Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

Layered "Recognition Cone" Networks that Pre-process, Classify, and Describe

by

Leonard Uhr

Computer Sciences Technical Report #132

December 1971

Revised March, 1972

LAYERED "RECOGNITION CONE" NETWORKS THAT PRE-PROCESS, CLASSIFY, AND DESCRIBE

by

LEONARD UHR*

Abstract--This paper gives a brief overview of six types of pattern recognition programs that 1) pre-process, then characterize, 2) pre-process and characterize together, 3) pre-process and characterize into a "recognition cone", 4) describe as well as name, 5) compose interrelated descriptions, and 6) converse.

A computer program (of types 3 through 6) is presented that transforms and characterizes the input scene through the successive layers of a recognition cone, and then engages in a stylized conversation, to describe the scene.

SUMMARY

This paper examines a sequence of six types of pattern recognition systems. It presents and describes one program to illustrate some of the issues raised and features developed.

The first type (similar to many of today's programs) pre-processes by applying layers of local averaging and differencing transforms to smooth, fill in gaps, and heighten contours, curves, and angles. Then it applies a set of characterizers; each characterizer implies a set of names: and the

---

*Mr. Uhr is Professor of Computer Sciences at the University of Wisconsin, Madison. This work was partially supported by grants NIMH-12266, NSF GJ-583, and NASA NGR-50-002-160. A preliminary version of this paper was presented at the IEEE, UMC, Two Dimensional Digital Signal Processing Conference, Columbia, Missouri, October 6-8, 1971.

program chooses the single most highly implied name.

The second combines the pre-processing transforms and the characterizers into a single general type of operation. Transforms build up new transformed representations of the input, whereas characterizers imply the output name.

The third type erases the distinction between a transform and an implication. Now all outputs are stored in the next transform layer. As the program averages and coalesces information its layers shrink, so that the system builds a cone of layers. When the program reaches the apex - a layer with only one cell that contains all the information - it chooses the single name with which it classifies the input.

The fourth type chooses names when it decides that it should decide among the implications stored in some cell. It thus can choose more than one name, and therefore describe as well as classify the scene.

The fifth examines the interrelations among the set of names chosen, to try to fit the pieces of the description together into a coherent and appropriate whole.

Finally, a sixth step can be taken, to converse about the scene, developing an appropriate description in response to suggestions and queries. This allows the program to do more computations, and to look again, on demand.

INTRODUCTION

Layered nets that perform local transformations have been the basis for a number of pattern recognition programs (see, for example, Rosenblatt,

1962; Rosenfeld, 1970; Levialdi; 1971). But typically these programs first do a sequence of pre-processing operations, such as averaging to smooth and differencing to contrast contours and angles; then, as a completely separate next step, another routine characterizes the output of this serial sequence of parallel transformations (see Uhr, 1966, Lipkin and Rosenfeld, 1970 for examples).

Overview of Present Work

This paper presents and describes a computer program that can intersperse pre-processing transformations and characterizing, and in fact use the same general mechanism to handle both. This immediately opens up attractive, simple and natural, possibilities for describing as well as merely classifying an input.

The generalized transformer-characterizer described in this paper outputs implied names into the next transformation layer along with any transformations that might be specified, but without making any distinctions between implied names and transformations.

Collapsing the matrix from one layer to the next, so that each layer contains fewer cells to the extent that information has been abstracted, allows for simple and natural criteria with which the program can decide when to decide what is implied at a particular region. This leads to the mechanism of a "recognition cone," whose base is the input matrix and whose apex is the single cell that results from successive collapsings from layer to layer. An overall name can be chosen from the apex.

But now each cell in the whole cone itself defines a sub-cone, whose base is the region covered by that cell if and when the program decides to choose among its implied names. The set of chosen names can now form the basis of a description of the scene's objects and their parts. It can also allow for an interacting conversation about the scene.

Background

There are, rather roughly, three major stages to the pattern recognition process:

A. The raw input is "pre-processed," to reduce various kinds of noise and make the pattern more regular and more easily recognizable.

B. The input pattern is "classified," in the sense that it is assigned a name.

C. More than one name is given to an input - either because a scene of several objects must be named or the single object should be "described."

Most research has focussed on classification (Stage B). The typical pattern recognition program applies a set of "characterizers" (for example, line, angle, or area detectors) to examine the input. Each characterizer implies one or more names, with or without weights. The program merges these implications and then chooses the single most highly implied name.

Pre-processing (stage A) means, roughly, "whatever happens before the characterizers are applied." For example, gaps may be filled and contours

enhanced so that edge detectors will work. Often the input patterns are sufficiently similar, and/or a program's characterizers work well enough so that pre-processing is not needed. Sometimes a characterizer does its own pre-processing: for example, a line detector can search for the line's continuation by groping and jumping over gaps, rather than rely upon a prior gap-filling operation (e.g. Grimsdale et al., 1959).

Very little work has been done on C, description. It is not even clear what one might mean by the term description, which lumps together several problems. A description sometimes lists the parts, possibly with their interrelations; sometimes gives general characteristics; sometimes points out likenesses.

PATTERN CLASSIFIERS

We will begin by examining two types of systems for classifying inputs into single pattern names.

1. Different kinds of operations to pre-process, then classify: A surprising amount of preprocessing can be done by simple local averaging and differencing operations on each cell of the matrix and its near neighbors. For example, let's consider a 3 by 3 matrix of a middle cell and its 4 square and 4 diagonal neighbors. To average, we might sum the values stored in each of these cells. We might also weight them, e.g.: multiply the middle cell by 4, the square neighbors by 2 and the diagonal neighbors by 1. To

To difference, we might subtract the value in each cell from the value in the middle cell, and sum these differences.

To do this, a program must: a) scan each of the cells of the matrix, b) extract the sub-matrix that surrounds that cell, c) compute the average or difference function, and d) store the results in the cell corresponding to the middle cell in a new output matrix that is being built. (See Uhr, in press, for detailed discussions and programs.)

