SAIL
((A,A,A,B,C,C,R),((B,A,A),(B,A,C),(B,A,R),(B,A),(B,C,C),(B,C,R),(B,C),(B,R)), ,(((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C))
))

STOP
```

```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
SAIL
((A,A,A,B,C,C,R),((B,A,A),(B,A,C),(B,A,R),(B,A),(B,C,C),(B,C,R),(B,C),(B,R)), ,(((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C))
))



(ARGUMENTS,OF,SAIL)
(A,A,A,B,C,C,R)
((B,A,A),(B,A,C),(B,A,R),(B,A),(B,C,C),(B,C,R),(B,C),(B,R))

(((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(A,A,A,B,C,C,R)
((B,A,C),(B,A,R),(B,A),(B,C,C),(B,C,R),(B,C),(B,R))

(((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,C)
((B,A,C),(B,A),(B,C))
(A,A,C,R)
(((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,C)
((B,A),(B,C))
(A,A,C,R)
(((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,A),(B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=     50 IR4 ON PDL=      14     GARBAGE= 14487


(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,C),(B,A,R),(B,A),(B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,R),(B,A),(B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
```

```
(B,A,A,A,C,R)
((B,A),(B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,A),(B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,C),(B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

GC. STAT.   ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   163 IR4 ON PDL=    49    GARBAGE= 14326


(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,R),(B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A),(B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,C),(B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A,R),(B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,A),(B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,A,A,C,R)
((B,C,R),(B,C),(B,R))
(C)
(((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

GC. STAT.   ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   160 IR4 ON PDL=    46    GARBAGE= 14219

(ARGUMENTS,OF,SAIL*)
(B,C,R,C)
```

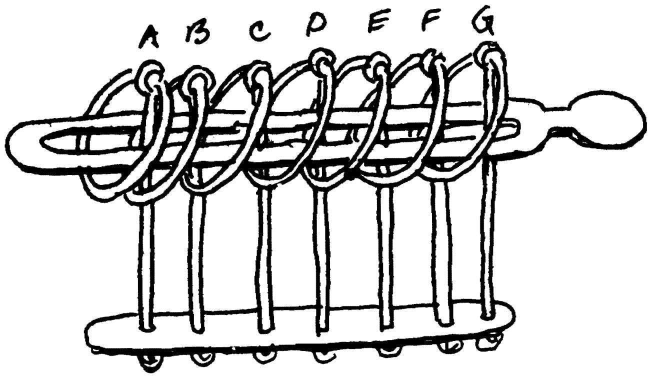
```
((B,C,R),(B,C,C),(B,C),(B,R,C),(B,R),(B,C))
(A,A,A)
(((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,C,R,C)
((B,C,C),(B,C),(B,R,C),(B,R),(B,C))
(A,A,A)
(((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,C,R,C)
((B,C),(B,R,C),(B,R),(B,C))
(A,A,A)
(((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,C,R,C)
((B,R,C),(B,R),(B,C))
(A,A,A)
(((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,C,R,C)
((B,R),(B,C))
(A,A,A)
(((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL)
(B,R,A,A,A)
((B,R,A),(B,R,A),(B,R,A),(B,R),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C,C)
(((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)
))

(ARGUMENTS,OF,SAIL)
(B,R,A,A,A)
((B,R,A),(B,R,A),(B,R),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C,C)
(((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)
))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   198 IR4 ON PDL=    56    GARBAGE= 13939

(ARGUMENTS,OF,SAIL)
(B,R,A,A,A)
((B,R,A),(B,R),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C,C)
(((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)
))

(ARGUMENTS,OF,SAIL)
(B,R,A,A,A)
((B,R),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C,C)
(((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)
))

(ARGUMENTS,OF,SAIL)
(B,R,A,A,A)
```

```
((B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C,C)
(((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)
))

(ARGUMENTS,OF,SAIL*)
(B,A,A,C,C)
((B,A,A),(B,A,C),(B,A,C),(B,A),(B,A,C),(B,A,C),(B,A),(B,C,C),(B,C),(B,C))
(R,A)
(((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)
),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,C,C)
((B,A,C),(B,A,C),(B,A),(B,A,C),(B,A,C),(B,A),(B,C,C),(B,C),(B,C))
(R,A)
(((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)
),((R,C),(A,A,A,B,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   250 IR4 ON PDL=     72    GARBAGE= 13782

(ARGUMENTS,OF,SAIL)
(B,A,C,R,A)
((B,A,C),(B,A,R),(B,A,A),(B,A),(B,C,R),(B,C,A),(B,C),(B,R,A),(B,R),(B,A))
(A,C)
(((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)
),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,C,R,A)
((B,A,R),(B,A,A),(B,A),(B,C,R),(B,C,A),(B,C),(B,R,A),(B,R),(B,A))
(A,C)
(((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)
),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,R,A,C)
((B,A,R),(B,A,A),(B,A,C),(B,A),(B,R,A),(B,R,C),(B,R),(B,A,C),(B,A),(B,C))
(C,A)
(((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)
),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,R,A,C)
((B,A,A),(B,A,C),(B,A),(B,R,A),(B,R,C),(B,R),(B,A,C),(B,A),(B,C))
(C,A)
(((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)
),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,R,A,C)
((B,A,C),(B,A),(B,R,A),(B,R,C),(B,R),(B,A,C),(B,A),(B,C))
(C,A)
(((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)
),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   275 IR4 ON PDL=     79    GARBAGE= 13511

