Artificial Intelligence Project--RLE and MIT Computation Center
Memo 46--

A heuristic program to solve Geometric Analogy Problems

by T. G. Evans

October 24, 1962

A program to solve a wide class of intelligence-test problems of the "geometric-analogy" type ("figure A is to figure B as figure C is to which of the following figures?") is being constructed. The program, which is written in LISP, uses heuristic methods to (a) calculate, from relatively primitive input descriptions, "articular" (cf. Minsky, Steps Toward Artificial Intelligence) descriptions of the figures, then (b) utilize these descriptions in finding an appropriate transformation rule and applying it, modifying it as necessary, to arrive at an answer. The current version has solved a number of geometric-analogy problems and is now being modified in several ways and run on further test cases.

The following, intended as a progress report, consists of three relatively short sections, containing (1) a description of the problem type and, quite sketchily, the solution process, (2) some remarks on the choice of problem and approach, and (3) a summary of the steps in the solution of a specific sample problem.

I should like to acknowledge the assistance of the Cooperative Test Division of ETS and, in particular, Mr. John J. Howell, for very helpfully supplying an extensive set of geometric-analogy problems from their files.

Part I

The problems to be considered can be described as follows; one is given a set of eight separate line drawings (each not, in general, connected, and possibly containing dots as well as lines). For later reference, we'll say these figures are labeled A, B, C, 1, 2, 3, 4, and 5, as in the accompanying example. The problem is presented to the subject as follows (quoted from the tests of the American Council on Education): "find the rule by which figure A has been changed to make figure B. Apply the rule to figure C. Select the resulting figure from figures 1-5." These innocent-sounding instructions, which people find rather easy to follow (though many of the very problems on which the program is being tested were thought hard enough to be usefully included on college entrance examinations) lead to a number of difficulties when one tries to mechanize their execution. For example, two of these difficulties, together with an indication of the attitudes taken toward them in the design of the program, are:

(a) First, one must develop a suitable way of presenting the figures to the machine. Considerable work has been done in various places on methods (using such devices as photocell matrices, light-pens, and flying-spot scanners) of going from a picture to a representation of it in a form that can be handled by computer. Since our basic concern is with the later processing, such problems are bypassed and we start off with an internal representation of the figures in list-structure form, read in from punched cards. This representation is a quite primitive form of description, however; it could be obtained <u>via</u> any of the input hardware mentioned by means no more elaborate than line tracing techniques already described in the literature. Roughly, the representation permits the description of an arbitrary line drawing to any degree of accuracy (but only one "shading" of line is permitted): a line is represented by a sequence of as many straight-line segments and arcs of circles as desired. A