One widely used and rather powerful type of characterizer, one that is especially compatible with these local transforms, can quite conveniently be handled by nets of threshold elements, and turns out to give good results in practical running pattern recognizers (e.g., Andrews, Atrubin and Hu, 1968), is what I will call an "n-tuple." An n-tuple specifies a) a set of pieces and the relative positions at which to look for each, b) a threshold or other criterion for success, c) one or more implied names, each with some weight (see Bledsoe and Browning, 1958, Uhr and Vossler, 1961, Uhr and Jordan, 1969). For example, an n-tuple might look for one horizontal and two vertical edges, succeed if any two are found, and therefore imply H, A, and B, with descending weights.

It is convenient to specify information about an n-tuple in a table, and have a subroutine interpret, to: a) get each piece and its expected contents, b) apply it correctly positioned to the input, c) test the threshold criterion to see if the n-tuple has succeeded and, if it has, d) merge the weight of

each implied name into a list of names found. Finally, after all n-tuples have been applied, the program e) chooses the single most highly implied name.

The preceding has really been a fairly detailed and complete description of a program that first pre-processes, and then characterizes and decides upon a single name. (See Uhr, in press, for programs.) Typically, it might average, difference, average, difference, and then apply a set of 5 to several 100 n-tuples. (These n-tuples are usually applied to the last transform layer. But it is very easy to specify Layer as well as Row and Column of a piece, and apply them to any mixtures of layers.)

2. Operations that Transform or Characterize: We can use the table that stores information about an n-tuple characterizer to store information about a local averaging or differencing transform. For an n-tuple can specify any set of pieces - including the tightly packed neighborhood of pieces the transform works with. All we need do is specify the cells involved (for example, the middle and its 8 neighbors), and where to look for this n-tuple (everywhere).

Now a transform stores the combined weights of the pieces in the next layer, whereas a characterizer implies output names which are merged into the list of names found. There thus remain slight differences between transforms and characterizers. But they can be handled so similarly that the program

needed is virtually half the length of the preceding one, since it (almost) combines the two halves, 1) pre-processing and 2) characterizing, into a single process. Such a program differs from the Appendix program chiefly in that it MERGEs IMPLIEDs into a FOUND list rather than into the next layer (statement 23).

AN ILLUSTRATIVE PROGRAM

There is space for only one complete, albeit simple,bare-bones program. The Appendix presents a type 3) recognition cone program that also 4) develops a simple description as well as choosing a single most highly implied name, and has the beginnings of abilities to 5) describe wholes and 6) converse. I will refer to that program when pertinent, and try to describe modifications that would change its type.

Memory Tables and User Commands

The program assumes that it has been given a number of Memory tables, showing it what LAYERS of transforms and CHARacterizerS it should apply. The user must follow the conventions: Input a scene ROW by ROW, preceded by a title card that starts 'SCENE ' with each line started by 'S ', the scene followed by a card that starts 'TRANS'. When the user wants to converse he must start the line with 'AND ' to get another object, or 'PARTS ' to get the parts of the object just described.

The following gives an overview of the program.

Program Overview

1. Memory is initialized (including the characterizers, which are not shown).
2. The scene (or a command) is input and stored as a matrix.
3. The transform layers are computed, as memory directs, by applying each operator (which is either a characterizer or a transform) in the sub-matrix specified, looking at each piece, testing whether the threshold has been achieved, and merging the implieds into the next layer.
4. The layer is erased after it has been completely transformed, and examined for peaks, which imply that a choice should be made among implied names.
5. The program iterates to the next layer, thus adding more layers, until the cone has completely collapsed into a layer with a single cell.
6. The program outputs the single most highly implied object name, and starts a conversation with the user, who can ask for additional names, and for a description of any object.
7. When the program merges a transform's implieds it checks whether it should choose a name to output. It thus builds up a whole description list of object names.

Program Description

The following gives a more detailed description of the program. (Numbers to the left refer to sections in the Overview; numbers to the right refer to statements in the program in the Appendix.)

Statement No.

1. GO Put the lists of LAYERS, CHARacterizerS, and each characterizer into memory. M1-MN

DEFINE the MERGE and CHOOSE functions, and SIZE of each spot. 1-3

2. INPUT and store the pattern to be transformed. 4-11

Get each SPOT and store it under its Layer (equals 0), Row and Column. Intensity (specified by '$') is the SPOT's attribute.

3. TRANSform. Get each Layer to NOWDO and its STEP-shrink size from the list of LAYERS and CHARacterizerS. 12-13

Get each n-tuple to DO and the dimensions of its sub-matrix from NOWDO. 14-15

Get the n-tuple's TyPe, THRESHold, DESCRiption, and IMPLIEDS. 16-18

TyPe A n-tuples average <u>all</u> attributes in all cells specified in the DESCRiption (which contains relative Row and Column locations and weights of each piece). 19

TyPe B looks in each location specified for the VALue of the ATRibute specified and, if this VALue exceeds the MINimum specified, adds 1 to GOT. If GOT exceeds the THRESHold, the IMPLIEDS are MERGED into the next layer.

Iterate through the sub-matrix. 24-25

4. ERASE (when the whole layer is done). 26-33

When a name is of lower weight than at the previous layer, add 1 to 'PEAK' (which will trigger a characterizer that implies '*CHOOSE', which will choose among names in that cell). 31-33

Statement No.

5. ITERate to TRANSform the next layer, 34-36
   Unless the cone has collapsed, 37
   In which case CHOOSE the most highly implied name TO OUTput. 38-39
6. AND OUTPUT and erase the single most highly implied name on TOOUT. Go to IN for a conversational command. (the command 'AND' will return the program to this statement, where another name will be output.) 40-41

   PARTS (input as a command) will have the program OUTPUT all the PARTs of the just-named object that are actually in the scene. 42-45

   $ allows the programmer to put INFOrmation on a new list, named LINE, to memory. Thus LAYERS and characterizers can be changed. 46
7. MERGE combines one list into another, summing weights, 47-52
   using the GOT sum of weights of the pieces found when 48
   '$' indicates, calling the CHOOSE function when the implied name is '*CHOOSE'. 49,52

   CHOOSE gets the name with the HIghest sum of WeighTs (HIWT). 53-56

Illustrations of Transforms and Characterizers

The following gives some simplified examples of the information that must be tabled in this program's memory.