(ARGUMENTS,OF,SAIL)
```

```
(B,A,C,C,A)
((B,A,C),(B,A,C),(B,A,A),(B,A),(B,C,C),(B,C,A),(B,C),(B,C,A),(B,C),(B,A))
(R,A)
(((B,A,C,C,A),(R,A)),((C,A),(B,A,R,A,C)),((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)
),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,C,C,A)
((B,A,C),(B,A,A),(B,A),(B,C,C),(B,C,A),(B,C),(B,C,A),(B,C),(B,A))
(R,A)
(((B,A,C,C,A),(R,A)),((C,A),(B,A,R,A,C)),((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)
),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL)
(B,A,C,C,A)
((B,A,A),(B,A),(B,C,C),(B,C,A),(B,C),(B,C,A),(B,C),(B,A))
(R,A)
(((B,A,C,C,A),(R,A)),((C,A),(B,A,R,A,C)),((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)
),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A,A),(B,A,R),(B,A,A),(B,A),(B,A,R),(B,A,A),(B,A),(B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A,R),(B,A,A),(B,A),(B,A,R),(B,A,A),(B,A),(B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=    341 IR4 ON PDL=     99    GARBAGE= 13242

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A,A),(B,A),(B,A,R),(B,A,A),(B,A),(B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A),(B,A,R),(B,A,A),(B,A),(B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A,R),(B,A,A),(B,A),(B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A,A),(B,A),(B,R,A),(B,R),(B,A))
```

```
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,A),(B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,R,A),(B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   370 IR4 ON PDL=   106    GARBAGE= 13115

(ARGUMENTS,OF,SAIL*)
(B,A,A,R,A)
((B,R),(B,A))
(C,C)
(((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)
),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL)
(B,R,C,C)
((B,R,C),(B,R,C),(B,R),(B,C,C),(B,C),(B,C))
(A,A,A)
(((B,R,C,C),(A,A,A)),((C,C),(B,A,A,R,A)),((B,A,C,C,A),(R,A)),((C,A),(B,A,R,A,C)),((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)
),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C
)))

(ARGUMENTS,OF,SAIL*)
(B,R,C,A,A,A)
((B,R,C),(B,R,A),(B,R,A),(B,R,A),(B,R),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C)
(((B,R,C,A,A,A),(C)),((A,A,A),(B,R,C,C)),((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)
),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R
)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,R,C,A,A,A)
((B,R,A),(B,R,A),(B,R,A),(B,R),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C)
(((B,R,C,A,A,A),(C)),((A,A,A),(B,R,C,C)),((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)
),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R
)),((R,C),(A,A,A,B,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   361 IR4 ON PDL=   103    GARBAGE= 12897

(ARGUMENTS,OF,SAIL*)
(B,R,C,A,A,A)
((B,R,A),(B,R,A),(B,R),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C)
(((B,R,C,A,A,A),(C)),((A,A,A),(B,R,C,C)),((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)
),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R
)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,R,C,A,A,A)
((B,R,A),(B,R),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C)
(((B,R,C,A,A,A),(C)),((A,A,A),(B,R,C,C)),((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)
),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R
)),((R,C),(A,A,A,B,C)))

(ARGUMENTS,OF,SAIL*)
(B,R,C,A,A,A)
((B,R),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,A,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))
(C)
(((B,R,C,A,A,A),(C)),((A,A,A),(B,R,C,C)),((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)
),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R
)),((R,C),(A,A,A,B,C)))

GC. STAT.    ENT. TIME 000000  MARK TIME 000000  EXIT TIME 000000  PDL DEPTH=   415 IR4 ON PDL=   119    GARBAGE= 12795

(ARGUMENTS,OF,SAIL)
(B,R,C)
((B,R,C),(B,R),(B,C))
(C,A,A,A)
(((B,R,C),(C,A,A,A)),((C),(B,R,C,A,A,A)),((B,R,C,C),(A,A,A)),((C,C),(B,A,A,R,A)),((B,A,C,C,A),(R,A)),((C,A),(B,A,R,A,C)
),((B,A,C,R,A),(A,C)),((R,A),(B,A,A,C,C)),((B,R,A,A,A),(C,C)),((A,A,A),(B,C,R,C)),((B,A,A,A,C,R),(C)),((A,A,C,R),(B,A,C
)),((A,A,A,B,C,C,R), ),((A,A,A,B,C),(R,C)))

(ARGUMENTS,OF,SAIL*)
(B,R,C,C,A,A,A)
((B,R,C),(B,R,C),(B,R,A),(B,R,A),(B,R,A),(B,R),(B,C,C),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,C,A),(B,C,A),(B,C,A),(B,C),(B,A
,A),(B,A,A),(B,A),(B,A,A),(B,A),(B,A))

(((B,R,C,C,A,A,A), ),((C,A,A,A),(B,R,C)),((B,R,C,A,A,A),(C)),((A,A,A),(B,R,C,C)),((B,A,A,R,A),(C,C)),((R,A),(B,A,C,C,A)
),((B,A,R,A,C),(C,A)),((A,C),(B,A,C,R,A)),((B,A,A,C,C),(R,A)),((C,C),(B,R,A,A,A)),((B,C,R,C),(A,A,A)),((C),(B,A,A,A,C,R
)),((B,A,C),(A,A,C,R)),( ,(A,A,A,B,C,C,R)),((R,C),(A,A,A,B,C)))

(VALUE,OF,SAIL*)

(VALUE,OF,SAIL)

(VALUE,OF,SAIL*)

(VALUE,OF,SAIL*)

(VALUE,OF,SAIL*)

(VALUE,OF,SAIL*)

(VALUE,OF,SAIL*)

