MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Vision Memo. No. 119

Memo.MAC-M-342
January 1967

A PRIMITIVE RECOGNIZER OF FIGURES IN A SCENE

Adolfo Guzmán.

Given a scene, as seen for instance from a T.V. camera or a picture, it is desired to analyze it to recognize, differentiate and identify desired objects or classes of objects (i.e., patterns) in it.

The present report describes a program, written in CONVERT, which partially achieves this goal. Two inputs to the program determine its behavior and response:

1. The scene to be analyzed, which is entered in a symbolic format (it may contain 3-dimensional and curved objects).
2. A symbolic description -- called the model -- of the class of the objects we want to identify in the scene (1):

Given a set of models of the objects we want to locate, and a scene or picture, the program will identify in it all those objects or figures which are similar to one of the models, provided they appear complete in the picture (i.e., no partial occlusion or hidden parts). Recognition is independent of position, orientation, size etc.; it strongly depends on the topology of the model.

Important restrictions and suppositions are: (a) the input is assumed perfect -- noiseless-- and highly organized; (b) more than one model is in general required for the description of one object and (c) only objects which appear unobstructed are recognized.

Work is continuing in order to drop restriction (c) and to improve (a).

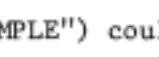
SUMMARY

The Problem.

A picture, scene or view, is read with the help of an optical device and stored as an array of light intensities in the memory of the computer. The ultimate goal will be to understand this information, that is, to identify, separate and position the different objects or bodies belonging to the scene(s). The demands of information will vary: sometimes we will be interested in knowing if object A is seen in the scene or not; while at other times we may require a complete description of the scene, including information on relative support and (3-d) position of the different components. Hence it is clear that the recognizer will need an additional input to specify the nature of the question that the program is to answer by analyzing the scene.

The Proposed Solution.

During the summer of 1966 this general problem was attacked (1) by a group of people, the so-called Vision Group, with results reported here and elsewhere.(2-5) The work here described is only a part of the total effort, as we will see below. We divide the system in two phases: "Preprocessing" and "recognition".

Preprocessing - A program (6) sweeps the array containing the scene and collects sets of points satisfying a given predicate; these sets are called regions, and roughly correspond to the different faces of the objects. Thus, in Fig. "EXAMPLE", some regions are A,B,C,...,L,Z. It is entirely possible, but it is undesirable, that two or more regions will be reported as one, for instance, A and B (cf. Fig. "EXAMPLE") could be reported as  ; similarly, depending on the chosen predicate, the long region D could be reported as two shorter ones, if the predicate were "equal

brightness", for instance.

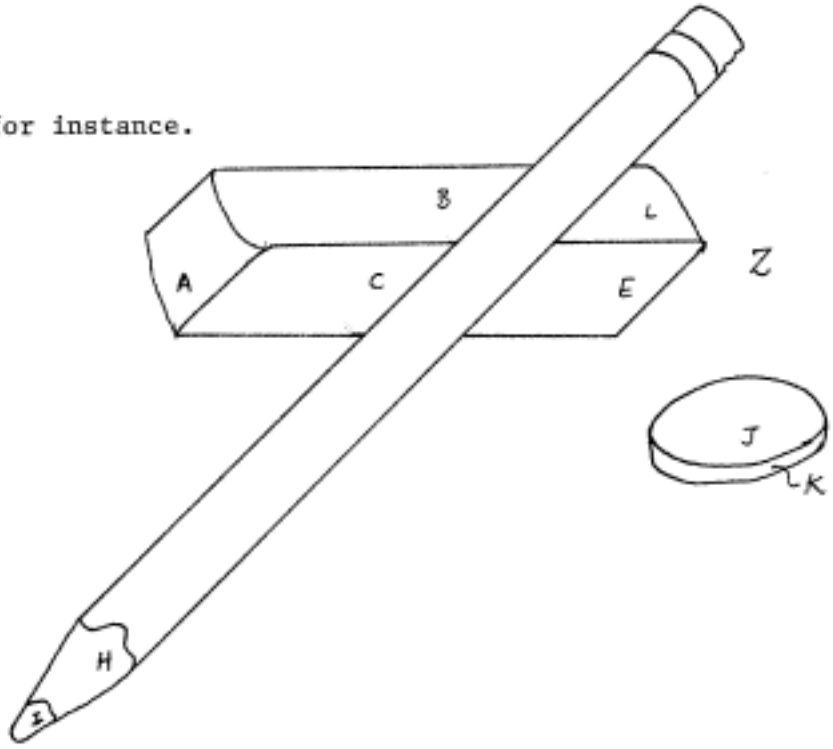


FIG. 'EXAMPLE' - A scene.

Another program[5] not yet debugged will drive the region-finder, supplying 'good' predicates; the boundaries will be sorted and "smoothed", and the bad regions eliminated and/or merged.

A further preprocessing is (or will be) then done[3,4], interpolating straight lines, segments of curves, etc., until finally each region is (or will be) described by a set of properties; for instance, in Fig. 'EXAMPLE',

Region A will be:

NEIGHBORS	C, B, Z
CENTER	(10, 27.5)
SHAPE	(\ ,/ ,\ ,/)

This input is the one which the recognizer uses.

It is understood that finding properly meaningful regions is non-trivial and that as the system develops we expect this part to

become more involved with feedback from the higher-level recognition programs.

Model. - This is a description, in a particular symbolic format, of an object. Models represent objects regardless of position, shape, orientation, etc., unless specifically specified otherwise. For instance, the model 'PARALLELEPIPED' in figure 'PARALLELEPIPED' is written as:

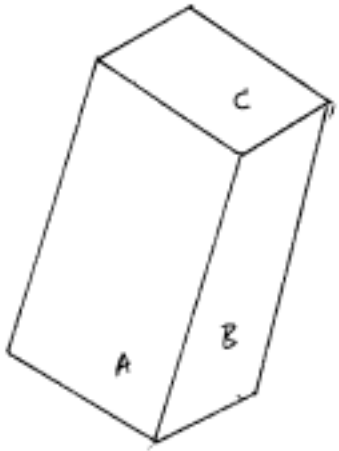


FIG. 'PARALLELEPIPED'. A Model.

```
((A* (NEIGHBOR B*)
     (NEIGHBOR C*)
     (SHAPE PARALLELOGRAM))

 (B* (NEIGHBOR A*)

     (NEIGHBOR C*)
     (SHAPE PARALLELOGRAM))

 (C* (NEIGHBOR A*)
     (NEIGHBOR B*)
     (SHAPE PARALLELOGRAM)) )
```

A model is composed of regions, with properties inter-relating them. Given an object, there is a large number of models which correctly describe it.