Layers

A first layer with a STEP-shrink of 2 and made up of an averaging transform (T+) that looked everywhere in the matrix, followed by a

characterizer (C1) that looked only at the submatrix Row 3 to 6, Column 4 to 7, is written:

LAYERS = '2 1 1 R C T+ 3 4 6 7 C1 /' (other layers follow)

Transforms

Then the averaging transform, T+, is written:

T+ = 'A 0 0 0 4 /-1 -1 1 /-1 0 2 /' (plus 6 more cells) 1='

Thus a TyPe A is a Transform, with a THRESHold of 0 and no IMPLIEDS. Statement 17 treats the ATRibute as a weight, and MERGEs into the next layer. (A transform can also be written using a TyPe B, since a zero THRESHold will allow it to succeed in all cases. This allows a transform to imply names also, if desired. But it forces the programmer to write a transform for each ATRibute name, whereas the TyPe A transform merges all attributes (statement 19).)

Characterizers

The following are two over-simplified characterizers:

a) C11 = 'B 2 0 0 $ 4 1 0 $ 4 =I 5 E 3 '

This says: 'it's a TyPe B with a THRESHold of 2. The center must have an intensity ($) of at least 4 and the cell directly below it must have an intensity of at least 4. Only then will GOT achieve the THRESHold, and then the IMPLIEDS (I with a weight of 5, E with a weight of 3) will be merged into the next layer.' (One more line of code would handle a loop through intervals, rather than a threshold. This would allow for tests on the absence of an attribute.) Such a characterizer should come later than the first layer, and would usually have more than just two pieces in its DESCRiption.

b) C101 = 'B 2 0 0 MAN 1 0 0 WOMAN 1 1 0 BOY 1 ...=FAMILY 12 '

This would imply a FAMILY if more than 3 pieces of the DESCRiption were found. (The reader should note that a more sophisticated search is needed, as done by Uhr and Jordan, 1969, over some wobble of a piece, rather than into a precisely positioned cell, so that unreasonably long descriptions would not have to be specified.)

PATTERN AND SCENE DESCRIBERS

The program in the Appendix does the following things (numbers in the text refer to its statement numbers):

Transform Cones to Classify and Describe

Up to now we have been talking about transforms that output into next-layer cells corresponding to the middle cells of the layer being transformed, so that the size of the matrices remain the same, as though the layers form a sheaf of paper sheets. But each layering averages, sifts, coalesces, abstracts, or in some other way refines and reduces the information. It would be natural and plausible to think of a cone of sheets, moving from the raw input base to the decision-making apex. (This is much like living sensory systems, whose neural paths converge from eyes and skin to the cortex of the brain.)

To do this we need merely collapse or shrink the matrix. For example, when a 3 by 3 sub-matrix of cells is averaged, the program might do this averaging with every other cell as the center (done by STEP in 14, 19, 23, 31, 33, 35, 36).

Now let the program put implied names into the next layer, along with the transforms (19,23). It must now name the transform (with '$', in 11) and merge transforms and implied names into the next layer just as a type 2 system merges implied names into the special list of found names. Averaging now serves to merge implied names as well as the intensity. When a judiciously chosen set of layers has averaged, differenced, characterized, averaged, characterized, and so on, at the same time collapsing to a single apex cell, the program can treat that apex cell as though it were a single found list, choosing and outputting the most highly weighted name that it contains (37-40).

(The appendix program does more, since it puts a whole description on TOOUT. A program that was just type 3 would eliminate 49, 53, 40-46, and OUTPUT CHOOSE in statement 38.)

Cones within Cones to Describe

We are now in a position to have such a system describe, in a very natural and powerful way. For rather than wait until the entire transform cone has collapsed into its apex, and then make a single decision, the program can decide (49) that information has crystallized locally, that a regional apex has been reached, and therefore CHOOSE an output name (52) to associate with that region.

There are a number of ways to approach this decision. I will suggest two specific sources of information.

Information about a region collapses into a local apex, where it is surrounded by information about mere background, or about some other region. So we can use differencing operations about implied names to trigger the decider. For this the program must be given characterizers that imply '*CHOOSE' (49) when a high difference for one or more names is found (20-22). Alternately the program can note that the strength with which one or more names is implied locally starts to go down from layer to layer (30-33). Thus both differencing over space and peaking (a type of differencing) over time can be used to trigger a choice of a single name (53-56) from those stored in the triggered cell. The name is added to a list of names TO OUTput (52).

Note that this intermediate chosen name is in the apex of a sub-cone, and in some sense covers that cone's base, a sub-matrix in the raw input pattern. But the program can still classify subsequent apexes that cover this apex, up to the final apex of the grand cone. Thus sub-cone can overlap, and cones can contain cones.

Interrelated Descriptions

I will merely suggest some of the ways descriptions can be made more sophisticated.

Deciding to decide is itself a strong push toward appropriate descriptions. Think of the cumbersome description, full of trivial detail (but also useful information) we could get by having the program choose the single

most highly implied name in each cell, and thus output a cone of choices. Deciding to decide selects key cells of that cone.

But there are a number of ways in which names chosen for these key cells can be built into a more meaningful description.

Let's consider four types of description: 1) a list of the parts, sometimes with their interrelations; 2) likenesses; 3) general characteristics, 4) interacting compounds. Some examples follow:

"It's an "A", made up of three straight "LINE"s."

"It's an "A", made up of a right-angled straight line forming an apex with a left-angled straight line, both connected by a horizontal line."

"It's the word "ART", made up of an "A", followed by an "R" and a "T"."

"It's a "FAMILY", made up of a "MAN", "WOMAN", "GIRL", "GIRL", "BOY". We can thus describe an object like "A" by parts like "LINE" or a scene like "FAMILY" by parts like "BOY", or a scene like "ART" by parts like "A". We can further specify interrelations, e.g."forming an apex with", or "followed by" And we can re-name the whole: certain sets of people are named "FAMILY".

A program of type 4 would decide to choose and output a number of names, each name describing the neighborhood covered by the sub-cone from whose apex it was chosen. Let's call this a "raw description," and look at some ways to refine it.

1) The names could be output in some order. For example the name chosen from the grand apex could be output first, on the assumption that it is the highest-level and most general statement about the scene. Then names could be output from apex back, so that more and more detailed statements are given. (The Appendix program would do this if it simply OUTPUT the TOOUT list, rather than discussing it.)