(VALUE,OF,SAIL)
```

THE CHINESE RING PUZZLE: ITS SOLUTION IN LISP



THE RING PUZZLE

The computer, of course, is incapable of analyzing, and therefore actually solving, the puzzle. In defining the puzzle to the computer, therefore, it is necessary to include all such analysis. When this is done the computer is able to mechanically perform each step under its particular conditions, according to rules.

The most obvious rule in the solution of the ring puzzle is that only the successor of the first ring left on can be removed, unless the first ring is ring A. As the object of the puzzle is to remove all of the rings, this rule makes it soon apparent that one must work toward removing the last ring (G)first, and work forward to A. To remove G, F must be left on and rings A through E removed. In order to do this, W must be removed, which means D must be left on and rings A, B and C removed. This means that C must be removed by leaving on B and removing A. It is obvious that every other ring, starting with the ring to be removed, must be removed first in order to remove the ring in question. Two cases exist, therefore, one when the number of rings is even, and one when the number of rings is odd. To discover whether the list of rings which remain on is even or odd, the predicate EVEN is used. The value is true if the list is nil, and if it is not, the predicate NOT EVEN is performed on CDR of the list.

Because of the primary rule of the puzzle and its dependence on the succession of the rings, two more functions of the analytical type are needed. One of these is a predicate which is true if a certain element is the successor of another given element according to a reference list, and the other is a function which chooses from the same reference list the successor of a given element. The first, called SUCCESSOR, works by testing the equality of the first element and CAR of the reference list. If this is true, it goes on to test if the second element is equal to CADR of the reference list. If this is true, then the value is true, if not, it is false. If the first element is not equal to CAR of the reference list, the predicate operates on CDR of the reference list. The second function, called CHOOSE goes through the reference list, comparing each element to the given element by means of the predicate SUCCESSOR until it finds one which equals the given element.

The functions which provide for the actual movement of the rings on and off are REMOVE and REPLACE. REMOVE compares a given element with each element of the list from which it is to be removed, rebuilding the part of the list it has already run through by means of the function CONS. REPLACE is used only to replace the second element in its correct place according to a reference list in another list.

The function which combines the others to perform the solution is CLOVIS. CLOVIS is a function of three arguments, all of which are lists. The first is the list of "on" elements, (M), the second is a list of "off" elements, (N), and the final list is a reference list of both on and off elements in the same sequence they would be if all rings were on (1). The function is divided essentially into three parts. The first of these is a conditional NULL function which ends the solution as soon as the list M is empty. The remaining portion is a test by EVEN to discover whether the list M is odd or even. If M is even, there exist the two possibilities that CADR M is the successor of CAR M according to L, or that it is not. If it is, then CLOVIS is repeated on a new list M made up of CAR M and CDDR M, a new list N made up of REPLACE of M in N, and the old list L. If CADR M is not the successor of CAR M, the function CLOVIS is repeated with a new M with the successor of CAR of M replaced, a new N with a corresponding removal made, and the list L. If the original M is ODD and CAR M equals CAR L, then CLOVIS is repeated on CDR M, the list N with CAR M placed in front of it, and the list L. If CAR M does not equal CAR L, CAR L is added to the beginning of M and removed from N and CLOVIS is repeated. It can be observed that this takes into consideration all of the cases necessary to solve the ring puzzle. In order to observe each step as it takes place, it is necessary only to use TRACLIS to track CLOVIS.

PETER CONRAD

```
APPLY OPERATOR AS OF FEBRUARY 10, 1960

READ IN LISTS ...
DEFINE
%%%EVEN,%LAMBDA,%L□,%COND,%%NULL,L□,T□,%T,%NOT,%EVEN,%CDR,L□□□□□□□,%SUCCESSOR,%LAMBDA,%X,Y,L□,%COND,%%NULL,L□,F□,%%EQ,%
CAR,L□,Y□,%COND,%%EQ,%CADR,L□,X□,T□,%T,F□□□,%T,%SUCCESSOR,X,Y,%CDR,L□□□□□□,%CHOOSE,%LAMBDA,%X,L□,%COND,%%SUCCESSOR,%CAD
R,L□,X,L□,%CADR,L□□,%T,%CHOOSE,X,%CDR,L□□□□□□,%REMOVE,%LAMBDA,%X,L□,%COND,%%NULL,L□,NIL□,%%EQ,X,%CAR,L□□,%CDR,L□□,%T,%C
ONS,%CAR,L□,%REMOVE,X,%CDR,L□□□□□□□,%REPLACE,%LAMBDA,%N,M,L□,%COND,%%NULL,N□,%LIST,%CADR,M□□□,%%EQ,%CAR,M□,%CAR,L□□,%CO
NS,%CADR,M□,N□□,%T,%CONS,%CAR,N□,%REPLACE,%CDR,N□,M,%CDR,L□□□□□□□[redacted],%COND,%%NULL,M□,NIL□,%%EVEN,M
□,%COND,%%SUCCESSOR,%CADR,M□,%CAR,M□,L□,%CLOVIS,%CONS,%CAR,M□,%CDDR,M□□,%REPLACE,N,M,L□,L□□,%T,%CLOVIS,%CONS,%CAR,M□,%C
ONS,%CHOOSE,%CAR,M□,L□,%CDR,M□□□,%REMOVE,%CHOOSE,%CAR,M□,L□,N□,L□□□□,%T,%COND,%%EQ,%CAR,M□,%CAR,L□□,%CLOVIS,%CDR,M□,%CO
NS,%CAR,M□,N□,L□□,%T,%CLOVIS,%CONS,%CAR,L□,M□,%CDR,N□,L□□□□□□□,%END,%LAMBDA, ,NIL□□□□