Recognition. - This report describes the recognizer, i.e., a program which, given a scene (such as 'EXAMPLE2') and a model (such as 'PARALLELEPIPED'), will identify all 'parallelepipeds' present in 'example2'. In this case, parallelepipeds 1 and 3 are found, parallelepiped 2 is parially hidden and is not recognized. Both the scene and the model are in symbolic format.

FIG.'EXAMPLE2'. Three parallelepipeds.

Restrictions: In this first experimental system we will live with the following constraints:

1. Noiseless data is supposed, i.e., the scene must be accurately described by its symbolic representation. Also, the set of shapes assumed is small, so that we need not worry about heuristic efficiency in algorithms.
2. Whenever a 3-dimensional object gives rise to several (2-dim) projections which are topologically different, all these need to be presented as models in order to cover the possible cases. The recognizer has an OR feature for this effect. For instance, Fig. 'L' has the same object in four different positions, requiring 3 or possibly 4 models of an 'L' to identify all. The exact number depends on the particular models in question and their "don't-care" conditions, which may depend on what other objects in the world have to be distinguished.
3. Only objects which are totally seen are recognized. Partially occluded or hidden parts or bodies may be present in the picture but the occulted objects will not be identified. For instance, parallelepiped 2 in fig. 'EXAMPLE2' was not found. Our current work will help to relax this last restriction, and also restriction (1). The reader unfamiliar with progress in that direction can see Refs. (10) for some earlier work of that kind.

The same object in four different positions, all of which differ in the topology of its two dimensional projection over the plane of the drawing.

4. In the present program we assume orthogonal projections. Later we will consider finite perspective. For small visual angles, a simple tolerance should suffice for most cases, but for large visual angles we will have to use other methods.

THE SCENE

Informally, a scene (picture) is a collection of regions,(projections of faces); a region is described by an ordered collection of segments (lines or curves), and these have several properties.

A scene is represented by an atom which has under the entry 'REGIONS' a list of the regions composing it; for instance (see fig. 'BOTTLES'), the atom BOTTLES is a scene for which

(GET (QUOTE BOTTLES) (QUOTE REGIONS)) = (A B C D E F G H I J K L M Z)

In this case the regions of 'bottles' are A,B, ...,M,Z.

A region is an atom which has in its property list the entries NEIGHBOR, SHAPE, and possibly others. A region corresponds to a surface or face in the scene, except that it is treated 2-dimensionally; i.e., in fig. 'EXAMPLE2', the upper face of the eraser A B C E L is composed of 2 regions, namely B and L.

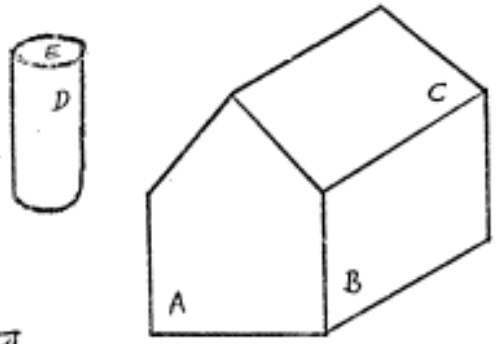
NEIGHBOR	(L Z)	(L and Z are limitrophe regions with M)
SHAPE	ELLIPSE	

At present, the shapes of regions can only be atoms; this is a severe restriction since it may be too much to require that the preprocessor recognize region M (fig. 'BOTTLES') as an ellipse or region A (fig. 'EXAMPLE') as a parallelogram. In the models, the shapes are also

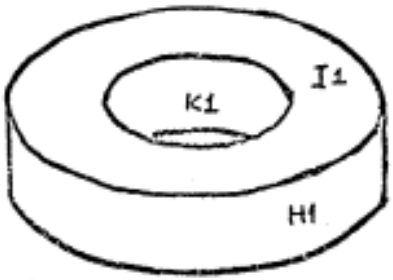
atoms. This restriction will be abandoned eventually, but now is observed.

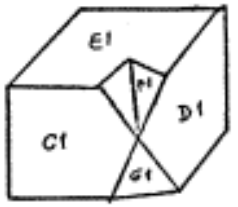
Complete example of a Scene. - The scene 'GREEN' (see fig. 'GREEN') is entered into memory as follows:

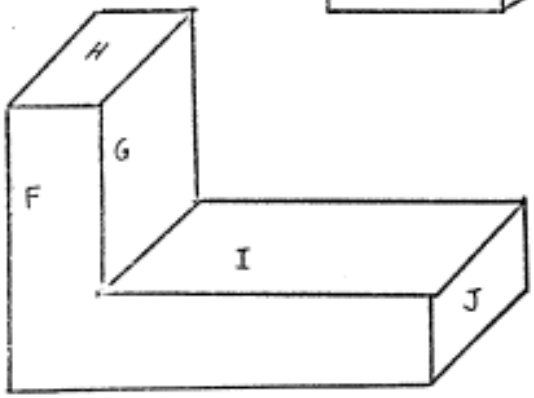
```
(DEFPROP A (B C M1) NEIGHBOR)
(DEFPROP B (A C M1) NEIGHBOR)
(DEFPROP C (A B M1) NEIGHBOR)
(DEFPROP D (E M1)   NEIGHBOR)
(DEFPROP E (D M1)   NEIGHBOR)
(DEFPROP F (G I J H M1)NEIGHBOR)
(DEFPROP G (F I H M1)NEIGHBOR)
(DEFPROP H (F G M1) NEIGHBOR)
(DEFPROP I (F G J M1)NEIGHBOR)
(DEFPROP J (F I M1) NEIGHBOR)
(DEFPROP K (N M1)   NEIGHBOR)
(DEFPROP L (N M M1) NEIGHBOR)
(DEFPROP M (L N M1) NEIGHBOR)
(DEFPROP N (O P L M K M1)NEIGHBOR)
(DEFPROP O (P N M1) NEIGHBOR)
(DEFPROP P (N O)    NEIGHBOR)
(DEFPROP Q (S T R M1)NEIGHBOR)
(DEFPROP R (Q S M1) NEIGHBOR)
(DEFPROP S (Q R T M1)NEIGHBOR)
(DEFPROP T (Q S M1) NEIGHBOR)
(DEFPROP U (V X B1 M1)NEIGHBOR)
(DEFPROP V (B1 U M1)NEIGHBOR)
(DEFPROP W (X B1 M1)NEIGHBOR)
(DEFPROP X (U W B1 M1)NEIGHBOR)
(DEFPROP Z (B1 M1)  NEIGHBOR)
(DEFPROP A1 (B1 M1) NEIGHBOR)
(DEFPROP B1 (A1 V U X W Z) NEIGHBOR)
(DEFPROP C1 (E1 F1 G1 D1 M1)NEIGHBOR)
(DEFPROP D1 (C1 F1 E1 G1 M1)NEIGHBOR)
(DEFPROP E1 (M1 C1 F1 D1)  NEIGHBOR)
(DEFPROP F1 (C1 D1 E1)     NEIGHBOR)
(DEFPROP G1 (C1 D1 M1)     NEIGHBOR)
(DEFPROP H1 (I1 M1)        NEIGHBOR)
(DEFPROP I1 (H1 K1 M1)     NEIGHBOR)
(DEFPROP K1 (I1 M1)        NEIGHBOR)
(DEFPROP A  PENTAGON SHAPE)
(DEFPROP B  PARALLELOGRAM SHAPE)
(DEFPROP C  PARALLELOGRAM SHAPE)
(DEFPROP D  (I C I D) SHAPE)
(DEFPROP E  ELLIPSE SHAPE
(DEFPROP F  ELE SHAPE)
(DEFPROP G  PARALLELOGRAM SHAPE)
(DEFPROP H  PARALLELOGRAM SHAPE)
(DEFPROP I  PARALLELOGRAM SHAPE)
(DEFPROP J  PARALLELOGRAM SHAPE)
(DEFPROP K  SPHERE SHAPE)
(DEFPROP L  TRIANGE SHAPE)
(DEFPROP M  TRIANGLE SHAPE)
(DEFPROP N  FUNNY SHAPE)
(DEFPROP O  PARALLELOGRAM SHAPE)
(DEFPROP P  PARALLELOGRAM SHAPE)
(DEFPROP Q  PENTAGON SHAPE)
```

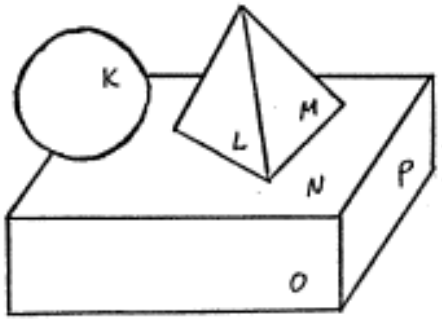


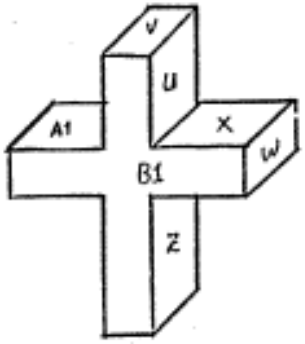
M1











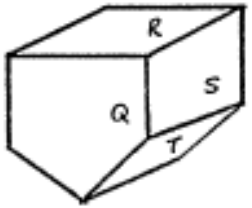


FIG. 'GREEN'.

```
(DEFPROP R  PARALLELOGRAM SHAPE)    (DEFPROP B1  CROSS SHAPE)
(DEFPROP S  PARALLELOGRAM SHAPE)    (DEFPROP C1  PENTAGON SHAPE)
(DEFPROP T  PARALLELOGRAM SHAPE)    (DEFPROP D1  PENTAGON SHAPE)
(DEFPROP U  PARALLELOGRAM SHAPE)    (DEFPROP E1  FUNNY SHAPE)
(DEFPROP V  PARALLELOGRAM SHAPE)    (DEFPROP F1  QUADRILATERAL SHAPE)
(DEFPROP W  PARALLELOGRAM SHAPE)    (DEFPROP G1  QUADRILATERAL SHAPE)
(DEFPROP X  PARALLELOGRAM SHAPE)    (DEFPROP H1  (I C I D) SHAPE)
(DEFPROP Z  TRAPEZIUM SHAPE)        (DEFPROP I1  (ELLIPSE INSIDE ELLIPSE)SHAPE)
(DEFPROP A1 TRAPEZIUM SHAPE)        (DEFPROP K1  FUNNY SHAPE)
```

THE MODEL

A model is an atom which contains in its property list, under the entry 'REGIONS', a list of the following form:

a) the 1st. element of such a list is an atom, the name of the region as far as the model is concerned

b) each of the remaining elements of such a list is a property; specifically, is either a list (NEIGHBOR ..)
or a list (SHAPE..)

-- more properties will be used when objects start getting more complicated--

Example: The model 'HOUSE' (see fig. 'HOUSE') is written in this way:

HOUSE

(in its property list, we find:)

```
REGIONS      ((A* (NEIGHBOR B*)  (NEIGHBOR C*) (SHAPE PENTAGON)
               (B* (NEIGHBOR A*)  (NEIGHBOR C*) (SHAPE PARALLELOGRAM)
               (C* (NEIGHBOR A*)  (NEIGHBOR B*) (SHAPE PARALLELOGRAM)) )
```

FIG. 'HOUSE'. - A model.

What this list means is that HOUSE is composed of three regions, namely A*, B* and C*; and that A* is neighbor of B* and C*, etc. More over, it says the shapes of A* (pentagon), B* (parallelogram) and C* (parallelogram). Additional properties could be inserted here. The names A*, B*, etc., given to the different faces, have no importance, they act as dummy variables (UAR or 'undefined' variables in CONVERT[7]); the names such as PARALLELOGRAM, PENTAGON etc., given to the shapes, are crucial, since they are going to be compared by equality with the corresponding names in the property list of the regions of the scene.
Note that the models we are using are not "categorical" - they don't contain enough information (usually) to reconstruct the object.

