WIS-CS-73-176
The University of Wisconsin
Computer Sciences Department
1210 West Dayton Street
Madison, Wisconsin 53706

Received February 1973

DESCRIBING, USING "RECOGNITION CONES"

by

Leonard Uhr

Computer Sciences Technical Report #176

February 1973

This research has been partially supported by grants from the National Institute of Mental Health (MH-12266), the National Science Foundation (GJ-36312), NASA (NGR-50-002-160) and the University of Wisconsin Graduate School.

DESCRIBING, USING "RECOGNITION CONES"

Leonard Uhr

ABSTRACT

Parallel-serial perceptual "recognition cones" (RE-CO-DERS[1]) are being developed to handle scenes of interacting objects, by successively transforming and coalescing information. Recognition cones apply the same mechanisms to pre-process, characterize, find parts of objects, find qualities, name, and describe. They attempt to model living perceptual systems, in their generality of functions and homogeneity of processes, as well as in their overall structure.

A recognition cone program for scenes of interacting objects is presented, described and discussed.

The concept of a "description" is discussed, and a variety of types of descriptive information is surveyed. The recognition cone is then extended to handle many of them. Several further problems in describing are discussed, including the importance of culling information from the overly large - indeed potentially infinite - exhaustive description that is possible, and using an understanding (whether built-in, or through commands, conversation, or learning), of the recipient of the description - the hearer - to help determine relevance.

---

[1] REcognition COne DEscribeRS, where Recognized Environments are Coalesced by the Organism to Describe Experiences Recaptured from the Scene.

DESCRIBING, USING "RECOGNITION CONES"

Leonard Uhr

INTRODUCTION

This paper examines a parallel-serial "recognition cone" model for its ability to describe scenes of objects, and to tailor its descriptions in response to commands input from a user-hearer. An actual program is presented, in an English-like language called EASEy, to allow the reader to see exactly what is happening (if help is needed, see the Appendix, and Uhr, 1973a). The program is kept as simple as possible, but when given the appropriate set of transforming and characterizing operators (just as a parser must be given rewrite rules), it is a relatively powerful recognizer and describer.

This paper is organized into the following sections:

1. Naming inputs vs. describing scenes.
2. "Recognition cones" described, and an example program.
3. "Describing" discussed.
4. Extending recognition cones to describe, as commanded.
5. Discussion of problems and possible improvements.

NAMING INPUTS VS. DESCRIBING SCENES

Pattern Recognizers that Assign a Single Name to the Input

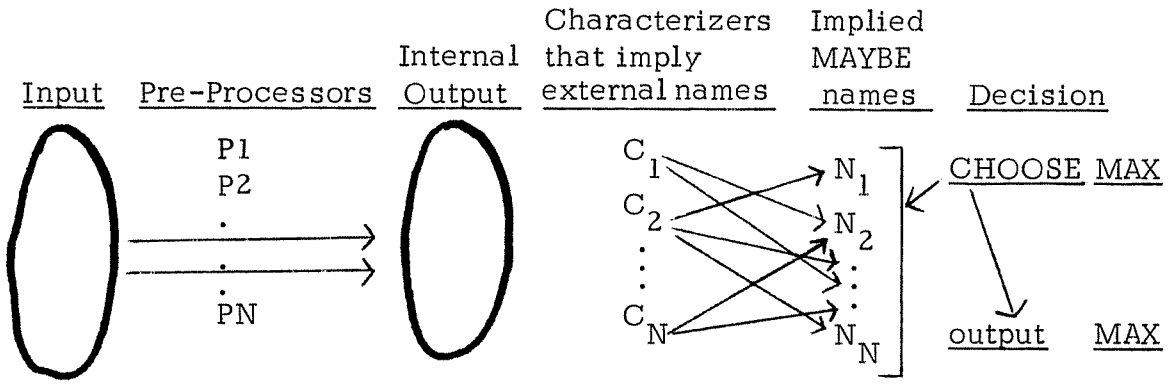
A typical pattern recognition program applies the following steps: 1) Pre-processing, to eliminate noise, smooth, average, sharpen edges, difference, and in other ways regularize. 2) Characterizing, to assess what possible names might be assigned to the input pattern; 3) Deciding, by choosing the single most highly implied name. There have been many variations on this theme, some of which follow:

Often the pre-processing is absent.