Names could be output in the opposite order, from detailed to general.

Starting with the grand apex, the names of sub-cones of that cone could be output; then of sub-cones of each sub-cone; and so on until all names have been given. This would organize by wholes, parts, and sub-parts, rather than by level of detail.

The weights with which names were implied can be used to order the output (as done in the Appendix program, 40). E.g. the most highly weighted implied name chosen might be output first, as the most salient. It might be followed by its sub-cone part names and/or its super-cone names (since now we start anywhere within the cone), or by the next most highly weighted name.

Positions could be assigned to each name - either the location of the apex cell from which it was chosen, or the boundaries of the base region of that cell. Names might also then be ordered by position, e.g. from top-left to bottom-right.

2) Other names might be output, to indicate possible confusions, similarities, and contrasts. Names almost chosen would indicate that the pattern looked like, and might indeed be, those other patterns. Names not implied, but often confused with this pattern in the past, would also be of interest.

3) The characterizers that implied the output name might be output, to describe general characteristics. (A characterizer is used because it conveys useful information to the program, but it may be gibberish to a human being. So for this type of description we must use characterizers that convey meaning to humans as well as their programs; for example we can designate and use a sub-set, of "public" characterizers.)

Several interesting sub-sets of characteristizers can be used, to highlight various aspects of the pattern. The program might output a) the set that implied the chosen name; the sub-sets that implied b) the chosen name but not its competitors; c) the competitors but not the chosen name; d) the chosen name and the competitors; and e) the sub-set that would have implied the chosen name, but did not characterize this pattern.

4) Finally, several names might interact directly, and compound into a new name. Thus the strokes of the letter interact and combine into the word; the individual people combine into a family. The Appendix program will handle this when given n-tuple characterizers that look for pieces that are names. Thus "FAMILY" might be implied with a high weight by

an n-tuple that looked for the pieces "MAN", "WOMAN", "BOY" and "GIRL", with a threshold set so that any three pieces would suffice.

The description can reflect this interaction, e.g. by outputting " "FAMILY" made up of "MAN", "WOMAN", and "GIRL"." This is done by memory lists that store the PARTs as a Description of the whole (used by 42-45). These pieces might be names, chosen or unchosen, or characterizers or qualities.

Interactive Conversational Descriptions

The discussion of descriptions, even though incomplete, should make clear how many different kinds we might make. There is no single proper description. A "complete" description would inundate us, as would a description that tried to anticipate everything that might be of interest. Most men's noise can always turn out to be some man's crucial piece of information. Different descriptions serve different needs and different purposes; their justness depends upon their audiences, and a describer cannot describe well unless it knows what its audience wants.

A dynamic conversational description seems the way to handle this problem. The program can first output what it decides is important (e.g. the overall choice, or (40) the most highly implied name), and then respond to queries and suggestions from its human users, outputting successively more descriptive information on demand.

Thus some sub-set of the many possible descriptions discussed in the previous section would be output, piece by piece as appropriate to answer the human user's successive queries. The Appendix program outputs the next most highly implied name each time it is asked 'AND ' (40-41). 'PARTS ' will get a list of that object's parts that were found (42-45).

A program must now be able to input user queries on-line, and decide what is the appropriate response from among all the data about the input that it has gathered in its recognition cone. It might further use these queries to trigger new transformations and computations (e.g. in answer to questions like "how big is the "A"?" or "how many people are there?").

To do this, we ultimately need a full-blown semantic understanding program to interpret the query, and a problem-solver to figure out how to get, and then actually get, the answer. But it is quite easy to build in the ability to respond to some simple queries, such as "What are the PARTS?", "How does it DIFFER from others?", "What ELSE implied it?", "Give MORE detail."

SUMMARY DISCUSSION

This paper gives a brief overview of six types of pattern recognition programs that 1) pre-process, then characterize, 2) pre-process and characterize together, 3) pre-process and characterize into a "recognition cone", 4) describe as well as name, 5) compose interrelated descriptions, and 6) converse.

A computer program (of types 3 through 6) is presented that transforms and characterizes the input scene through the successive layers of a recognition cone. It can choose and output names of parts of the scene, and thus describe. It uses its n-tuple characterizers to combine pieces of a description into interrelated wholes. It enters into a conversation (albeit very simple and stylized) about what it has seen.

Only the beginnings of the descriptive and conversational capabilities of this type of program have been presented here. But the technique of recognition cones that combine pre-processing transformations with characterizing appears to be a promising, simple and natural way to move pattern recognizers from their present task of classifying an input, by assigning a single name, to describing, discussing and conversing about a complex scene.

REFERENCES

[1] D. R. Andrews, A. J. Atrubin, and K. Hu The IBM 1975 optical page reader: Part III: Recognition and logic development. IBM J. Research and Development, 12, 1968, 364-372.

[2] W. W. Bledsoe, and I. Browning Pattern recognition and reading by machine. Proc. East. Joint Computer Conf., 16, 1959, 225-232.

[3] R. L. Grimsdale, F. H. Sumner, C. J. Tunis, and T. Kilburn. A system for the automatic recognition of patterns. Proc. IEE, Part B, 106, 1959, 210-221.

[4] R. E. Griswold, J. F. Poage, and I. P. Polonsky, The SNOBOL4 Programming Language. Englewood Cliffs: Prentice-Hall, 1968.

[5] D. J. Farber, R. E. Griwold, and I. P. Polonsky. The SNOBOL3 Programming Language, Bell System Technical Journal, 45, 1966, 895-944.

[6] S. Levialdi, Parallel pattern processing. IEEE Trans. Systems, Man, and Cybernetics, 1, 1971, 292-296.

[7] Bernice Lipkin and A. Rosenfeld, Picture Processing and Psychopictorics. New York: Academic Press, 1970.

[8] F. Rosenblatt, Principles of Neurodynamics. Washington: Spartan, 1962.

[9] A. Rosenfeld, Connectivity in digital patterns, J. Assoc. Comput. Machinery, 17, 1970, 146-160.

[10] L. Uhr, Pattern Recognition. New York: Wiley, 1966.

[11] L. Uhr, EASEy, An Encoder for Algorithmic Syntactic English. Computer Sciences Department Technical Report, University of Wisconsin, 1972.