TRACLIS
%%CLOVIS□□

CLOVIS
%%A,B,C,D,E,F,G□, ,%A,B,C,D,E,F,G□□

STOP
```

```
OBJECT LIST NOW IS ...
%G,E,C,END,CLOVIS,REPLACE,REMOVE,CHOOSE,SUCCESSOR,EVEN,PF,VAR,FORM,SEARCHF,T1,UF,FF,DF,COMPSRCH,BNDV,SELECT,CSETQ,CONC,
CDDDDR,CDDDAR,CDDADR,CDDAAR,CDADDR,CDADAR,CDAADR,CDAAAR,CADDDR,CADDAR,CADADR,CADAAR,CAADDR,CAADAR,CAAADR,CAAAAR,VALUE,A
RGUMENTS,TERRIBLYUNIQUEPROGRAMVARIABLE,UNTRACLIS,TRACLIS,PROPERTY,ITS,HAS,TRAC3,TRACLISED,UNTRAC1,PROPER,IN,DEFINITION,
TRAC2,NAME,FN,TRAC1,K,B,SELECT2,REMP1,CF,PROPERTIES,PRINTPROP,PROG2,PRINTP1,C,U,MAPCAR,WAY,J,MAKCBLR,PICK,R,EQUALS,NO,A
TOMIC,FORMATQ,FORMATP,P,T2XD0303A,S,FTGZ01R,H,FORMAT,AS,TO,GIVEN,WAS,V,G,CSET,CONC1,DEFINE,DEFLIST,DEFLIS1,PRO,L,OB,DEF
1,A,AND,APPEND,APPLY,APVAL,APVAL1,ATOM,ATTRIB,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,18,CAR,CDR,,,COMPTRAC,COND,CONS,CON
ST,CP1,COPY,COUNT,DESC,EQ,EQ1,EQUAL,ERROR,EVAL,EVLIS,EXPR,EXPT,F,FEXPR,FIX,FLO,FSUBR,FUNARG,FUNCTION,GENSYM,GO,INST,INT
,INTERN,LABEL,LAMBDA,LIST,LOAD,LOC,LOCQ,%,MAKENU,MAKEOB,MAP,MAPCON,MAPLIST,-1,-2,MINUS,NCONC,NFLVAL,NIL,NFWL,NOT,NULL,N
UMBER,OBLIST,OR,PAIR,PAUSE,PFWL,PLB,PLUS,PNAME,POWER,PRDCT,PRINT,PRIN2,PROG,PROP,QUOTE,READ,RECIP,RECLAIM,RPLACA,RPLACD
,RPLACW,RETURN,¤,SASSOC,SEARCH,SET,SETQ,SPEAK,STOP,SUB,SUBR,SUBLIS,SUBST,SUM,SYMBOL,T,TEST1,TEST2,TEST3,TEST4,TIMES,TRA
CE,TSFLOT,UNCOUNT,ADD,ALS,ARS,BSS,CLA,COM,LDQ,LXA,LXD,PAX,PDX,PXD,STA,STO,STD,STQ,SXD,TIX,TNX,TNZ,TRA,TSX,TXH,TXI,TXL,T
ZE,ATOM1,EQ11,NULL1,SIN,EXP,FUNC,TEMF,TEMP,AC,MQ,$ARG2,$ARG3,$ARG4,$ARG5,$ARG6,$ARG7,$ARG8,*&1,*&2,*&3,*&4,*&5,CAAR,CDA
R,CADR,CDDR,CAAAR,CAADR,CADAR,CADDR,CDAAR,CDADR,CDDAR,CDDDR,$CPPI,$ENPDL,$NCPDL&1,$FREE,$FROUT,$ONE,$ZERO,X,Y,Z,COMPILE
,COMP2,COMPAI,DEFF3,SET,NUMAP,SAP,REMPROP, INTEGRATED 709 COMPILER-INTERPRETER LISP 20 OCT 60   ¤

 FUNCTION  APPLY%F,X,P¤ HAS BEEN ENTERED, ARGUMENTS..
DEFINE
%%%EVEN,%LAMBDA,%L¤,%COND,%%NULL,L¤,T¤,%T,%NOT,%EVEN,%CDR,L¤¤¤¤¤¤¤,%SUCCESSCR,%LAMBDA,%X,Y,L¤,%COND,%%NULL,L¤,F¤,%%EQ,%
CAR,L¤,Y¤,%COND,%%EQ,%CADR,L¤,X¤,T¤,%T,F¤¤¤,%T,%SUCCESSOR,X,Y,%CDR,L¤¤¤¤¤¤,%CHOOSE,%LAMBDA,%X,L¤,%COND,%%SUCCESSOR,%CAD
R,L¤,X,L¤,%CADR,L¤¤,%T,%CHOOSE,X,%CDR,L¤¤¤¤¤¤,%REMOVE,%LAMBDA,%X,L¤,%COND,%%NULL,L¤,NIL¤,%%EQ,X,%CAR,L¤¤,%CDR,L¤¤,%T,%C
ONS,%CAR,L¤,%REMOVE,X,%CDR,L¤¤¤¤¤¤¤,%REPLACE,%LAMBDA,%N,M,L¤,%COND,%%NULL,N¤,%LIST,%CADR,M¤¤¤,%%EQ,%CAR,M¤,%CAR,L¤¤,%CO
NS,%CADR,M¤,N¤¤,%T,%CONS,%CAR,N¤,%REPLACE,%CDR,N¤,M,%CDR,L¤¤¤¤¤¤¤,%CLOVIS,%LAMBDA,%N,M,L¤,%COND,%%NULL,M¤,NIL¤,%%EVEN,M
¤,%COND,%%SUCCESSOR,%CADR,M¤,%CAR,M¤,L¤,%CLOVIS,%CONS,%CAR,M¤,%CDDR,M¤¤,%REPLACE,N,M,L¤,L¤¤,%T,%CLOVIS,%CONS,%CAR,M¤,%C
ONS,%CHOOSE,%CAR,M¤,L¤,%CDR,M¤¤¤,%REMOVE,%CHOOSE,%CAR,M¤,L¤,N¤,L¤¤¤¤,%T,%COND,%%EQ,%CAR,M¤,%CAR,L¤¤,%CLOVIS,%CDR,M¤,%CO
NS,%CAR,M¤,N¤,L¤¤,%T,%CLOVIS,%CONS,%CAR,L¤,M¤,%CDR,N¤,L¤¤¤¤¤¤¤,%END,%LAMBDA, ,NIL¤¤¤¤