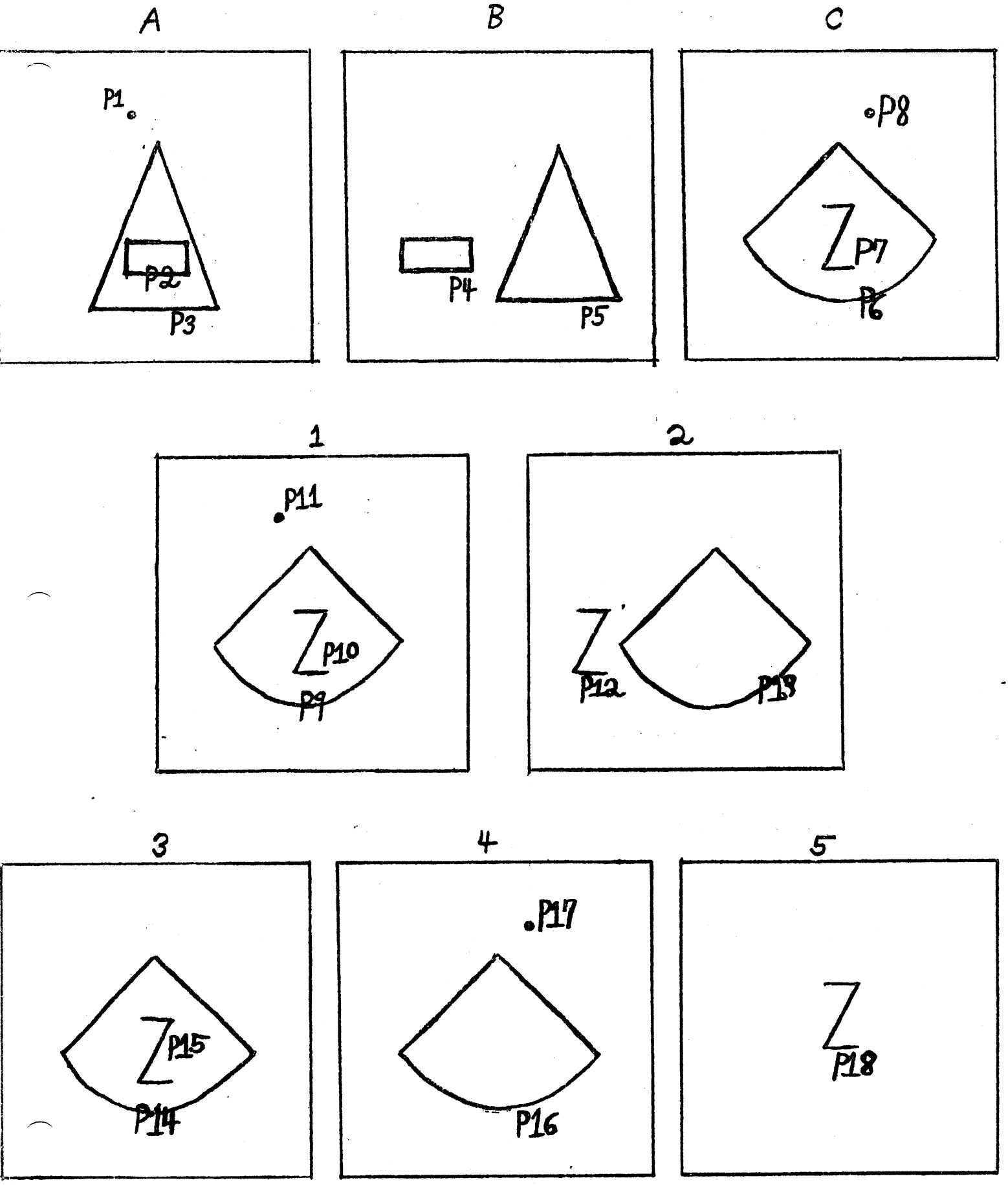

FIG. I

Note: The P1 - P18 in the above are, of course, not

sample of what this list-structure representation looks like is given below, in sec. 3. This representation is convenient for our purposes, since it permits quite economical descriptions of typical problem figures and lends itself well to programmed manipulations. The nature of the required manipulations on the figures is indicated later in this section and in sec. 3.

(b) Next, one must decide what is meant by "rule", i.e., what class of transformations on the figures is to be admitted. Several considerations might govern this choice; (i) one could admit only a formally-defined class of rules R that, together with some formalization of the problem figures, would permit one to obtain "theorems" about the resulting system, e.g., that a given problem has a unique solution within the permitted rule set, i.e., that there is a unique rule T in R such that T(A) = B and T(C) = exactly one of the figures 1-5; or (ii) one could attempt to achieve a very high level of performance by gathering a set of essentially ad hoc tricks, well-adapted for success on a few particular types of problem of interest, and restricting the inputs to these types. However, since our object, described in sec. 2, is not the formal study or maximally efficient solution of geometric-analogy problems per se, but the study of the use of descriptive-language methods in problem-solving, for which these problems seem to provide a fruitful subject matter, our approach is rather different, as will be seen from the description of the process below. (Though rejecting the first alternative, we do have formally-defined rules that can be manipulated formally to obtain new ones; though rejecting the second, we do have formally-defined rules that can be manipulated formally to obtain new ones; though rejecting the second, we do employ techniques special to geometric problems and, in fact, to restricted classes of them--however, considerable pains have been taken to avoid ad hoc solutions which do not contribute to the main goals of study).