Example.- PYRAMID (see fig. 'PYRAMID'). This model is written as:

```
(DEFPROP PYRAMID
((A*  (NEIGHBOR B*)
      (SHAPE TRIANGLE))
 (B*  (NEIGHBOR A*)
      (SHAPE TRIANGLE)))   REGIONS)
```

but also see figure 'PYRAM1'

FIG. 'PYRAMID

The following examples of models will show in general how to build or define models.

```
(PUTPROP (QUOTE CYLINDER)
(QUOTE ((A* (NEIGHBOR B*) (SHAPE ELLIPSE)) (B* (NEIGHBOR A*)
                                              (SHAPE (I C I D)))))

(QUOTE REGIONS))
```

FIG. 'CYLINDER' A Model.

Remark: Note how we describe B*'s shape as (SHAPE (I C I D)), i.e., as (straight, convex, straight, concave).

```
(DEFPROP  HOLLOWCYLINDER
  ((A*  (NEIGHBOR B*)    (SHAPE ELLIPSE)  )
   (B*  (NEIGHBOR A*)  (NEIGHBOR C*) (SHAPE(ELLIPSE INSIDE ELLIPSE)) )
   (C*  (NEIGHBOR B*)  (SHAPE (I C I D)) ) )   REGIONS)
```

FIG. 'HOLLOWCYLINDER'.
A Model.

```
 (PUTPROP (QUOTE CUBE)  (QUOTE)
((C* (NEIGHBOR E*)
    (NEIGHBOR D*)
    (SHAPE PARALLELOGRAM))
 (D* (NEIGHBOR C*)
    (NEIGHBOR E*)
    (SHAPE PARALLELOGRAM))
 (E* (NEIGHBOR D*)
    (NEIGHBOR C* ) (SHAPE PARALLELOGRAM)  )  ))
(QUOTE REGIONS))
```

FIG. 'CUBE' A Model.
It is really a parallelepiped.

```
(DEFPROP  HOLLOWBRICK)
  ( (D* (NEIGHBOR E*) (NEIGHBOR F*)(NEIGHBOR G*)(NEIGHBOR H*)
        (SHAPE (PARALLELOGRAM INSIDE PARALLELOGRAM)) )
    (E* (NEIGHBOR U*) (NEIGHBOR F*) (SHAPE TRAPEZ)  )
    (F* (NEIGHBOR E*)  (NEIGHBOR D*) (SHAPE TRAPEZ) )
    (G* (NEIGHBOR D*)  (NEIGHBOR H*) (SHAPE PARALLELOGRAM))
    (H* (NEIGHBOR D*)  (NEIGHBOR G*) (SHAPE PARALLELOGRAM)) )
 REGIONS)
```

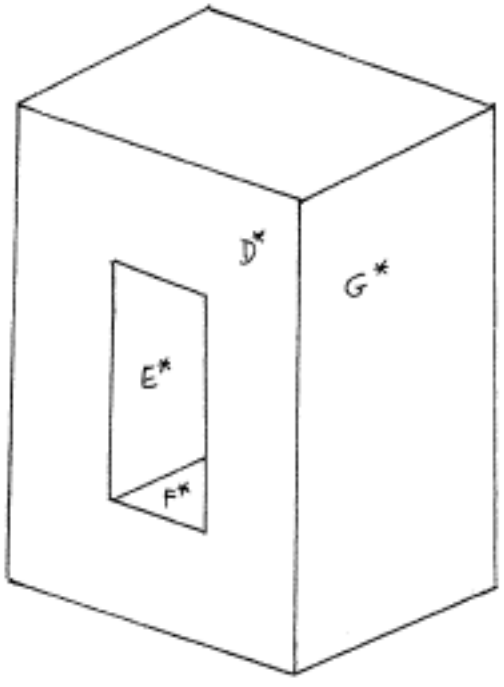


FIG. 'HOLLOWBRICK'.
A Model.

This page is missing from the original document.

THE RESULTS

We will present now several examples of scenes analyzed by DT, the program, in the PDP-6 computer. The symbol # marks the lines typed by the user.

```
@   CONV 4                                          Bring the CONVERT processor
                                                    from tape 4.

#   (UREAD DT LISP 5 ↑Q ↑W)                         Load the file containing DT,
                                                    the recognizer.

#   (UREAD EX2 LISP  ↑Q  ↑W)                        Bring the scene EX2 into
                                                    memory. (see fig. 'EX2').

#   (UREAD MOD2 LISP ↑Q  ↑W)  (IOC V)               Load the models.
    (v)

#   (DT (QUOTE CUBE) (QUOTE EX2))                   Look for 'CUBEs' in'EX2'.
    (CUBE 1. IS (A B C))                            (See fig. 'EX2').
    (CUBE 2. IS (J L M))                            2 cubes are found.
    (D E F G H I K N O P Q R S T U V W X Y Z)  Remaining of scene.

#   (DT (QUOTE CYLINDER) (QUOTE EX2))               Look for cylinders (see fig.
                                                    'CYLINDER').
    (CYLINDER 1. IS (E D))
    (CYLINDER 2. is (G F))
    (A B C H I J K L M N O P Q R S T U V W X Y Z)  ←remaining of scene.

#   (DT (QUOTE HOLLOWCYLINDER) (QUOTE EX2))
    (HOLLOWCYLINDER 1. IS (T U S))
    (A B C D E F G H I J K L M N O P Q R V W X Y Z)

#   (DT (QUOTE HOLLOWBRICK) (QUOTE EX2))
    (HOLLOWBRICK 1. IS (N O P Q R))                      ←see fig.'HOLLOWBRICK'.
    (A B C D E F G H I J K L M S T U V W X Y Z)

    Since we are tired to write 'QUOTE', we define DD:

#   (DEFPROP DD (LAMBDA (A)  (DT (CAR A) (CADR A)))  FEXPR)
    DD

#   (DD HOLLOWBRICK EX2)                            Compare with above.
    (HOLLOWBRICK 1. is (N O P Q R))
    (A B C D E F G H I J K L M S T U V W X Y Z)   Good. Let us see something
                                                                    else.
```