END OF APPLY, VALUE IS ...
%EVEN,SUCCESSOR,CHOOSE,REMOVE,REPLACE,CLOVIS,END¤
```

```
FUNCTION  APPLY%F,X,P¤ HAS BEEN ENTERED, ARGUMENTS..
CLOVIS
%%A,B,C,D,E,F,G¤,  ,%A,B,C,D,E,F,G¤¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,D,E,F,G¤

%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,D,E,F,G¤
%A¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,D,E,F,G¤
%A,C¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,D,E,F,G¤
%C¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,D,E,F,G¤
%B,C¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%D,E,F,G¤
%A,B,C¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%D,F,G¤
%A,B,C,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,D,F,G¤
%B,C,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,D,F,G¤
%C,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,D,F,G¤
%A,C,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,D,F,G¤
%A,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,D,F,G¤
%E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,D,F,G¤
%B,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,D,F,G¤
%A,B,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,F,G¤
%A,B,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,F,G¤
%B,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,F,G¤
%D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,F,G¤
%A,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,F,G¤
%A,C,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,F,G¤
%C,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,F,G¤
%B,C,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%F,G¤
%A,B,C,D,E¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%F¤
%A,B,C,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,F¤
%B,C,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,F¤
%C,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,F¤
%A,C,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,F¤
%A,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,F¤
%D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,F¤
%B,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,F¤
%A,B,D,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,D,F¤
%A,B,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,D,F¤
%B,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,D,F¤
%E,G¤
%A,B,C,D,E,F,G¤
```

```
%ARGUMENTS.OF.CLOVIS¤
%B,C,D,F¤
%A,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,D,F¤
%A,C,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,D,F¤
%C,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,D,F¤
%B,C,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%D,F¤
%A,B,C,E,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%D,E,F¤
%A,B,C,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,D,E,F¤
%B,C,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,D,E,F¤
%C,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,D,E,F¤
%A,C,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,D,E,F¤
%A,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,D,E,F¤
%G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,D,E,F¤
%B,G¤
%A,B,C,D,E,F,G¤
```

```
%ARGUMENTS,OF,CLOVIS¤
%C,D,E,F¤
%A,B,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,E,F¤
%A,B,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,E,F¤
%B,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,E,F¤
%D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,E,F¤
%A,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,E,F¤
%A,C,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,E,F¤
%C,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,E,F¤
%B,C,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%E,F¤
%A,B,C,D,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%E¤
%A,B,C,D,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,E¤
%B,C,D,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,E¤
%C,D,F,G¤
%A,B,C,D,E,F,G¤
```

```
%ARGUMENTS,OF,CLOVIS¤
%B,E¤
%A,C,D,F,G¤
%A,B,C,D,E,F,G¤

GC. STAT.    ENT. TIME

%ARGUMENTS,OF,CLOVIS¤
%B,C,E¤
%A,D,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,E¤
%D,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,E¤
%B,D,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,E¤
%A,B,D,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%C,D,E¤
%A,B,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,C,D,E¤
%B,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,C,D,E¤
%F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,C,D,E¤
%A,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%B,D,E¤
%A,C,F,G¤
%A,B,C,D,E,F,G¤

%ARGUMENTS,OF,CLOVIS¤
%A,B,D,E¤
%C,F,G¤
%A,B,C,D,E,F,G¤
```

```
%ARGUMENTS,OF,CLOVIS□
%A,D,E□
%B,C,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%D,E□
%A,B,C,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%D□
%A,B,C,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,D□
%B,C,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,B,D□
%C,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%B,D□
%A,C,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%B,C,D□
%A,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,B,C,D□
%E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,C,D□
%B,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%C,D□
%A,B,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%C□
%A,B,D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,C□
%B,D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,B,C□
%D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%B,C□
%A,D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%B□
%A,C,D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A,B□
%C,D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□
%A□
%B,C,D,E,F,G□
%A,B,C,D,E,F,G□