[12] L. Uhr, Pattern Recognition, Learning, and Thought. Prentice-Hall, in press.

[13] L. Uhr and Sara Jordan. The learning of parameters for generating compound characterizers for pattern recognition. Proc. Joint Conf. on Artificial Intelligence, Washington, 1969, 361-412.

[14] L. Uhr and C. Vossler. A pattern recognition program that generates, evaluates and adjusts its own operators. Proc. Western Joint Computer Conf., 1961, 555-569.

[15] V. H. Yngve. COMIT Programmer's Reference Manual, Cambridge, MIT Press, 1961.

The Recognition Cone Program

```
(RECOGNITION CONE PROGRAM.  TRANSFORMS, NAMES, DESCRIBES,                STATEMENT
(CONVERSES.)                                                             NUMBERS

(FIRST OUTPUTS THE MOST HIGHLY IMPLIED NAME ON TOOUT(NOT
(THE OVERALL NAME))

(THEN DESCRIBES IN RESPONSE TO STYLIZED CONVERSATIONAL QUESTIONS.)

GO      SET LAYERS = (the list of transforms and characterizers is given
          here, by layer)                                                M1

        SET CHARS = (the list of characterizers to be applied after all
          layers)                                                        M2

        SET C1 = (each characterizer must be given, with a list of
            pieces, threshold, implied)                                  M3
                                                                          :
(DEFINES FUNCTIONS AND INITIALIZES MEMORY.)                               :

        DEFINE:  MERGE OF OLD,NEW,WTI.                                    1

        DEFINE:  CHOOSE OF THINGS                                         2

        SET SIZE = 1                                                      3

(INPUTS THE SCENE.  GET AND STORE EACH SPOT UNDER ITS LOCATION.)

IN      INPUT THE TYPE, ROW AND INFO TILL '  '
          [SUCCEEDTO $TYPE, FAILTO END]                                   4

SCENE   OUTPUT ROW ' SCENE IS BEING INPUT AND TRANSFORMED.'               5

        ERASE L AND R [GOTO IN]                                           6

S       SET R = R + 1                                                     7

        ERASE C                                                           8

S1      FROM THE ROW, GET AND CALL THE NEXT SIZE SYMBOLS SPOT.
            ERASE.  [FAILTO IN]                                           9

        SET C = C + 1                                                     10

        SET $(L'.' R '.' C)  = '$ ' SPOT ' ' [GOTO S1]                    11

TRANS   SET TODO = LAYERS CHARS                                           12

(TRANSFORMS INTO THE NEXT LAYER.  'STEP' GIVES THE SHRINK SIZE.)

T2      FROM TODO GET THE STEP AND NOWDO TILL '/'
            ERASE.  [FAILTO TRANS]                                        13
```

```
(GETS THE CHARACTERIZER TO DO, AND THE BOUNDS FOR DOING IT.)
T5      FROM NOWDO, GET RA, CAA, RMAX, CMAX, AND DO.
.           ERASE. [FAILTO ERASE]                                    14
T1      SET CA = CAA                                                 15
(GETS TYPE, DESCRIPTION, THRESHOLD, AND IMPLIEDS.)
T4      FROM $DO, GET THE TYPE, THRESH, DESCR TILL '=' AND
.           IMPLIEDS TILL END                                        16
        ERASE GOT                                                    17
(GETS RELATIVE LOCATION, ATRIBUTE, AND MINIMUM OF NEXT
(PIECE IN DESCRIPTION.)
T3      FROM THE DESCR, GET THE NEXT DR, DC, ATR, AND
.           MIN. ERASE. [SUCCEEDTO $(TYPE 1). FAILTO $(TYPE 2)]      18
(MERGES ALL OBJECTS IN SPECIFIED CELL INTO NEXT LAYER.
(ATR CONTAINS WEIGHT.)
A1      MERGE $(L '.' RA+DR '.' CA+DC) INTO
.           '$(L+1 '.' RA/STEP '.' CA/STEP) ' (WTI = ATR) [GOTO T3]  19
(GETS AND TESTS THE VALUE OF THE ATRIBUTE SPECIFIED.)
B1      FROM $(L''.' RA+DR '.' CA+DC) GET THAT ATR AND
.           ITS VAL [FAILTO T3]                                      20
        IS VAL GREATERTHAN MIN? YES- SET GOT = GOT + 1 [GOTO T3]     21
.
B2      IS GOT LESSTHAN TH? [SUCCEEDTO A2]                           22
        MERGE THE IMPLIEDS INTO '$(L+1 '.' RA/STEP
.           '.' CA/STEP)' (WTI = 1)                                  23
A2      NO- IS CA LESSTHAN $CMAX? YES- SET CA = CA + STEP
.           [SUCCEEDTO T4]                                           24
        IS RA LESSTHAN $RMAX? YES- SET RA = RA + STEP
.           [SUCCEEDTO T1. FAILTO T5]                                25
(ERASES EACH LAYER ONCE IT HAS BEEN TRANSFORMED INTO THE NEXT.)
ERASE   ERASE RA                                                     26
E1      IS RA LESSTHAN R? YES- SET RA = RA + 1 [FAILTO ITER]         27
        ERASE CA                                                     28
```

```
E2      IS CA LESSTHAN C?  YES- SET CA = CA + 1 [FAILTO E1]                29
E3      FROM $(L '.' RA '.' CA) GET THE NEXT NAME AND WTI.  ERASE.
.           [FAILTO E2)                                                    30
        FROM $(L+1 '.' RA/STEP '.' CA/STEP) GET THAT NAME AND WTJ.         31
(DIVIDES WTJ BY 8 FOR A ROUGH NORMALIZATION (SHOULD NORMALIZE
(EACH TRANSFORM.)
        IS WTI GREATERTHAN WTJ / 8?  [FAILTO E3)                           32
        MERGE 'PEAK  1 ' INTO '$(L+1 '.' RA/STEP '.' CA/STEP)'
.           (WTI = 1) [GOTO E3]                                            33
ITER    SET L = L + 1                                                      34
        SET R = R/STEP                                                     35
        SET C = C/STEP                                                     36
(ITERATES UNTIL THE CONE HAS COLLAPSED INTO A ONE-CELL APEX.)
        IS R LESSTHAN 1?  IS C LESSTHAN 1?  [FAILTO T2]                    37
        AT START OF TOOUT LIST CHOOSE OF $(L '.' 0 '.' 0) AND HIWT         38
        ERASE $(L '.' 0 '.' 0)                                             39
('AND' GETS MORE, AND FIRST OUTPUTS THE SINGLE MOST HIGHLY
(IMPLIED NAME.)
AND     OUTPUT THE CHOOSE OF TOOUT 'IS MOST HIGHLY'
.           'IMPLIED.  ASK MORE.'                                          40
        FROM TOOUT, GET THAT CHOOSE AND ITS WT.  ERASE.  [GOTO IN]         41
(DESCRIBES BY OUTPUTTING THE FOUND PARTS, ERASES THEM FROM TOOUT.)
PARTS   FROM $CHOOSE, GET '/D=' AND THE DESCR TILL '/'                     42
P1      FROM THE DESCR, GET A PART AND ITS WT.  ERASE.  [FAILTO IN]        43
        FROM TOOUT, GET THAT PART AND WT ERASE.  [FAILTO P1]               44
        OUTPUT THE PART [GOTO P1]                                          45
(PROGRAMMER CAN USE '$' TO ADD TO AND CHANGE MEMORY.)
$       SET $LINE = THE INFO [GOTO IN]                                     46
(MERGES TWO LISTS.  '*CHOOSE' TRIGGERS A CHOICE.)
MERGE FROM OLD, GET A NAME AND ITS WT.  ERASE.  [FAILTO RETURN]            47
        FROM WT, GET '$' AND REPLACE BY GOT                                48
```

```
          FROM NAME, GET '*CHOOSE' [SUCCEEDTO MCH]                    49

          FROM $NEW, GET THAT NAME AND ITS SUM.  REPLACE BY THE
     .         NAME AND SUM + WT * WTI [SUCCEEDTO MERGE]               50

          ON $NEW, LIST THE NAME AND ITS WT * WTI [GOTO MERGE]        51

(CHOOSES THE THING IN THIS CELL WITH THE HIGHEST COMBINED
(WEIGHT.  ERASES IT.)

MCH       AT START OF TOOUT LIST CHOOSE OF $(L '.' RA '.' CA) AND ITS
     .         HIWT [GOTO MERGE]                                       52

(CHOOSES THE MOST HIGHLY WEIGHTED THING.)

CHOOSE    FROM THINGS, GET CHOOSE AND HIWT.  ERASE.  [FAILTO RETURN]  53

CH2       FROM THE THINGS, GET A NAME AND ITS ORWT.  ERASE.
     .         [FAILTO RETURN]                                         54

          IS ORWT GREATERTHAN HIWT?  YES- SET CHOOSE = THE NAME
     .         [FAILTO CH2]                                            55

          YES-  SET HIWT = ORWT [GOTO  CH2]                           56

END       [GOTO GO]                                                   -
```