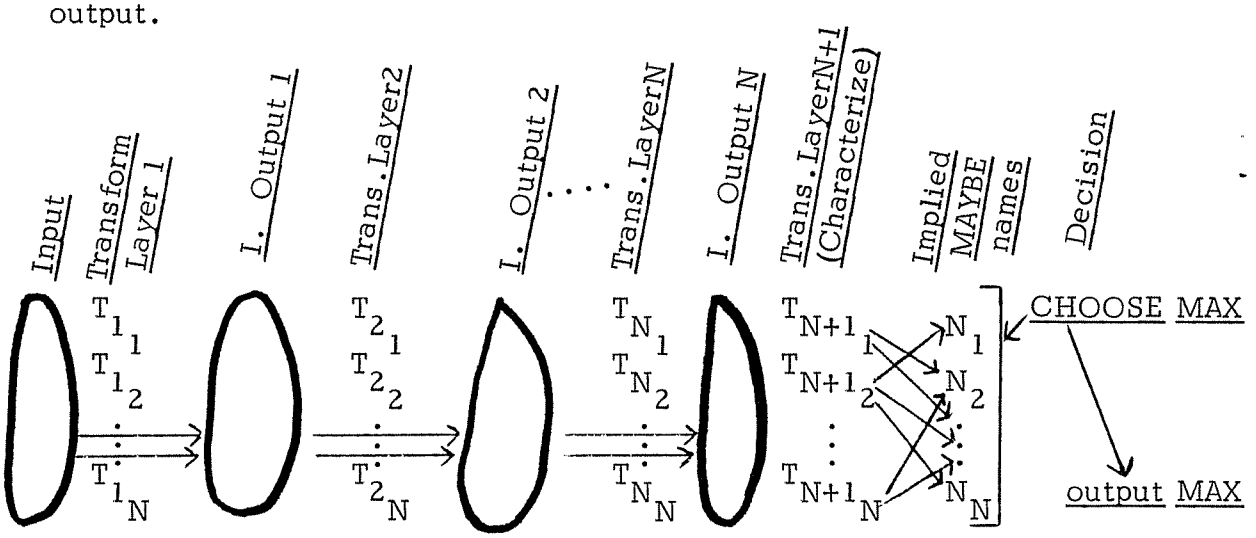
Usually the program applies its entire set of characterizers to the input, combines all the implications of possible names, and then chooses the single most highly implied among them. But sometimes it applies characterizers in several sets (let's call them "layers"), so that the transforms from each layer (these can be preprocessings, or internal names, e.g. of compounds) are input to the next. Usually only the final layer outputs implied names to be chosen among; sometimes all the layers output a mixture, of implied names, and information to be input to the next layer.

Sometimes a decision rule for choosing is periodically invoked, so that the program will choose and output a name before all characterizers have been applied. Sometimes the characterizer implies additional characterizers to apply, so that the particular set of characterizers used will be a function of what information has been gathered so far about this particular input pattern. Figure 1 diagrams several alternate structures.