%ARGUMENTS,OF,CLOVIS□

%A,B,C,D,E,F,G□
%A,B,C,D,E,F,G□
```

The LISP Function FLEXAGON

Flexagon is a function of a list of sublists. The list is the sign sequence for a single-cycle flexagon of any cycle, G. The sublists are the individual signs of the sign sequence, and must consist of +'s. Thus for a cycle eight flexagon, - must be written as +++++++, -- as ++++++, etc. The function constructs a leaf for each sign of the sign sequence according to the following scheme:

The hinge positions around a class G leaf are represented by a sublist: (Position 0, Position 1, Position 2, ..., Position G-1). The symbols I, ϕ, and X where present in such a list designate respectively the positions of the input-hinge, the output-hinge, and the unused edges. These sublists comprise a list of the leaves in their position in the unit as read from top to bottom. Since for single-cycle flexagons each leaf is hinged to the one next below it, the ϕ of one leaf will be in the same position as the I of the next on the list.

To process a sign sequence for a single cycle flexagon, each sign must be made into a list of +'s. Thus +3 = (+,+,+), etc. These sublists must be strung together into a sign sequence list, S. Thus the sign sequence +2 -1 -2 +1 +1 becomes ((+,+), (+,+,+,+), (+,+,+), (+), (+)).

The following functions are used to evaluate FLEXAGON:

Element

Prepare

Reassociate

Invert

Compare (L,M) - predicate - T if Lis null or $\leq$ M is length
F if M>L

<u>Sub</u> (X,L,M) - function - L and M are lists; X is an atom - subtracts the length of L from the length of M and makes a list composed of X's equal in length to this difference.

<u>Dummy</u> (X,L) - function - L is a list - Creates a list of X's equal in length to L.

<u>Remove</u> (I,L)

<u>Leaf</u> (E,L) - function - E is a sign sublist of the sign sequence list L. The function creates a leaf list with I in the zero position and ϕ in the 0+length - of - E position.

<u>Rrot</u> (L) (right rotate) - function - L is a list - The last item of L is removed and placed at the beginning of the list.

<u>Lrot</u> (L) (left rotate) - function - L is a list - The first item of L is removed and placed at the end of the list.

<u>Orient</u> (V,U) - function - V and U are leaf lists. V is oriented with respect to U in the sense that it is right rotated until its I position coincides with the ϕ position of U.

<u>Hinge</u> (E,L) - function - E is a leaf list; L is either null or is a list of leaf lists - If L is null, the result is a list with one element E. If not, E is oriented with respect to the first member of L and then inserted at the beginning of L.

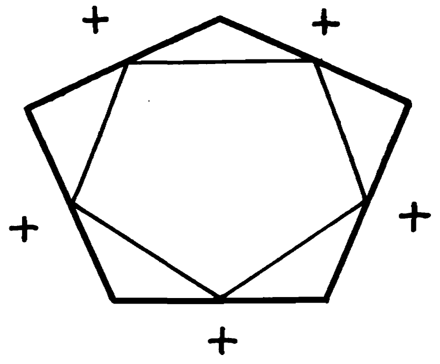
<u>Construct</u> (L,M) - function - L and M are identical sign sequence lists - This hinges all the leaf lists together in reverse order from the final flexagon list.

<u>Flexagon</u> (S) - Makes a flexagon out of sign sequence list S.

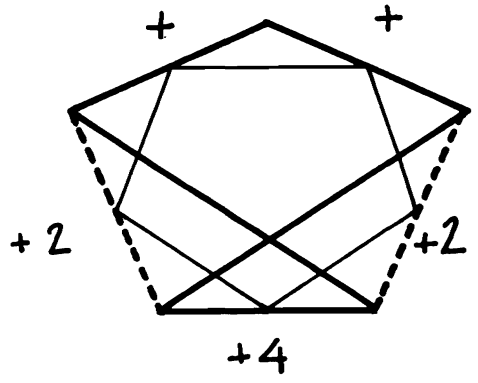
D. Hartline

```
*       CHK                  01-128  7550-000-27350          MCINTOSH          L
*       REM      D HARTLINE                                                    L
*       REM      PLACE LISP BINARY TAPE ON TAPE UNIT B-7                  143  L
*       REM      SCRATCH TAPE ON TAPE UNIT B-8                                 L
*       REM     OUTPUT MAY BE MONITORED BY DEPRESSING SWITCH 3.                L
*       REM      NORMAL HALT      OCTAL 100                                    L
*55   3 +    178  SXN-    76     INM      /    10 3
        TST                                                                    L
        TST                                                                    L
                                                                      DEFINE ((L

(FLEXAGON (LAMBDA (S)                                                          L
(INVERT (CONSTRUCT (INVERT S) S))))                                            L

(CONSTRUCT (LAMBDA (L M) (COND                                                 L
((NULL (CDR L)) (CONS (LEAF (CAR L) M) NIL))                                   L
(T (HINGE (LEAF (CAR L) M) (CONSTRUCT (CDR L) M))))))                          L

(HINGE (LAMBDA (E L) (COND                                                     L
((NULL L) (CONS E NIL))                                                        L
(T (CONS (ORIENT E (CAR L)) L)))))

(ORIENT (LAMBDA (V U) (COND
((NOT (ELEMENT (QUOTE O) U)) (QUOTE BAD2))
((EQUAL (QUOTE O) (CAR U)) V)                                                  L
(T (ORIENT (RROT V) (LROT U))))))

(LROT (LAMBDA (L)                                                              L
(APPEND (CDR L) (CONS (CAR L) NIL))))                                          L

(RROT (LAMBDA (L)                                                              L
(INVERT (LROT (INVERT L)))))

(LEAF (LAMBDA (E L)                                                            L
(APPEND (CONS (QUOTE I) (REMOVE (QUOTE X) (DUMMY (QUOTE X) E)))                L
(CONS (QUOTE O) (REMOVE (QUOTE X) (SUB (QUOTE X) E L)))))))                    L

(REMOVE (LAMBDA (I L) (COND                                                    L
((NULL L) NIL)                                                                 L
((NOT (EQUAL I (CAR L))) (CONS (CAR L) (REMOVE I (CDR L))))                    L
(T (CDR L)))))                                                                 L