What follows is a brief summary of the entire solution

process. A problem of the type described is chosen. The corresponding primitive descriptions are written down for each of the eight figures and punched on cards. These cards are the input to part I of the program (because of storage considerations, the current program is segmented into two large blocks which occupy core at different times -see sec. 4). The first step is to decompose each figure into "objects". The current decomposition program is quite simple-minded; it merely divides a figure into its connected parts, e.g., figure A in the example consists of the three objects labeled P1, P2, and P3. Eventually, one would like to have a more sophisticated decomposition program with, say, the capability of separating overlapped objects on appropriate cues, e.g., suppose fig. A is and fig. B is . The decomposition program should be able to decompose fig. A into the rectangle and triangle on the information that these objects are present in fig. B. Such a more elaborate decomposition program (which has been partly designed) would be of considerable interest, both for itself, as an interesting manipulation on line drawings, and for the extension in problem-solving power it would permit. However, the presence or absence of such a program is immaterial to what follows, since the system has been written so that such a program may be installed simply by removing the current decomposition program and adding the new one, with no other changes any place (except a few in the property and relation subroutines (see below), which are regarded as variable "parameters" of the part I program). The "objects" permitted as output of the decomposition routine need not even be connected.

Next, the objects thus generated are given to a routine which calculates a specified set of properties of these objects and relations between them. As stated above, the program designed so that this set can conveniently be changed. Furthermore, lists of these properties and relations are program parameters, specifying which of the ones currently present in the system are to be calculated on a given run. The current relation-calculation control program permits only two-place

relations, but this could be changed without much trouble; the part II program is written to accept information about arbitrary n-place relations. As a sample of a relation-calculating subroutine, there is one that calculates in figure A of the sample problem, that the object P2 lies inside that labeled P3 and outputs a corresponding expression, (INSIDE P2 P3). The method involves calculating all intersections with P3 of a line segment drawn from a point on P2 to the edge of the field (all figures are considered as drawn on a unit square). In this case P2 lies inside P3 since the number of such intersections is odd, namely 1 (and P3 is known to be a simple closed curve). This calculation indicates the substantial repertoire of "analytic geometry" routines required for part I, to determine, for example, intersections of straight line segments and arcs of circles in all cases and combinations. These routines are also heavily used in the similarity calculations, which, aside from the property and relation calculations, are the principal business of part I. What is calculated, for each appropriate pair of objects is a relation which is a slightly extended form of Euclidean similarity. Namely, the transformation tested for consists of a horizontal, vertical, or null reflection, followed by a Euclidean similarity transformation (uniform scale change and rotation), followed again by a reflection, again chosen (independently of the first) from horizontal, vertical, or null. Part I contains a set of routines that, given two arbitrary line drawings X and Y, will calculate all instances of the above transformation taking X into Y (more precisely, making X congruent to Y up to certain metric tolerances which are parameters in the corresponding programs). This routine is, in effect, a pattern recognition program, with built-in invariance under scale changes, rotations, and certain types of reflections. It consists essentially of a topological matching process, with metric comparisons being made between lines the topological matching selects. Incidentally, it would be easy to suppress the metric parts to obtain a program which

is a completely general topological equivalence test for networks. At any rate, this similarity information is computed for every required pair of objects, both within a figure and between figures, and this information, together with the property and relation information, is punched out on cards in a standard format for input to part II. (For a typical set of figures, the total output of part I, punched at up to 72 col's./card, might come to perhaps 15 to 20 cards).

Part II is given these cards as input. Its final output is either the number of the answer figure or the statement that it failed to find an answer (the selective trace printing facility of LISP provides an easy and flexible means of following the steps in the problem-solving process). The first step generates a rule (or sometimes several alternate rules) which transforms figure A into figure B. Such a rule specifies, as we shall see in a specific case in sec. 3, how the objects of figure A are removed, added to, or altered in their properties and relations to other objects to generate figure B. Once this set of rule possibilities has been generated, all subsequent effort is devoted to trying to "generalize" one of these rules just enough so that the resulting rule, which still takes fig. A into fig. B, now takes fig. C into exactly one of the answer figures. This requires a quite complex mechanism for manipulating and testing the rules and criteria for deciding which of several rule candidates, the results of different initial rules or of different "generalizations", is to be chosen. The principal method embodied in part II at present is able to deal quite generally with problems in which the number of parts added, removed, and matched in taking fig. A into fig. B are the same as the number of parts added, removed, and matched, respectively in taking fig. C into the answer figure. A substantial majority of the ACE test questions are of this type; virtually all would be under a sufficiently "foresighted" decomposition process. This restriction still permits a wide variety of transformation rules. Supplementing this method, which is currently working in full generality, other, rather more specialized, methods