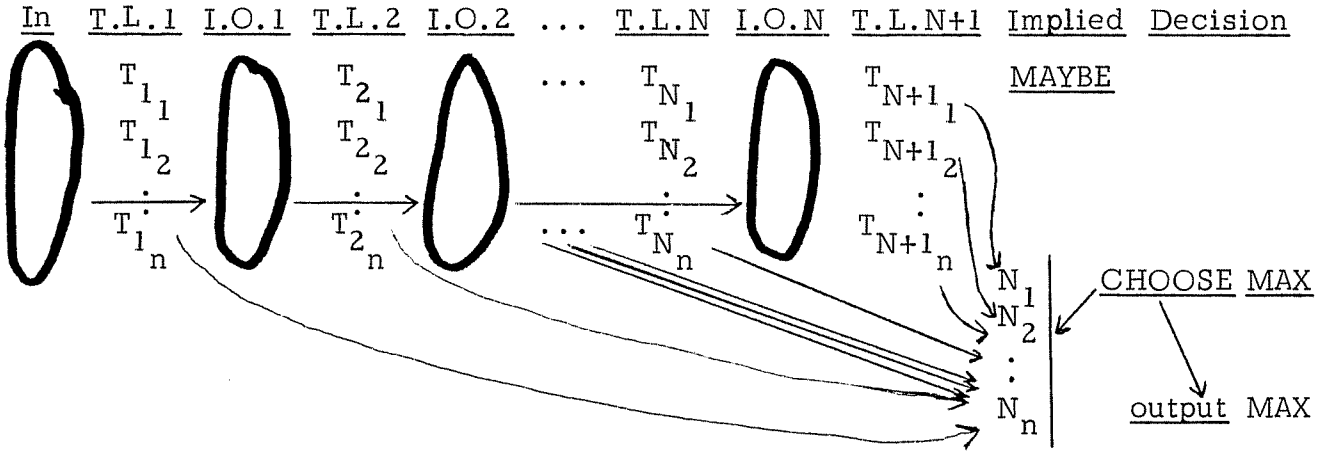
Figure 1. Some Alternate Structures for Pattern Namers



A. Basic structure of a pattern namer. Pre-processors transform the raw input; characterizers of this Internal Output imply external name possibilities, among which the MAXimum implied is chosen for output.

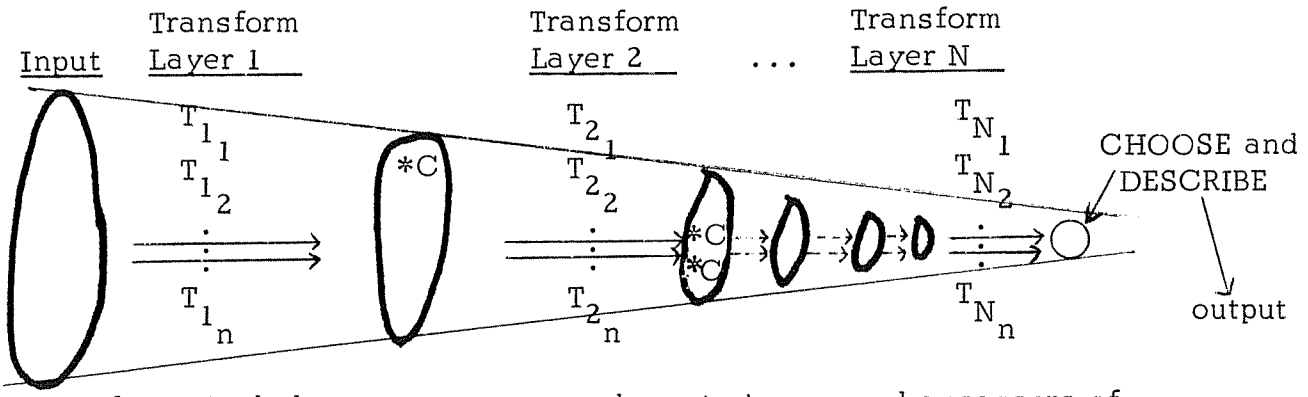


B. Layered pattern namer. Early Transform Layers pre-process; later may characterize, by building up internal structures; layer T_{N+1} characterizers, implying possible external names, from which the MAXimum implied is chosen and output.



C. MAYBE Names can be Implied from any layer.

Figure 2. The Structure of a Recognition Cone to Name or Describe



Transforms include pre-processors, characterizers, and assessors of qualities. The apex cell of the collapsed cone becomes the MAYBE list from which the system can CHOOSE a name or description. (Transforms can also trigger choices in prior layers, as indicated by *C above. These will be carried along, finally arriving at the apex, by transforms that average and sum.)