APPENDIX: EASEy-1: An English-Like Program Language

Overview of the EASEy-1 Programming Language

The following gives programs in and explanations of an English-like programming language called EASEy-1 (an Encoder for Algorithmic Syntactic English that's easy-Version 1). EASEy is modelled after pattern matching languages like SNOBOL (Farber, Griswold, and Polonsky, 1966; Griswold, Poage, and Polonsky, 1968) and Comit (Yngve, 1961). At present it exists in the form of a SNOBOL4 program that translates an EASEy program into an equivalent SNOBOL4 program that can then be executed by a SNOBOL4 compiler (Uhr, 1972).

EASEy is designed primarily for easy reading, to be understood by someone who knows nothing about programming. EASEy programs are stilted and occasionally awkward. But they should give the reader at least a general idea of what the system is doing, along with the opportunity to study the actual code, when desired, until it is understood. Most of the difficulties in reading will result from the logical structure of the program's processes, rather than the peculiarities of the program's language--that is, from content and not form.

A concise primer for EASEy follows the program. But the reader should first try to read the program without the primer.

Here are the essentials: EASEy allows the user to name lists, and then manipulate them. EASEy defines a list by assigning a string of objects as the contents of a name (e.g.: SET TODO = LAYERS CHARS). Objects are got from lists (e.g.: FROM TODO GET...) and added to lists (e.g.: LIST NAME WEIGHT ON MAYBE).

GOTO a label is indicated at the right of a statement, in brackets. Comment cards start with '(' and continuation cards start with '+'.

Most other conventions are quite natural, except for the very confusing construct that means "the contents of the contents of this name", which can be indicated by $name. E.g.:

Code	Meaning	Result
SET R = R + 1	Add 1 to the contents of R	R contains 1
SET $(L'.'R) = R '*0011'	Assign '1*0011' as the contents of (L'.'R)	L.1 contains 1*0011

List structures and graphs can now be handled by storing a string of names, getting a name, and looking at the string it points to, using the $name construct.

A Primer for EASEy-1, an Encoder for Algorithmic Syntactic English

EASEy-1 is a list processing, pattern-matching language that uses simple English-like formats designed to be easy to read.

An EASEy program is a sequence of statements that construct and rearrange lists of information, find items on these lists, compute transformations on these items, rearrange information within and between lists, and input and output information. Statements are executed from top to bottom except when GOTO's indicate otherwise. A GOTO may be conditional on the success or failure of the statement's search for a pattern, or test for an inequality.