(DUMMY (LAMBDA (X L) (COND                                                     L
((ATOM L) (QUOTE BAD1))
((NULL L) NIL)                                                                 L
((NULL (CDR L)) (CONS X NIL))                                                  L
(T (CONS X (DUMMY X (CDR L)))))))

(SUB (LAMBDA (X L M) (COND                                                     L
((NOT (COMPARE L M)) (SUB X (SUB (CAR L) M L) M))                              L
((NULL L) (DUMMY X M))                                                         L
(T (SUB X (CDR L) (CDR M))))))                                                 L

(COMPARE (LAMBDA (L M) (COND                                                   L
((NULL L) T)                                                                   L
((NULL M) F)                                                                   L
(T (COMPARE (CDR L) (CDR M))))))                                               L

(INVERT (LAMBDA (L) (COND                                                      L
((NULL (PREPARE L)) L)                                                         L
(T (TRANSPOSE (REASSOCIATE (PREPARE L)))))))                                   L
(TRANSPOSE (LAMBDA (L) (COND
((NULL (CDR L)) L)
(T (CONS (CADR L) (TRANSPOSE (CAR L)))))))

(REASSOCIATE (LAMBDA (L) (COND
((NULL (CDDR L)) L)
(T (REASSOCIATE (CONS (LIST (CAR L) (CADR L)) (CDDR L)))))))

(PREPARE (LAMBDA (L) (COND
((OR (ATOM L) (NULL L)) NIL)
(T (CONS (CONS (CAR L) NIL) (CDR L))))))

(TRUTHVALUE (LAMBDA (P) (COND
(P (QUOTE TRUE))
(T (QUOTE FALSE)))))

(ELEMENT (LAMBDA (X L) (COND
((NULL L) F)
((EQUAL X (CAR L)) T)
(T (ELEMENT X (CDR L))))))

(END (LAMBDA () NIL))))   ()


(LAMBDA (L) (TRUTHVALUE (ATOM L))) ((A B C)) ()
(LAMBDA (L) (TRUTHVALUE (ATOM L))) (A) ()
(LAMBDA (L) (TRUTHVALUE (NULL L))) ((A)) ()
(LAMBDA (L) (TRUTHVALUE (NULL L))) ((() () ())) ()
(LAMBDA (L) (TRUTHVALUE (NULL L))) ((())) ()
(LAMBDA (L) (TRUTHVALUE (NULL L))) (()) ()
LROT ((A B C D)) ()
DUMMY (X ()) ()
DUMMY (X (A B C D (E F))) ()
COMPARE ((A B C) (L M N O)) ()
COMPARE ((R S T U) (V W X Y)) ()
COMPARE ((A B C D E F) (M N O)) ()
(LAMBDA (L M) (TRUTHVALUE (COMPARE L M))) ((A B C) (L M N O)) ()
(LAMBDA (L M) (TRUTHVALUE (COMPARE L M))) ((R S T U) (V W X Y)) ()
(LAMBDA (L M) (TRUTHVALUE (COMPARE L M))) ((A B C D E F) (M N O)) ()
SUB (X (A (B C) D) (M N LM R D)) ()
SUB (X (+ + +) ((+) (+ +) (+ + +) (+))) ()
SUB (X (+ + + + + +) ((+) (+ +) (+ + +) (+))) ()
SUB (X (A B C) (M N O P Q R S)) ()
REMOVE (A (R M A C A L)) ()
REMOVE (A ()) ()
LEAF ((+ + +) ((+) (+ + +) (+ +) (+ + + +) (+))) ()
PREPARE ((A B C D)) ()
PREPARE ((A B)) ()
PREPARE ((A)) ()
PREPARE (()) ()
REASSOCIATE ((A B C D)) ()
REASSOCIATE ((A B C)) ()
REASSOCIATE ((A B)) ()
TRANSPOSE ((((A) B) C) D)) ()
INVERT ((A B C D)) ()
RROT ((A B C D)) ()
ORIENT ((I X O X X) (I X X O X)) ()
ORIENT ((I X X X X O X X) (I X X X O X X X)) ()
ORIENT ((I X X X X O X X) (X X I X X X O X)) ()
HINGE ((I X X O X) ((I X O X X) (A B C) (D E))) ()
```

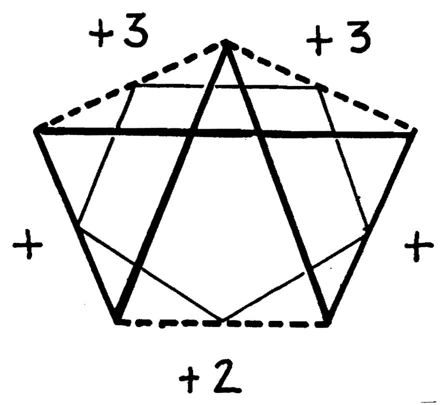
```
CONSTRUCT (((+) (+ + +) (+ +) (+ + + +) (+))) ()
FLEXAGON (((+) (+ + +) (+ +) (+ + + +) (+))) ()
FLEXAGON (((+ + + + +) (+ + +) (+) (+ + +))) ()
FLEXAGON (((+) (+) (+) (+) (+))) ()
FLEXAGON (((+) (+) (+ +) (+ + + +) (+ +))) ()
FLEXACON (((+) (+ +) (+) (+ + +) (+ + +))) ()
FLEXAGON (((+ + +) (+ + +) (+ + +) (+ + +) (+ + +))) ()
FLEXAGON (((+) (+ + +) (+ + +) (+ + + + + + +) (+ + + +)
(+) (+ +) (+ + +))) ()
(LAMBDA (X L) (TRUTHVALUE (ELEMENT (X L)))) (A (R C B F A L)) ()
(LAMBDA (X L) (TRUTHVALUE (ELEMENT (X L)))) (D (R C B F A L)) ()
HINGE ((I X X O X) (I X O X X)) ()
       STOP    )))))))))))))))
       FIN
```



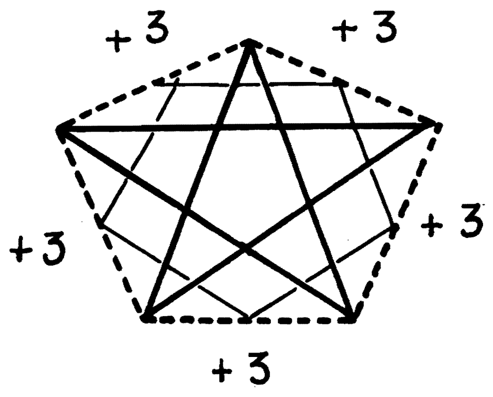
```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FLEXAGON
(((+),(+),(+),(+),(+)))