Fig. 1 about here

But in all cases a grand decision is made, which chooses a single name to output. (See Uhr, 1973b, chapters 2 through 9, for an examination of these and of other alternatives.)

"Recognition Cones" (RE-CO-DERS) to Model Perception

"Recognition cones" are designed to handle 1) pre-processing, 2) layers of characterizing, and 3) deciding, using the same mechanisms, in a simple and homogeneous way (see Uhr, 1972, for details). A recognition cone (see Figure 2) consists of a series of transformation layers, where transformation operations can be either pre-processors (e.g. summing a cell and its neighbors, to "average") or configurational characterizers (e.g. that look for several relatively positioned edges and angles). All transforms output to the next layer of the cone, in contrast to the typical pattern recognizer, where transforms' outputs go either to the next layer or to the list (let's call this the MAYBE list) of implied names among which a choice is to be made.

Fig. 2 about here

Layers grow smaller, collapsing from the raw input scene at the base of the cone to a final layer, with only one cell, at the apex. To merely do classical pattern recognition, which is really pattern naming, this single apex cell can be used for choosing, since it contains all the implications of possible output names, along with the combined weights of each. This is so because these implications have

been merged by the transforms into the next layers' cells, along with all other information.

The collapsing from layer to layer not only serves to merge all information together into the apex, but also appears to be desirable in terms of sound design: the transforms extract, abstract and coalesce information, usually in a many-to-one fashion. Each transform's output is a function of and summarizes many input cells; collapsing and converging acknowledges this funneling of information.

The recognition cone seems attractive for its relative simplicity and homogeneity. But its use leads to two further consequences that are of special value for describing, as opposed to merely naming.

Recognition Cones for Describing

First, one of the most difficult problems for a describing program is to decide where one thing ends and another begins - that is, to decide where to assign names. The classic naming program can ignore this decision completely, for it merely chooses the single most highly implied name. In fact such a program is really naming the entire input matrix within which the pattern lies, and not the pattern itself. But the instant more than one thing might be in the input the program must worry about whether to choose among several alternate possibilities to assign to a single region, or to assign two (or more) names to several different regions.

Deciding How Many Names to Assign, and Where

Consider the relatively simple (but unsolved) problem of hand-writing, where the program must decide among a number of overlapping

combinations, eg "c i i i" or "a u" or "c w" or "a i i" . Such problems have almost always been handled by trying first to segment the input into regions, where each region contains only one object, and then applying classical techniques for characterizing , and choosing a single name (e.g. Frishkopf and Harmon, 1961; Uhr and Vossler; 1963; Mermelstein, 1964; Eden, 1968.)

But that is clearly the wrong thing to do, for it ignores an enormously rich source of pertinent information - the patterns being segregated. Rather than trying to find boundaries between regions before it even begins to look at those regions, a sensible system would characterize the scene in order to find the regions of interest. Boundaries would then grow naturally between such regions. Even better, once something is chosen and recognized as filling a certain region it could in effect act as a boundary that helps to delineate all the regions it touches.

But we have not had systems capable of doing things this way, for putting implied names into a single MAYBE list destroys all positional and regional information. The recognition cone, since it keeps passing names from layer to layer back toward the apex, along a line from where the characterizer was applied on the input, keeps all this information separated into regions, in what seems quite a natural way.

Deciding to Choose Names for Sub-Regions of the Scene

Further, the recognition cone, since its transforms are capable of implying a variety of different things, can imply signals that trigger the decision to choose among names in any particular cell,

not only in the grand apex cell. This means that information about a name that is highly implied, and/or about a region that contains one, or several, strongly implied names, as opposed to its neighborhood which contains very little (that is, looks like background), can trigger the decision to choose.

Whenever such a choice is made, the choosing cell can be treated as an apex of a sub-cone within the grand recognition cone. Now, moving back toward the base, the system can make use of whatever information is stored in that sub-cone in order to develop a description of it. And of course the sub-apex choices serve to describe the grand cone.