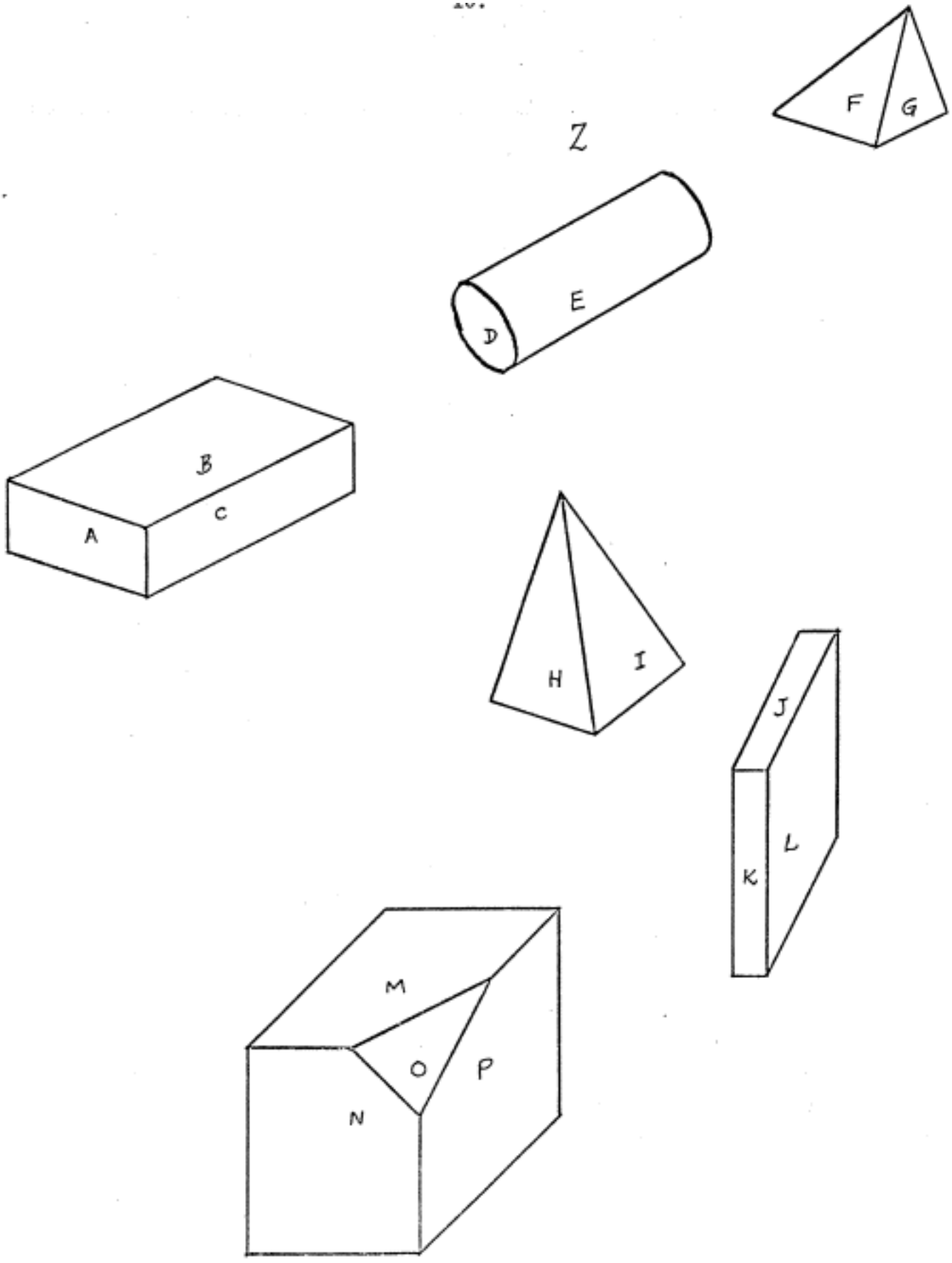


Fig. 'FIG1' - A scene

Now we begin to examine the scene FIG1 (see fig. 'FIG1').
In this figure we are going to look for the presence of the following objects: CUBEs, CYLINDERs, PYRAMIDs, HOLLOWCYLINDERs, HOLLOWBRICKs.
(See figures corresponding to those models).

```
#   $$                                          Out of TECO
#   4$
#   CONV 4                                      Go to CONVERT
#   (UREAD EX4 LISP 5)
    5                                           Load scene
#   ↑Q
    FIG1

    A

    B
#   ↑W↑V (UREAD DT LISP)                        Load program (recognizer)
    5
#   ↑Q
    EQUAL*

#   PR↑W↑V (UREAD EX3 LISP)                     Load models
    5
    ↑Q
    QU
#   ((A* ↑W↑V
#   (DT (QUOTE CUBE) (QUOTE FIG1))              Look for CUBEs in FIG1.

    (CUBE 1  IS (A B C))
    (CUBE 2 IS (J K L))
    (D E F G H I M N O P Z)
#   (DT (QUOTE PYRAMID )  (QUOTE FIG1))
    (PYRAMID 1 IS (F G))                        2 pyramids are found.
    (PYRAMID 2 IS (H I))                        (see also fig. 'PYRAMID').
    (A B C D E J K L M N O P Z)

#   (DT (QUOTE CYLINDER) (QUOTE FIG1))
    (CYLINDER 1 IS (D E))                       Cylinder D E is located.
    (A B C F G H I J K L M N O P Z)
#   (DT (QUOTE HOLLOWCYLINDER) (QUOTE FIG1))
    (A B C D E F G H I J K L M N O P Z)         No hollowcylinders are
                                                present.
#   (DT (QUOTE HOLLOWBRICK) (QUOTE FIG1))
    (A B C D E F G H I J K L M N O P Z)         DT could not find any HOLLOWBRICK
                                                (see fig. 'HOLLOWBRICK') in scene
                                                FIG1 (see fig. 'FIG1').
#   (MACDMP)
#   1$                                          Go back to MACDMP and tape 1.
```

This page is missing from the original document.

We analyze now FIG2 (see fig. 'FIG2').

```
#   (DD PYRAMID FIG2)                          Looking for PYRAMIDs
                                               (see model in fig. 'PYRAMID')
    (PYRAMID 1 is (L M))
    (PYRAMID 2 is (X Y))
    (A B C D E F G H I J K N O P Q R S T U V W Z)
```

Note that pyramid L M N is not reported as such, but only L M is reported or recognized. Why is this? Because the model 'PYRAMID' (see fig. 'PYRAMID') is composed of <u>two</u> triangles. Also, it is in the nature of our algorithm that L M prevents recognizing M N. In order to get L M N, we define PYRAM1 as a pyramid which has <u>three</u> visible triangular faces:

```
#   (DEFPROP PYRAM1
#    ((A* (NEIGHBOR B*) (SHAPE TRIANGLE))
#     (B* (NEIGHBOR A*) (NEIGHBOR C*) (SHAPE TRIANGLE))
#     (C* (NEIGHBOR B*) (SHAPE TRIANGLE)) )          REGIONS)
```

PYRAM1
See fig. 'PYRAM1'.