are being developed to make appropriate rule "generalizations" to handle other types of problems from the class characterized above. It should be mentioned that, so far, at least, all the methods of part II have been kept subject-matter-free in the sense that no use is made of any geometric properties of the properties and relations appearing in the input to part II. The more detailed workings of both parts I and II are best explained through an example; sec. 3 is devoted to a typical one.

Part II

The motivations in choosing geometric-analogy problems as the subject matter for a problem-solving program center around the notion of descriptive "languages". One can argue that powerful problem-solving programs must have a good internal "linguistic" representation of the problems they deal with and of the methods they have available for attacking these problems. In particular, this seems a requirement for programs having very much more sophisticated learning capabilities than those of present-generation programs. The current program represents a relatively modest exploration and exploitation of this idea at three levels; (i) first, the inputs to part I constitute a representation of the problem figures in a form convenient for the manipulations of that part; (ii) the outputs from part I again constitute a representation of the figures, in a quite different, more "abstract" form, more convenient for the manipulations of part II; (iii) finally, the rule expressions generated and altered in part II are representations of geometric transformations in a form convenient for the manipulations carried out on them in the course of arriving at a solution.

I hope that the ideas embodied in the figure descriptions and the associated processing techniques mentioned in (i) and (ii) above may contribute rather directly to efforts to construct programs to process line drawings in a variety of applications. Furthermore, the methods associated with the rule descriptions of (iii) may be suggestive, if not directly applicable, in the development of true "theory-forming" problem-solving programs. Our part II may be viewed as a rather modest example of a program which forms a "theory" (the A → B rule) on the basis of some evidence, then "generalizes" this theory, as required, to fit further evidence (fig. C) as well, then makes a prediction from this theory and tests it on the basis of some experimental criterion (here, that the transformation under consideration gives a unique answer figure),

continuing to modify and test its theory until it succeeds or runs out of resources. It seems reasonable to guess that "theory-forming" in a language suitable for describing a given task may be a means of surmounting the limitations of present-day learning programs, and I anticipate that the current program may perhaps contribute to the development of programs with such capabilities.

Part III

To begin the discussion of the sample problem shown in the accompanying figure, I'll give part of the input to part I, namely the input description of fig. A. It looks like:

```
(  (DOT (0.4 . 0.8))
   (SCC ((0.3 . 0.2) 0.0 (0.7 . 0.2) 0.0
         (0.5 . 0.7) 0.0 (0.3 . 0.2)))
   (SCC ((0.4 . 0.3) 0.0 (0.6 . 0.3) 0.0 (0.6 . 0.4)
         0.0 (0.4 . 0.4) 0.0 (0.4 . 0.3)))  )
```

The first line above corresponds to the dot (at coord's. x = 0.4 and y = 0.8 on the unit square). The next two lines correspond to the triangle (SCC stands for simple closed curve: all figures are divided into three classes, dots, simple closed curves, and all the rest in the internal handling of their descriptions for reasons of programming convenience--no other use is made of this three-way classification)--coordinate pairs alternate with the curvatures are zero here since the lines in question are all straight). Similarly, the final two lines correspond to the rectangle; the entire description is a list of the descriptions of these three parts. The format corresponding to non-SCC figures like the **Z** of fig. C is similar though somewhat more complex; the top level describes the connectivity by stating the connections between the vertices--sublists describe the lines joining them.

This input and the corresponding input for the other seven figures is processed and the output from part I is, in its entirety, as follows (in this example, for brevity in writing, since this problem can be handled without them, I have omitted all similarity transformation information which contains non-null reflections). The output consists of ten expressions: (I have replaced the symbols generated internally for the parts found by the decomposition program by the part names (P1, etc.) which appear on the accompanying figure).