I. A Simple EASEy-1 Program

```
                                                                    A
(PROGRAM A.  AN EXAMPLE PATTERN RECOGNIZER.                         C1*
(POSITIONED N-TUPLES IMPLY WEIGHTED NAMES.                          C2
(INITIALIZES CHARACTERIZERS                                         C3

INIT    SET CHAR1 = '0111 2 1000 9 1111 24 =B 6 F 9 '               M1

        SET CHAR2 = '001111111 3 0000000 18 =5 E 9 '                M2

                  :                                                  :
                  :                                                  :

        SET CHARN = . . .                                           MN

SENSE   SET LOOKFOR = 'CHAR1  CHAR2  . . . CHARN '                  1

        ERASE MAYBE.                                                2

IN      INPUT THE PATTERN TILL '/' [FAILTO END]                     3

(GETS EACH CHARACTERIZER'S DESCRIPTION AND IMPLIEDS                 C4

RESPOND FROM LOOKFOR GET THE NEXT CHAR.  ERASE.  [FAILTO OUT]       4

        FROM $CHAR GET THE DESCR TILL '=' AND THE IMPLIEDS
+           TILL THE END.                                           5

(ALL HUNKS MUST BE FOUND FOR THE CHARACTERIZER TO SUCCEED.)         C5

R1      FROM THE DESCR, GET A HUNK AND ITS LOCATION.  ERASE.
+           [FAILTO IMPLY]                                          6

        AT THE START OF PATTERN, GET AND CALL LOCATION SYMBOLS
+           LEFT, AND GET THAT HUNK.                                7
+           [SUCCEEDTO R1.  FAILTO RESPOND]

(MERGES IMPLIED NAMES ONTO MAYBE.

IMPLY   FROM THE IMPLIEDS, GET THE NEXT NAME AND ITS WT.  ERASE.    C6
+           [FAILTO RESPOND]                                        8

        FROM MAYBE, GET THAT NAME AND ITS SUM.  REPLACE BY NAME
+           AND SUM + WT [SUCCEEDTO TEST]                           9

        ON MAYBE LIST THE NAME AND ITS WT [GOTO IMPLY]              10

(OUTPUTS THE FIRST NAME WHOSE SUM OF WEIGHTS EXCEEDS 30,
(OR THE LAST NAME IMPLIED.)                                         C7
```

*A number in the right margin refers to a statement in Program A that illustrates the construct being discussed. C = Comment, D = Data program inputs, M = Memory initialization.

```
TEST     IS THE SUM + WT GREATERTHAN 30? [FAILTO IMPLY]                    11
OUT      YES- OUTPUT THE PATTERN ' IS A ' NAME [GOTO SENSE]                12
(THE END CARD, AND 3 PATTERNS TO BE READ IN ON DATA CARDS FOLLOW.)         C8
END      [GOTO INIT]                                                       13
0001111111000010101010101011000/     (first two hunks of CHAR1
                                       will succeed, third fails)          D1
0001111111000010101010101011111/     (CHAR1 succeeds)                    D2
0000011111111000010000000000/        (CHAR2 succeeds)                    D3
```

II. EASEy-1 Constructs

A. Basic Statement Types for List Manipulation

1. Lists are initialized and added to:

   a. Names can be assigned to strings of objects:

      E.g.: SET name = objects M1,1*

      ```
      SET C1 = '00111'

      SET LOOKFOR = C1 ' ' C1 ' ' C1 ' '
      ```

      assigns '00111' as the contents of C1, and then

      assigns '00111 00111 00111 ' as the contents of LOOKFOR.

   b. Objects can be added to the end of a named list:

      ON name SET objects

      E.g: ON COUNTRIES SET COUNTRY '=' AREA ','

      adds the contents of COUNTRY followed by '=',

      the contents of AREA, ',' to the end of COUNTRIES.

c. Objects can be added to the start of a named list:

AT START OF name SET objects

E.g.:

AT START OF DESCRIPTORS SET DESCRIPTOR ' ' WT ' '

d. Objects can be listed under a name

ON name LIST objects

E.g.:

ON IMPLIED LIST NAME WT 10

LIST is much like SET, except that it automatically puts a delimiter (one space) after each object listed, except for literal strings (those enclosed in quotes).

e. Objects can be listed at the start of a named list:

AT START OF name LIST objects

E.g.:

AT START OF DESCRIPTORS LIST DESCRIPTOR WT

2. Information is got, erased, and replaced in lists:

a. Objects can be got from a named list:

FROM name GET objects

E.g.:

FROM SENTENCE GET WORD 4,5,6

will assign the name WORD to the first string on SENTENCE, ending with the space delimiter.

Delimiters: An Aside

EASEy uses one space as its internal delimiter. So the user can specify delimiters, as:

E.g.:

SET LOOKFOR = '3 00111 B 3 F 6 /'..

Then, FROM LOOKFOR GET POSITION, DESCRIPTION IMPLIED TILL '/' 5 will assign 3 as the contents of POSITION, 00111 as the contents of DESCRIPTION, and 'B 3 F 6 ' as the contents of IMPLIED. EASEy uses the next internal delimiter (one space) to define a variable name if it is followed by another variable name; otherwise it assumes the next object (or the end of the string) is the delimiter.

The user must take care that spaces used for other purposes are not mistakenly found and used as delimiters.

b. Objects can be got and erased by extending the GET command:

FROM name GET objects ERASE

E.g.:

FROM LOOKFOR GET CHAR ERASE 4,8

c. The objects can be replaced by other objects:

FROM name GET objects REPLACE BY objects

E.g.:

FROM LOOKFOR GET CHAR WT REPLACE BY TRANS 9

An equal sign ('=') can be used instead of 'ERASE' or 'REPLACE BY'.

d. All contents can be erased from named list:

ERASE names

E.g.:

ERASE R C MAYBE 2

3. Information is input and output:

a. One card of data can be input and names assigned to its contents:

INPUT objects

E.g.:

INPUT TYPE TILL '*' LINE TILL ' ' 3

b. Lists can be printed out:

OUTPUT objects

E.g.:

OUTPUT LOOKFOR ' = ' $LOOKFOR 12

B. Types of Objects Used

An object is a string of symbols followed by one or more spaces. Such a string is often a name whose contents are some other string of objects to which it points. Several different kinds of string are used, as follows:

1. Names: A name is an alphanumeric string that points to (names) some contents. 1,4

2. Literals: When a string is in quotes it is a literal ikon that signifies itself.

E.g.: FROM SENTENCE GET ' AND ' 3,5

means that the thing in quotes-- ' AND ' should be found in SENTENCE.

3. Variable names. A string of symbols that comes after GET is treated as a name to be assigned some contents. It will be assigned the string in the named list up to the next space delimiter, unless it is followed by a specified object or a literal object, in which case it is assigned the string up to that object. END (or TILL END) will assign the rest of the list, till its end, to the variable name.