Now we apply this model to scene FIG2:

```
#   (DD PYRAM1 FIG2)
    (PYRAM1 1 IS (L M N))                          Aja. Only 1 is found.Correct
    (A B C D E F G H I J K O P Q R S T U V W X Y Z)  (only one pyramid with three
                                                     visible faces is present in
                                                                        FIG2)
```

What we really want is to define a pyramid as something which shows either two or three triangular faces; so,

```
#   (DEFPROP PYR    (OR PYRAMID PYRAM1)  REGIONS)
    PYR                                        -- the last model, PYRAM1 in
                                              this case, is searched first--
```

Note the use of the OR in a model.
At this moment, PYR is a model which stands for either  or 

```
#   (DD PYR FIG2)
    (PYRAM1 is  (L M N))
    (PYRAMID 1 IS (X Y))
    (A B C D E F G H I J K O P Q R S T U V W Z)
```

Good. 2 objects were found to match with PYR: (L M N) AND (X Y). (see fig. 'FIG2').

What would have happened if we define PYR in the reverse order?

Let us define

```
#   (DEFPROP PYR  (OR PYRAM1  PYRAMID)   REGIONS))
    PYR
```

The last model in OR list, PYRAMID in this case, is searched first. The answer is:

```
#   (DD PYR FIG2)
    (PYRAMID 1 IS (L M))
    (PYRAMID 2 IS (X Y))
    (A B C D E F G H I J K N O P Q R S T U V W Z)
```

Two objects matched with PYRAMID: (X Y) and (L M); after this, no object was found to match with PYRAM1.

Conclusion: Order in the models is important, so long as we leave things to the normal CONVERT matching algorithm.

```
#   (DD PYR FIG3)
    NIL
```

FIG3 is an empty scene.

```
#   (DD CYLINDER  FIG2)
    (A B D D E F G H I J K L M N O P Q R S T U V W X Y Z)
```

No cylinders. Cylinder P O is partially occulted, so is not found.

```
#   (DD CUBE FIG2)
    (CUBE 1 IS (I J K))
    (A B C D E F G H L M N O P Q R S T U V W X Y Z)
```

FIG2 contains one cube.

```
#   (DD ANGLE FIG2)
    (ANGLE 1 is (D A B C))
    (E F G H I J K L M N O P Q R S T U V W X Y Z)
```

Angle is a model described in the next page (see fig. 'ANGLE').

Angle Q V R T U was not found because has a different form (its

two dimensional projection has a different topology from model 'ANGLE'; namely has 5 faces or regions and model 'ANGLE' has only 4).

Angle E F G H was not found because it is partially occulted.

```
#   (DD SPHERE FIG2)
    (SPHERE 1 is (S))                          Two spheres in FIG2.
    (SPHERE 2 is (W))
    (A B C D E F G H I J K L M N O P Q R T U V X Y Z)
        W ?? This is a surprise!  Let us see what kind of region W is:

#   (GET (QUOTE W) (QUOTE NEIGHBOR))

    NIL
#   (GET (QUOTE W) (QUOTE SHAPE))
    NIL
       Aja! W has no neighbors, and has no shape. Why does it match
    (SHAPE CIRCLE)? (see fig. 'FIG2').  There is no region named W.
    This was an error in the data; instead of
    (DEFPROP FIG2 (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) REGIONS)
    we should have said:
    (DEFPROP FIG2 (A B C D E F G H I J K L M N O P Q R S T U V X Y Z)  REGIONS)
    (DEFPROP ANGLE
    ((A* (NEIGHBOR B*) (SHAPE FUNNY))
     (B* (NEIGHBOR A*)  (NEIGHBOR C*)  (NEIGHBOR D*)  (SHAPE ELE))
     (C* (NEIGHBOR B*)  (NEIGHBOR D*)  (SHAPE PARALLELOGRAM))
     (D* (NEIGHBOR B*)  (NEIGHBOR C*) (SHAPE PARALLELOGRAM)) )
                REGIONS)
```

This page is missing from the original document.

THE PROGRAM

The results just presented were obtained in the PDP-6 computer, using the CONVERT[7] processor. This last is imbebbed in LISP.
DT is the main function, it has two arguments: the scene and the model.

DT handles the (OR ...) feature, constructs a CONVERT pattern from the given model and calls to the CONVERT processor to deal with it. DT, therefore, works much in the way of a compiler, transforming a MODEL into a PATTERN; in this form, we avoid the double scanning that would originate if we look at the model in order to understand what does it require, then look at the scene to try to meet the requirement; once done this, come back to the model to see what else is necessary; go again to the scene to look for that feature etc.

There is a marked similarity between DT and TD [9], and the reader is referred to the report[9] on TD, which talks in detail the way the generated pattern drives the match and the search. See also appendix.

The program is:

```
(DEFPROP DT(LAMBDA (MODEL SCENE) (COND
           ((AND (COND ((ATOM SCENE) (SETQ SCENE
                       (GET SCENE (QUOTE REGIONS))))(T NIL) ) NIL)NIL)
          ((AND
            (ATOM MODEL) (SETQ  W MODEL)
           (NULL (SETQ MODEL (GET MODEL (QUOTE REGIONS)))) )SCENE)
          ((EQ (CAR MODEL) (QUOTE OR)) (COND (NULL (CDR MODEL)) SCENE)
                  (T (DT (CADR MODEL) (DT (CONS (CAR MODEL) (CDDR MODEL)) SCENE)))
))((DEFPROP NU (O) PROEXP)
 (CONVERT (APPEND (PREPARE MODEL)
                  (QU (\C PAT   (EV (CONSTRUCT MODEL))
                         MODEL EXPR   (EV W)
                         SCENE
                         EXPR
                         (EV SCENE )))
```

```
                (QUOTE (    \X UAR NIL
                            USED BUV  ==
                            NUM SKEL  (=SETQ= NUM (=INCR= NU)))))
    NIL
                  SCENE
    (QUOTE (C1 (( C  (=PROG= NIL
                             (=PRNT= (MODEL NUM IS USED))
                             (=SKEL= SCENE
                                     EXPR
                                      (=COMP= SCENE USED)
                                     (=RETN= (=REPT= SCENE C1)
                                 )) )))))))))
EXPR)
```

The dictionary M of CONVERT is increased with (PREPARE A), where A is the model. A pattern is generated from the model with the help of (CONSTRUCT A), and then is baptized with the name /C.