(1) ((P1 P2 P3).((INSIDE P2 P3) (ABOVE P1 P3) (ABOVE P1 P2)))

```
(2)  ((P4 P5).((LEFT P4 P5)))
(3)  ((P6 P7 P8).((INSIDE P7 P6) (ABOVE P8 P6) (ABOVE P8 P7)))
(4)  ((P2 P4 (((1.0 . 0.0).(N.N)) ((1.0 .3.14).(N.N))))
      (P3 P5 (((1.0 . 0.0).(N.N)))))
(5)  ((P1 P8 (((1.0 . 0.0).(N.N)))))
(6)  NIL
(7)  ( (P9 P10 P11) (P12 P13) (P14 P15) (P16 P17) (P18)  )
(8)  ( ((INSIDE P10 P11) (ABOVE P11 P9) (ABOVE P11 P10))
       ((LEFT P12 P13)) ((INSIDE P15 P14)) ((ABOVE P17 P16))
       NIL  )
(9)  ( ((P6 P9 (((1.0 . 0.0).(N.N)))) (P7 P10 (((1.0 . 0.0).
        (N.N)) ((1.0 . -3.14).(N.N)))) (P8 P11 (((1.0 . 0.0).
        (N.N)))))
       ((P6 P13 (((1.0 . 0.0).(N.N)))) (P7 P12 (((1.0 . 0.0).
        (N.N)) ((1.0 . -3.14).(N.N)))))
       ((P6 P14 (((1.0 . 0.0).(N.N)))) (P7 P15 (((1.0 . 0.0). (N.N))
         ((1.0 . -3.14).(N.N)))))
       ((P6 P16 (((1.0 . 0.0).(N.N)))) (P8 P17 (((1.0 . 0.0).
        (N.N)))))
       ((P7 P18 (((1.0 . 0.0).(N.N))))  )
(10) ( ( (( P1 P11 (((1.0 . 0.0).(N.N))))) NIL NIL
         ((P1 P17 (((1.0 . 0.0).(N.N))))) NIL  )
     . (NIL NIL NIL NIL NIL) )
```

To explain some of this: the first expression corresponds to fig. A. It says fig. A has been decomposed into three parts, which have been given the names P1, P2, and P3. Then we have a list of properties and relations and similarity infromation internal to fig. A, namely, here, that P2 is inside P3, P1 is above P2, and P1 is above P3. The next two expressions give the corresponding information for figs. B and C. The fourth expression gives information about similarities between fig. A and fig. B. For example, P3 goes into P5 under a "scale factor = 1, rotation angle = 0, and both reflections null" transformation. The next two expressions contain the corresponding information for fig. A to fig. C and from fig. B to fig. C,

respectively. The seventh list is a 5-element list of lists of the parts of the five answer figures; the eighth a 5-element list of lists, one for each answer figure, giving prop., rel., and sim. information. The ninth is again a 5-element list, each a "similarity" list from fig. C to one of the answer figs. The tenth, and last, expression is a dotted pair of expressions, the first again a 5-element list, a "similarity" list from fig. A to each of the answer figures, the second the same from fig. B to each of the answer figures. This brief description leaves a lot of loose ends, but it should indicate what's going on.

Now these ten expressions are given as arguments to the top-level function of part II (optimistically called solve). The sub-method of solve which suffices to do this problem begins by matching the parts of fig. A and those of fig. B in all possible ways compatible with the similarity information. From this process, it concludes, in the case in question, that P2 → P4, P3 → P5, and P1 is removed in going from A to B. (The machinery can also handle far more complicated cases, in which alternate matchings are possible and parts are both added and removed). On the basis of this matching, a statement of a rule taking A into B is generated. It looks like:

```
((REMOVE A1 ((ABOVE A1 A3) (ABOVE A1 A2) (SIM OB3 A1
(((1.0 . 0.0).(N.N))))))(MATCH A2 (((INSIDE A2 A3)
(ABOVE A1 A2) (SIM OB2 A2 (((1.0 . 0.0).(N.N)))))
((LEFT A2 A3) (SIM OB2 A2 (((1.0 . 0.0).(N.N))
((1.0 . 3.14).(N.N)))) (SIMTRAN (((1.0 . 0.0).(N.N))
((1.0 . 3.14).(N.N)))))))(MATCH A3 (((INSIDE A2 A3)
(ABOVE A1 A3) (SIM OB1 A3 (((1.0 . 0.0).(N.N)))))
((LEFT A2 A3) (SIM OB1 A3 (((1.0 . 0.0).(N.N))))
(SIMTRAN (((1.0 . 0.0).(N.N)))))))
```