Describing Named Objects' Parts, Structure, Qualities, and Other Attributes

The sub-cone stores and organizes this information about the named object chosen from its apex. This includes the parts of the object, and, implicit in where they are stored, their locations, and, therefore, their structure. The characterizing transforms that implied the objects can also be used for descriptive information.

Finally, a variety of different kinds of potentially descriptive information - about qualities like shading and color, or features like edges, curves and angles, or abstractions like feelings or imports (eg "ominous" or "lively" or "beautiful") can be implied by transforms, and stored along with everything else in the cells of the recognition cone. We are now in a position to ask the recognition cone to use this kind of information to enrich its descriptions of the scenes input to it.

Program RE-CO-DERS-1 Described (See Uhr, 1972, for more Information)

RECODERS-1 must be given a set of operators - transforms and characterizers - to apply, and these must be named on the list of *LAYERS (statement M1), at the layer desired, or on CHARacterizerS (M2) which are applied after all LAYERS. The STEP-size for collapsing each layer must also be specified (M1,9), along with SpotSIZE (M(N+1)) and the maximum number of ITEMS in a description (M(N+3)).

Initializing. A new scene is INput, ROW by ROW, and decomposed into SPOTs stored in separate cells (1-6) until a card starting TRANSform indicates that the scene has been completed and RECODERS-1 should start perceiving it. TRANSforming continues layer by layer until the scene has collapsed into a single apex cell (7-29). Each layer has a STEP-size and a set of operators on NOWDO, each with the sub-matrix through which it should be applied (9). First the next layer is Erased (10,25), then the operators are applied (12-24), their outcomes being MERGEd into the corresponding cell (which is a function of STEP-size) in the next layer (17,22).

Building Layers. Operators are of three TYPEs: TYPE A MERGE All things into the next layer (17), and are therefore convenient for averaging and differencing without having to bother specifying each specific thing being merged. (A whole set of TYPE C operators, one for each thing, would, in a very cumbersome way, give the same results.) TYPE C first look for a Configuration DESCRibed in the operator (18-21) and, if enough described THINGs are GOT in their specified relative location to TOTAL above the specified THRESHold, the set of IMPLIEDS (which can include names, qualities, and triggers

*Caps in the text and OVERVIEWs refer to names in the programs; numbers refer to statement numbers (in the right margin).

to *CHOOSE) is MERGEd into the next layer. TYPE E, which are automatically added at the start of each layer (10), Erase the corresponding cell in the next layer (25).

Choosing. RECODERS-1 CHOOSEs a name to put on a TOOUTput list whenever it finds (37) a *CHOOSE that has been implied (several characterizers should assess how well information about a variety of implied things has peaked and coalesced in a certain region, in contrast to little activity in neighboring regions - all implying a PRECHOOSE name which, when it reaches a certain weight, triggers another characterizer to imply *CHOOSE - thus this is handled entirely by operators). A final name is CHOOSEn from the grand apex cell (30). Finally RECODERS-1 CHOOSEs and outputs the most SALIENT names on TOOUTput, quitting when the specified number of ITEMS (= 7) has been given, or TOOUT is empty (31-33).

Describing. RECORDERS-1 collects an enormous amount of descriptive information, in the cells of the recognition cone and in the TOOUT list. This information includes all implied things, whether wholes, sub-wholes, parts, qualities, characteristics, moods, or whatever. It also collects HISTory information about the transforms, and the particular configurations that implied each, and their weights and locations. But very little of this information is used in the present version, which outputs only one completely stylized description of up to 7 most SALIENT ITEMS from TOOUT, entirely ignoring their structure, and any information in the cone. After the program is presented and "describing" is discussed we will add a bit of rather simple code to let RECODERS-1 output a wide variety of descriptive information in response to commands input to it.