The 4th argument of CONVERT consists only of one set of rules, C1, with only one rule which says:

```
[/C  (=PROG= ()
             (=PRNT= (MODEL NUM IS  USED))
             (=SKEL= SCENE  EXPR  (=COMP= SCENE USED)
                     (=RETN=  (=REPT= SCENE C1)) ))]
```

"If a match is found between the scene and /C (/C is the generated pattern), print '(MODEL NUM IS USED)', for instance, (CUBE 5 IS (A B C)); rebind SCENE to the remaining of the SCENE --the complement of SCENE and (A B C)--; and repite over this again." The =PROG=ram feature is explained in [8].

We will see now the auxiliary functions.

```
(PREPARE A)

(DEFPROP PREPARE (LAMBDA (A)
(COND ((NULL A)NIL)
      (T (CONS  (CAAR A)
                (CONS (QUOTE PAV)
                      (CONS  (CONS  (QUOTE =AND=)
                                    (MAPCAR (FUNCTION (LAMBDA (U)
                                                              (QU (=XEC=
(COMPARE (EV (APPEND U (LIST (LIST (QUOTE =QUOT=) (CADR U)))))
         =SAME= ))))  (CDAR A)))
                                (PREPARE (CDR A))))))))EXPR)

Explanation.- If

MODEL = ( (A* (NEIGHBOR B*) (NEIGHBOR C*) (SHAPE TRIANGLE))
          (B* (   ...     ) (   ...     ) (      ...     ))
                            ......                        )

then

(PREPARE A) = (A* PAV (=AND= (=XEC= COMPARE (NEIGHBOR B* (=QUOT=B*)) =SAME=)
                             (=XEC= COMPARE (NEIGHBOR C* (=QUOT=C*)) =SAME=)
                             (=XEC= COMPARE (SHAPE TRIANGLE (=QUOT= TRIANGLE))
                                                                    =SAME=))
              B* PAV (=AND= (=XEC= ...) (=XEC= ...))
                               .....                                        )

(DEFPROP EQUAL *(LAMBDA (A B) (OR (NULL B) (EQUAL A B)) ) EXPR)

(CONSTRUCT A)

(DEFPROP CONSTRUCT (LAMBDA (A)
(LIST (QUOTE =UNO=
      (NCONC (MAPCAR (FUNCTION (LAMBDA (B) (QU (=AND= (EV (CAR B))USED))))A)
             (QUOTE (===))))) EXPR)

    (CONSTRUCT MODEL)= (=UNO= ((=AND= A* USED) (=AND= B* USED) (=AND= C* USED)===))

   (COMPARE A B)
               A has the form (NEIGHBOR B* (=QUOT= B*))
                              OR (SHAPE TRIANGLE (=QUOT= TRIANGLE))
               B has the form D5
                              OR N  (is the name of a region of the scene).
```

```
(DEFPROP COMPARE (LAMBDA (A B)
(COND ((EQ (CAR A) (QUOTE NEIGHBOR)) (OR (EQ (CADR A)(CADDR A))
                                         (MEMBER (CADR A)(GET B(CAR A)))))
      (T (EQUAL* (CADR A) (GET B (CAR A)))))) EXPR)
```

value is
- if (car A) = NEIGHBOR
  - T if (cadr A) = (caddr A) --region not yet found-
  - otherwise
    - T if (cadr A) (list of neighbors of B)
    - F otherwise
- otherwise
  - T if B is empty
  - (EQUAL A B) otherwise

(QU A)

QU quotes its argument. Quotes all the elements of its argument, except those which have (EV ..) in front of it. For instance, if A = 35, B = 36, then

```
(QU (A B B (EV B) () (( C A M O T E S))  (( A (EV A) C (EV A) )) 59 ))  is
    (A B B   36   () (( C A M O T E S))  (( A  35    C  35    )) 59 )).
```

```
(DEFPROP QU (LAMBDA (A L)
  (COND ((ATOM (CAR A))(CAR A))
        ((EQ (CAAR A) (QUOTE EV))(EVAL (CADAR A)L))
        (T (MAPCAR (FUNCTION (LAMBDA (K)
                                     (EVAL (LIST (QUOTE QU)
                                                 K)
                                           L)))
                   (CAR A)))))

FEXPR)
```

Those are all the functions we need.

REFERENCES

1. Papert,S. The Summer Vision Project. A.I. Memo 104-1; Jul 66. (Project MAC, MIT)

2. Guzman,A. and Sussman,G. A Quick Look to Some of Our Programs. A.I. Internal Memo V-104-1; Jul 66.
3. White,J. Additions to Vision Library. A.I. Internal Memo V-104-2; Aug 66.
4. White,J. Figure Boundary Description Routines for the PDP-6 Vision Project. A.I. Memo No. 104-5 Vision (MAC-M-328); Sept 66.
5. Lamport,L. Summer Vision Programs. A.I. Memo No. 104-6 Vision (MAC-M-332).Oct 66.
6. Sussman, G. Unwritten report. See also (2).
7. Guzman,A. and McIntosh,H.V. CONVERT. Communications of the A.C.M.,9, 8 (aug 66). pp. 604-615; also available as A.I. Memo 99(MAC-M-316) June 66.
8. Guzman,A. and McIntosh,H.V. A Program Feature for CONVERT. A.I. Memo 95 (MAC-M-305). April 66.
9. Guzman A. Scene Analysis Using the Concept of Model. TR Report 12, Computer Corporation of America,Cambridge, Mass. (in preparation).
10. Guzman,A. POLYBRICK: Adventures in the domain of parallelepipeds. **MAC-M-308 (AI Memo 96) Project MAC, MIT (may 1966).**

L.G. Roberts. Machine Perception of Three Dimensional Solids. Optical and Electro-optical Information Processing. M.I.T. Press.

T. Marill et al. CYCLOPS-1: A second-generation recognition system. Proc. AFIPS Fall Joint Computer Conference, 27-33 (1963).

R.H. Canaday. The description of overlapping figures. M.S. Thesis, Electrical Engineering Dept. M.I.T. 1962.