The A's are used as "variables" representing objects. The format is rather simple. For each object added, removed, or

matched, there is a list of the properties, relations, and similarity information pertaining to it. (In the case of a matched object, there are two such lists, one pertaining to fig. A and the other to fig. B). There are two special devices; the (SIM OB1, ...-form expressions give a means of comparing types of objects between, say, fig. A and fig. C; the other device is the use of the SIMTRAN expressions in the fig. B-list for each matched object. This enables us to handle conveniently some additional situations that I won't attempt to describe here.

This rule contains everything about figs. A and B and their relationship that is used in the rest of the process. (The reader may easily verify that the rule does, in some sense, describe the transformation of fig. A into fig. B in the example).

Now a similarity matching is carried out between C and each of the five answer figures. Matchings which don't correspond to the ones between A and B in number of parts added, removed, and matched are discarded. If all are rejected this method has failed and we go on to try some other method. In our case, figs. 1 and 5 are rejected on this basis. However figs. 2, 3, and 4 pass this test and are examined further, as follows: for a given matching of fig. C to the answer figure in question (and we will go through all possible matchings compatible with similarity) we take each A → B rule and attempt to fit it to the new case, making all matchings of objects between the A's of the rule statement and the objects of C and the answer fig. compatible with preserving add, remove, and match categories, then testing to see which information is preserved, thus getting a new, "reduced" rule which fits both A → B and C → the answer figure in question. In our case, for each of the three possible answer figures we get two reduced rules in this way (since there are two possible pairings between A and C, namely, P1 → P8, P2 → P7, and P3 → P6, or P1 → P8, P2 → P6, and P3 → P7). In some sense, each of these

rules provides an answer. However, we want a "best" rule--if one interprets this to mean the "strongest" rule, i.e., the one that says the most or is the least alteration in the original A → B rule that fits C → some answer figure, then a simple device seems to approximate human opinion on this question rather well; we define a rather simple "strength" function on the rules and sort them by this. If a rule is a clear winner in this test, the corresponding answer figure is chosen; if it results in a tie, the method has failed. In our case when the values for the six rules are computed, the winner is one of the rules corresponding to figure 2, so the program like all humans consulted so far, chooses it as the answer. The rule looks like this;

((REMOVE A1 ((ABOVE A1 A3) (ABOVE A1 A2) (SIM OB3 A1 (((1.0 . 0.0).(N.N)))))) (MATCH A2 (((INSIDE A2 A3) (ABOVE A1 A2)).((LEFT A2 A3) (SIMTRAN (((1.0 . 0.0).(N.N)) ((1.0 . 3.14).(N.N))))))) (MATCH A3 (((INSIDE A2 A3) (ABOVE A1 A3)).((LEFT A2 A3) (SIMTRAN (((1.0 . 0.0). (N.N)))))))

Again, it is easy to check that this rule both takes A into B and C into 2, but not into any of the other answer figures.

Note:

The possibility frequently exists in LISP of writing the same S-expression in alternate forms, by using either dot or list notation, in various combinations. In writing the S-expressions contained in this memo, I have taken the liberty in several places of using this possibility to express them, for greater clarity, in a form different from that produced by the LISP print program. In all these cases, the meaning of the expression in terms of list structure is unchanged by this rewriting.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-46

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 15 (19 - IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☒ Single-sided or
☐ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description : Page Number:
IMAGE MAP: (1-15) TITLE PAGE, 2, ~~WHT~~ FIG, 3-8, 8a-13
(16-19) SCANCONTROL TRGT'S (3)

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 12/11/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 12/11/1995

M.I.T. Libraries
Document Services