E.g.: FROM SENTENCE GET MODIFIER NOUN TILL ' IS '

+###OBJECT TILL END 3,5,9

4. Specified Objects. THAT string will look for the contents of the string.

E.g.: SET PHRASE = 'THE TABLE ' 9

FROM TEXT GET THAT PHRASE

will see whether 'THE TABLE ' (the contents that has been assigned to PHRASE) is in TEXT, whereas:

FROM TEXT GET WORD

assigns the name WORD to the first string ending with the

space delimiter in TEXT.

5. Indirect and Compound Names. $string will treat the contents named by that string as a name, and look in the string it names. Parentheses can be used to compound together a sequence of several literals and named strings.

E.g.: SET R = 1 5

SET $('ROW.' R) = '1001100'

will set ROW.1. to contain 1001100 (since R contains 1).

6. Matching from the Start of the List. AT START OF insists that the match begin at the very start of the list.

E.g.: AT START OF SENTENCE GET 'THE' 7

looks for 'THE' only at the very start of the SENTENCE.

7. Specifying the Length of a String. CALL length SYMBOLS string will get a string exactly length symbols long, and assign the string following the word SYMBOLS as its name.

E.g.: FROM PATTERN GET AND CALL 6 SYMBOLS PIECE 7

will assign PIECE as the name of the first 6 symbols in PATTERN.

C. Functions.

1. Arithmetic is handled in the conventional way. Parentheses are not needed if ordinary precedence of operators is desired. +=add,

- = subtract, * = multiply, / = divide, ** = exponentiate.

E.g.: SET WEIGHT = WEIGHT + INCREMENT 9,11

2. Tests for inequalities are of the form: IS Object1 TEST Object2? The tests are a) numeric: GREATERTHAN, LESSTHAN, or EQUALTO and b) string-matching: SAMEAS.

E.g.: IS SUM GREATERTHAN THRESHOLD? 11

3. The built-in function SIZE OF object will count the symbols in the object (if it is a literal) or the list named (if the object is a name).

The function RANDOM OF Number will get a random number between 1 and the number specified.

4. The user can define his own functions by saying DEFINE: followed by the function name, OF, and the arguments. When the function name is then used in a statement, the program goes to the statement with that name as its label, executes the function, and exits using RETURN and FAILRETURN in gotos.

D. Flow of Control.

1. A label can be used to name a statement. All labels must start in column 1. No two statements can have the same label. 1,3

2. Statements are tied together by gotos at the right of the statement which name labels at the extreme left of the statement to be gone to. Unconditional gotos are of the form: [GOTO label]. 10⇒8

Gotos conditional on the success or failure of the statement (either a pattern match or a test) are specified by [SUCCEEDTO label] and [FAILTO label]. Alternately, parentheses, and +TO and -TO can be used. 9⇒11 11⇒8

3. A program statement can be continued by starting the next card with '+' or '.' . 7

4. A comment card must start with a left parenthesis ('(') in column 1. C

5. A program must end with a card that has END in its first three columns. 13

6. An EASEy program is a sequence of statements (each card can contain up to 72 columns; the last 8 columns are reserved for identification), b) an END card, and c) cards with data (all 80 columns can be used).

E. Flexible Constructs

1. A number of words and punctuation marks are ignored, so that they can be used as filler by the programmer, to make his statements easier to read. These include the words (when between two spaces) A, AND, INTO, IT, ITS, TILL, NEXT, NO, OF, THE, YES, and the punctuation marks (only when followed by a space) . , : , - and , .

(Note that the colon and period can be used in command words. E.g.: either GET: or GET, and ERASE. or ERASE are acceptable.)

2. Several spacing variants are allowed: a) One or more spaces must bound all names and objects, except b) No spaces are needed around arithmetic operators.

III. Summary of EASEy-1 Constructs

A. Basic Statement Types for Manipulation:

1. Build lists:

a.	Assign:	SET	name	=	objects
b.	Add: (at end)	ON	name	SET	objects
c.	Add: (at start)	AT START OF	name	SET	objects
d.	List: (at end)	ON	name	LIST	objects
e.	List: (at start)	AT START OF	name	LIST	objects

2. Get, erase, replace:

a.	Get:	FROM	name	GET objects
b.	Get and erase:	FROM	name	GET objects ERASE
c.	Get and replace:	FROM	name	GET objects1 REPLACE BY objects2
d.	Erase:	ERASE	names	

3. Input and output:

a.	Input:	INPUT	objects	(inputs one data card)
b.	Output:	OUTPUT	objects	

B. Types of Object:

1. Names: alphanumeric strings
2. Literals: strings surrounded by quotes
3. Variable names: to be assigned contents up to either:
   a. the next delimiter space;
   b. if a literal or specified object follows that object;
   c. if END or TILL END follows, the end of the list.
4. Specified objects: THAT name specifies the contents of the name.
5. Indirect and compound names: $name, $(name literal...)
6. To match from the start: AT START OF name GET objects
7. To specify length: FROM name GET AND CALL length SYMBOLS object

C. Functions:

1. Arithmetic: +,-,*,/,**. E.g.: RESULT = A + B - C * D/E**F
2. Inequalities: IS number1 INEQ number 2?,
   (where INEQ is GREATERTHAN, LESSTHAN, EQUALTO)
   IS object1 SAMEAS object2? (objects must match exactly)
3. Built-in: a) SIZE OF objects (counts symbols): b) RANDOM OF number
4. User defined: DEFINE: name OF arguments

D. Flow of Control:

1. Labels start statements at the left, in column 1.
2. GOTOs at the right in brackets or parentheses name labelled statements to be branched to:
   a. Always: [GOTO label] or [TO label]
   b. On success: [SUCCEEDTO label] or [+TO label]
   c. On failure: [FAILTO label] or [-TO label]
3. Continuation cards start '+' or '.'
4. Comment cards start '('
5. END starts the END card that ends the program.
6. Program structure: a) Program (72 cols); b) END card; c) data (80 cols).

E. Filler words and variants for flexibility:

1. Filler words that are ignored: ' A ', ' AND ', ' INTO ', ' IT ', 'ITS ', ' NEXT ', ' NO ', ' OF ', ' TILL ', ' THE ', 'YES ', ' . ', ' , ', ' : ', '_ '.
2. One or more spaces must bound names and objects, except arithmetic operators. '=' can replace 'ERASE' or 'REPLACE BY'.