Figure 3. OVERVIEW and RECODERS-1 Program

```
(OVERVIEW RECODERS-1. Describes, giving salient things found.
(Recognition Cone characterizes, transforms, names, gets qualities, using
(similar mechanisms as it collapses input scene layer by layer.
(Chooses names to output in description when choice is triggered.

START  Initialize LAYERS, CHARacterizers, each CHaracterizerI,          M1-MN
         and SPOTSIZE.
NEXT     erase Layer, Row, TOOUTput list, and apex cell for new scene.  MN+1
         set ITEMs in a description to equal 7.                         MN+2
IN       input a ROW and its TYPE and go to what's in TYPE ($TYPE)        1
(If TYPE is  S, will Sense the scene, Row by Row
SENSE  get each SPOT and store it under its Layer . Row . Column.       2-6
(If TYPE is TRANS starts TRANSforming, since the raw scene is stored.
TRANS  output that starting; initialize TODO to contain LAYERS and CHARS 7-8
T1       Get STEP size and NOWDO for next layer from TODO                 9
           and set DO to Erase the next layer, and go to T4               10
T5       Get and apply each TRANSform from NOWDO,                      11-16
           get the submatrix to look at and what to DO from in TRANS,    12
T4         get the TyPe, THRESHold and DESCRiption from in DO,           14
T3         get each THING and its relative locations and MINimum from DESCR 16
(TyPe A TRANSForm MERGEs all NAMEs in the cell into the next layer.
A1       MERGE the specified cell into the next Layer.                   17
(TyPe C TRANSform looks for a configuration over a THRESHold.
C1       If enough specified THINGs are found with VALues above MINimum 18-21
           to TOTAL above THRESHold,
         MERGEs IMPLIEDS and history of DO and GOT into next layer.      22
A2       Loop  through the specified sub-matrix until done, then go to T5. 23-24
(TYPE E TRANSform is set up by statement 10 to erase the next layer.
E1       Erase the cell in the next layer and go to A2 to loop through.  25
(When all NOWDO for a layer have been DOne, ITERates to next layer.
ITER   Add 1 to Layer, reduce Row and Column by STEP-size,             26-28
           and test if the cone has collapsed to its apex.  No - go to T1 29
             Yes - CHOOSE the HIWeighTed 'NAME' from the grand apex      30
(OUTput a description by CHOOSEing ITEMS (= 7) most SALIENT names on TOOUT
SALIENT  CHOOSE and output the hiweighted name on TOOUT,               31-32
           until ITEMS (= 7) names have been output, or TOOUT is empty,  33
           and go to INITialize for, and INput, the next scene.
(MERGE and CHOOSE are functions used by the program, and DEFINEd here.
MERGE    MERGE each NAME and WeighT from LISTA into location in LISTB  34-40
           TOTALing weights, and listing location (LISTB), and GOTA on HISTory. 38
           *CHOOSE triggers CHOOSEing the HIWeighTed name of the CLASS 37,40
CHOOSE   CHOOSE the HIWeighTed name of the CLASS specified from LISTA  41-50
(Input data cards follow the program.
```

```
(*Program RECODERS-1. Recognition Cone Describes giving SALIENT things found.
(Characterizes, transforms, names, and gets qualities.
(Collapses input scene layer by layer to apex, choosing names to output when triggered.
START  set LAYERS = (the list of transforms - including averaging, fading and      M1
            differencing, if desired - and characterizers is given here, layer by layer)
       set CHARS = (the list of characterizers to be applied after all layers)     M2
       set CI = (each characterizer must be given, with its description of         M3-MN
            pieces, threshold, and implieds)
       set SPOTSIZE = 1                                                            M(N+1)
NEXT   erase L, R, TOOUT and $(L . 0 . 0)                                          M(n+2)
       set ITEMS = 7                                                               M(N+3)
(INput a ROW and go to TYPE (SENSE, TRANSform, or a command)
IN     input TYPE, ROW]          [+to $TYPE. -to END]                               1
(Sense the input scene, ROW by ROW. Store each SPOT under Layer . Row. Column
SENSE  erase C                                                                     2
S1     from ROW get and call SPOTSIZE symbols SPOT. erase [-to S2]                 3
       in  $(L . R . C) list ] SPOT '1 * ; ]'                                      4
       C = C + 1 [to S1]                                                           5
S2     R = R + 1 [IN]                                                              6
(The raw scene has been stored in the first layer. Transforming starts.
TRANS  output Row ' SCENE HAS BEEN INPUT, IS BEING TRANSFORMED.'                   7
TREPEAT TODO = LAYERS CHARS                                                        8
(Cycle through TODO (which contains LAYERS and final CHARacterizers) till collapsed.
T1     from TODO get STEP NOWDO] = [- TREPEAT]                                     9
(Erase next layer with TYPE E operator                                             10
       DO = 'E X X X X ]]' [T4]
T5     from NOWDO get TRANS = [-ITER]                                              11
(Get the bounds of the sub-matrix through which to iterate the TRANSform to DO
       from $TRANS get RA CAA RMAX CMAX DO                                         12
T2     CA = CAA                                                                    13
T4     from $DO get TP THRESH DESCR ] IMPLIEDS ]]                                  14
       erase GOT TOTAL                                                             15
(Get the relative Distance, THING and its MINimum value to look for (if TYPE C)
T3     from DESCR get DR DC THING MIN = [+ $(TP 1) -$(TP 2)]                       16
(TYPE A TRANSform MERGEs all NAMEs in the cell into the next layer, MIN has weight.
A1     MERGE( $(L . RA + DR. CA + DC), '$(L+1. RA / STEP . CA / STEP)' ,           17
+         MIN ) [T3]
(TYPE C TRANSform looks for a configuration over a THRESHold.
C1     from $(L. RA + DR . CA + DC) get # that THING # VAL [-T3]                   18
       is MIN lessthen VAL? yes- TOTAL = TOTAL + 1 [T3]                            19
       on GOT list THING [T3]                                                      20
C2     is TOTAL lessthan THRESH [+ A2]                                             21
(THRESHold is exceeded, so MERGEs IMPLIEDS into the next layer.
       MERGE(IMPLIEDS , '$(L+1 . RA / STEP . CA / STEP)' , 1 , DO, Got)            22
(If CMAX contains C or RMAX contains R, $CMAX and $RMAX get grand Row or
(Column bounds.
A2     is CA lessthan $CMAX ? yes- CA = CA + 1 [+ T4]                              23
       is RA lessthan $RMAX? yes RA = RA + 1 [+ T2. - T5]                          24
```