11. A more general discussion of scene analysis is given in Guzman,A. Some Aspects of Pattern Recognition by Computer. M.S. Thesis, Electrical Engineering Dept. M.I.T. (February 1967).

APPENDIX

UNORDERED PATTERNS IN CONVERT

A UNO (P1 P2 ...Pn) — A is an atom (not a fragment) in the mode UNO; the A_i are patterns (expressions, not fragments).

A will match with an expression E if at least one of the patterns P_i matches with such an E. That being the case, the successful pattern is removed from the associated expression in the dictionary and this entry in the dictionary becomes

A UNO (P1 P2 .. P_{i-1} P_{i+1} ... Pn)

Therefore, if several A's occur in a pattern, each one represents a different Pi to be matched.

Example: If A is defined in the dictionary as A UNO (1 4 2), then a pattern such as (K A M A A) will match:

	(K 4 M 1 2)
and	(K 1 M 2 4)
and	(K 1 M 4 2)
will not match	(K 1 M 1 2)
or	(K 1 M 4 3)
or	(K 4 1 2 M)

and the pattern (A A A A) will not match anything, since A has only 3 patterns available in the associated expression (in the dictionary), and (A A A A) requires 4.

Compare with =OR=

pattern
X is (K A M A A)

expression E	dictionary L → A UNO (1 4 2)	A PAT (=OR= 1 4 2)
	Is a match obtained?	Is a match obtained?
(K 4 M 1 2)	yes	yes
(K 1 M 2 4)	yes	yes
(K 1 M 4 2)	yes	yes
(K 1 M 1 2)	no	yes
(K 1 M 4 3)	no	no
(K 1 M 1 1)	no	yes
(K 1 M 2 4 2)	no	no

(=UNO= (P1 P2 ... Pn)) The P_i are (expression) patterns.

This pattern will match a list E having n elements if there exists a permutation of (P1 P2 ... Pn) which matches E.

(=UNO= (P1 P2 ... Pn ===)) The P_i (i = 1,2,...,n) are (expression) patterns. The last P_{n+1} is the symbol ===

This pattern will match a list E having m n elements, if E contains n elements e_1, e_2, ... e_n such that a permutation of (P1 P2 ... Pn) matches (e_1 e_2 ...e_n).

Example: (=UNO= (1 4 5)) will match (1 4 5), (5 4 1), but not (1 4 4), (5 1 1), (1 4 6), (1 4 5 5).

expression E ↓	pattern X	
	(=UNO= (1 4 5))	(=UNO= (1 4 5 ===))
	is a match obtained?	is a match obtained?
(1 4 5)	yes	yes
(5 4 1)	yes	yes
(1 4 4)	no	no
(5 1 1)	no	no
(1 4 6)	no	no
(1 4 5 5)	no	yes
(4 1 8 5)	no	yes
(1 2 3 4)	no	no
(1 2 3 4 5)	no	yes
(A B C)	no	no

Note that in (=UNO= L), L is a list whose elements are considered to be patterns (expressions) to be matched against the elements of E, which is assumed to be a list. Therefore, a pattern like (=UNO= (=AND= A B C)) will match with a list having the 4 atoms =AND=, A, B and C in any order (we are assuming an empty dictionary); in other words, (=AND= A B C), and for this case any other similar list, is not interpreted as a pattern by =UNO=.

(=UNO= L) interprets its argument L as a list of patters (without order). The only exception is the following:

(=UNO= (=GEN= S)) S is a skeleton.
(=GEN= S) produces a replacement on it, and the result, assumed to be a list, is treated as an unordered pattern, which may or may not have the === at the end.

EXAMPLES OF UNORDERED PATTERNS.

```
NIL means failure  (no match);
Something else (the dictionary) means a match did occur.

start
W 2130.0

resemble(  ( =uno= (a b c)) (* * *) (a b c))
 (* * *)

resemble( (=uno= (a b c (=uno= (1 2 3)) d e)) (* * *) (d e (1 3 2) b a c))
 (* * *)

resemble( (=uno= (a b c)) (* * *) (a b c d))
 NIL

resemble( (=uno= (=ato= =num= =num=)) (* * *) (1 4 a))
 (* * *)

resemble( (=uno= (x y z)) (x pav =num= y pav =ato= z uar ())  ( (a) 3 aa))
 (Z VAR (A) Y VAR AA Y PAT =ATO= VAR AA X VAR 3 X PAT =NUM=
 X VAR 3 X PAV =NUM= Y PAV =ATO= Z UAR NIL)

resemble( (=uno= (a b c ===)) (* * *) (1 2 3 b 4 a 5 6 c 7))
 (* * *)

resemble( (=uno= (a b c ===)) (* * *) (1 2 a 3 b 4 a 5 5 b 6 a b b a))
 NIL

resemble( (a a a) (a uno (1 2 3))  (1 2 3))
 (A UNO NIL A UNO (3) A UNO (2 3) A UNO (1 2 3))

resemble( (a a b a) (a uno (1 2 3)) (2 3 b 1))
 (A UNO NIL A UNO (1) A UNO (1 3) A UNO (1 2 3))

resemble( (a a a a) (a uno (1 2 3))  (1 3 2 1))
 NIL

resemble( (a a a a) (a uno (1 2 3)) (1 2 3))
 NIL

resemble( (b a a a) (a uno (1 2 =ato=)) (b 2 c 1))
 (A UNO NIL A UNO (1) A UNO (1 =ATO=) A UNO (1 2 =ATO=))
```

```
resemble( (a a a a) (a uno (1 2 3)) (1 2 3 1))
 NIL

resemble( (a a a a) (a pat (=or= 1 2 3)) (1 2 3 1))
 (A PAT (=OR= 1 2 3))

resemble ( (a b a) (a uno (1 2)) (2 b 1))
 (A UNO NIL A UNO (1) A UNO (1 2))

resemble( (a b a) (a uno  (1 2)) (b 2 a))
 NIL

resemble ( (=uno= (a b a)) (a pat (=or= 1 2)) (2 b 1))
 (A PAT (=OR= 1 2))

resemble( (=uno= (a b a))  (a pat  (=or= 1 2)) (b 2 a))
 NIL

resemble ( (=uno= (a b a))  (a pat (=or= 1 2))  (b 2 1))
 (A PAT (=OR= 1 2))

resemble ( (a b a)  (a uno  (1 2))  (b 2 1))
 NIL
```