END OF APPLY, VALUE IS ...
((I,O,X,X,X),(X,I,O,X,X),(X,X,I,O,X),(X,X,X,I,O),(O,X,X,X,I))
```



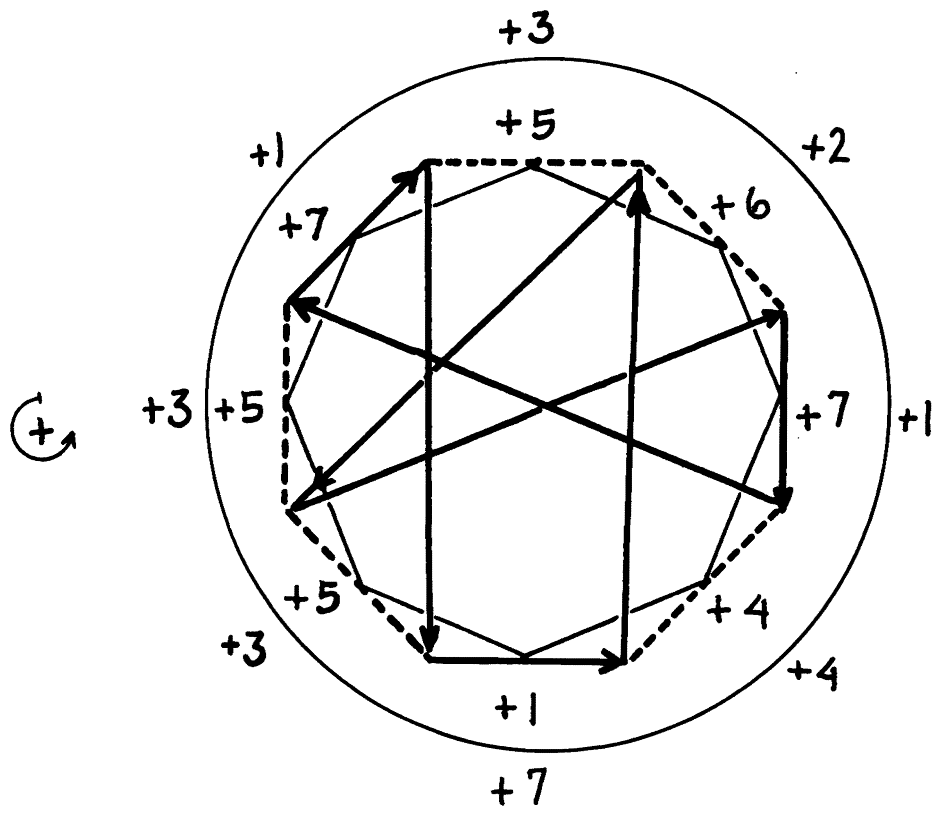
```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FLEXAGON
(((+),(+),(+,+),(+,+,+,+),(+,+)))


END OF APPLY, VALUE IS ...
((I,O,X,X,X),(X,I,O,X,X),(X,X,I,X,O),(X,X,X,O,I),(O,X,X,I,X))
```



```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FLEXAGON
((( + ),( +,+),( +),( +,+,+),( +,+,+)))


END OF APPLY, VALUE IS ...
((I,O,X,X,X),(X,I,X,O,X),(X,X,X,I,O),(X,X,O,X,I),(O,X,I,X,X))
```



```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FLEXAGON
(((+,+,+),(+,+,+),(+,+,+),(+,+,+),(+,+,+)))


END OF APPLY, VALUE IS ...
((I,X,X,O,X),(X,O,X,I,X),(X,I,X,X,O),(X,X,O,X,I),(O,X,I,X,X))
```



```
 FUNCTION  APPLY(F,X,P) HAS BEEN ENTERED, ARGUMENTS..
FLEXAGON
((((+),(+,+,+),(+,+,+),(+,+,+,+,+,+,+),(+,+,+,+),(+),(+,+),(+,+,+)))


END OF APPLY, VALUE IS ...
 ((I,0,X,X,X,X,X,X),(X,I,X,X,0,X,X,X),(X,X,X,X,I,X,X,0),(X,X,X,X,X,X,0,I),
(X,X,0,X,X,X,I,X),(X,X,I,0,X,X,X,X),(X,X,X,I,X,C,X,X),(C,X,X,X,X,I,X,X))
```

INDEX

(Not including special functions or their satellites)

***