*See Appendix, and Uhr (1973a), for explanations of program constructs.

```
(Operator that Erases next layer.
E1      erase $(L + 1 . RA / STEP . CA / STEP ) [A2]                    25
(ITERate till cone has collapsed into a one-cell apex.
ITER    L = L + 1                                                       26
        R = R / STEP                                                    27
        C = C / STEP                                                    28
        is R lessthan 1 ? is C lessthan 1 ? [-T1]                       29
(Apex reached. CHOOSE HIWeighTed NAME and OUTput SALIENTs on TOOUT.
        at start of TOOUT list CHOOSE($(L . 0 . 0), 'NAME')             30
+           HIWT L . 0 . 0 HIHIST  ]
SALIENT from TOOUT get CHOOSE ( TOOUT) WT LOC HIST ] = [- NEXT]         31
OUT     output 'THERE IS A ' CHOOSE ' WEIGHT = ' WT                     32
        is 0 lessthan ITEMS ? yes- ITEMS = ITEMS - 1 [+ SALIENT. -NEXT]  33
(MERGE LISTA into LISTB; add TOTAL WeighT, and HISTory of TRANSform applied
(and parts GOT
MERGE   DEFINE: MERGE(LISTA, LISTB, WT, TRANSA, GOTA)                   34
M1      from LISTA get ] NAME WTL HIST ] = [- RETURN]                   35
(If WT contain '$' set WT = TOTAL of pieces got
        from WTL get '$' = TOTAL                                        36
(IF NAME contains '*CHOOSE' go to CHoose
        from NAME get CLASS '*CHOOSE' [+ CH]                            37
        from $LISTB get ] that NAME TOTAL TRANSB ; HISTB ] = NAME TOTAL +  38
+         WT * WTL TRANSB TRANSA ; HISTB GOTA ] [+M1]
(LISTB contains the location where implied (but it's never used by this program).
        on $LISTB list ] NAME WT * WTL LISTB TRANSA ; GOTA ] [M1]       39
(CHOOSE the HIWeighTed NAME of the CLASS specified from this cell, and put
(on TOOUT.
CH      at start of TOOUT list CHOOSE($(L.RA.CA),CLASS)                 40
+       HIWT L . RA . CA  HIHIST ] [M1]
(CHOOSE the HIWeighTed NAME of the CLASS specified from LISTA.
CHOOSE DEFINE: CHOOSE (LISTA, CLASS)                                    41
CH1     from LISTA get ]CHOOSE HIWT HIHIST ] = [- RETURN]               42
          is CLASS EMPTY ? [+ CH2]                                      43
        from $CHOOSE get 'C=' LEFT # that CLASS # [-CH1]                44
CH2     from LISTA get ] ORNAME ORWT ORHIST ] = [- RETURN]              45
          is CLASS EMPTY ? [+ CH3]                                      46
        from $ORNAME get 'C=' LEFT # that CLASS #  [-CH2]               47
CH3      is HIWT lessthan ORWT ? yes- CHOOSE = ORNAME [-CH2]            48
        HIWT = ORWT                                                     49
        HIHIST = ORHIST [CH2]                                           50
END     [to START]                                                      -
S 01110111110]                                                          I1
S 00100110000]                                                          I2
S 01110111110]                                                          I3
TRANS I,C]                                                              I4
```

Scenes are Input one ROW at a time, each row starting with 'S ' (for Sense) and ending with ']' (which means that spaces, but not brackets, can be part of an input) (e.g., I1-I3). One Input card, which starts 'TRANS ' and, optionally, contains the correct names, or any other identifying information, and ends with ']' must follow each scene (e.g., I4).

The four Input rows shown (I1-I4) give a simple example of a Scene of the two objects, an I and a C . To recognize them, RECODERS-1 would need a number of characterizers that looked at configurations of their various parts. If the scene were larger, and more realistically noisy, additional pre-processing transforms would be needed to eliminate local noise (e.g., by transforming a 1 virtually surrounded by 0s into a 0), smooth, and in other ways regularize.

DESCRIBING AND DESCRIBING AND DESCRIBING

Let's briefly survey what's been done and the problems to be faced, and then examine what we might mean by a description.

History

A small but substantial number of papers have been written about programs that describe; and a few programs have been coded. These have been about a variety of different things - from finding blobs or edges in aerial photos to counting and identifying cells in microscope slides (e.g., Kirsch, 1964; Ledley and Ruddle, 1965) to looking for aberrant bone structures in X-rays (Ausherman et al., 1972); from recognizing the letters in continuous handwriting (Eden, 1968) and the phonemes in speech (Forgie and Forgie, 1959; Reddy, 1967) to finding a whole, like a face or a table or a 3-dimensional block, and describing

its parts, such as noses, legs, and edges (Guzman, 1968; Sauvain and Uhr, 1969). (See Uhr, 1973b, for a more detailed survey, and Lipkin and Rosenfeld, 1970, for more examples.)

But much of this work still has the strong feeling of beginnings. Each program attacks a particular problem and, worse, describes in a particular, pre-ordained way. Describing programs take an extremely important step beyond classical pattern recognizer-namers, which simply assign a single name to the input. But it is an extremely difficult step, and progress is slow. A namer outputs one symbol - the chosen name. As soon as a program outputs more than one name it's probably fair to call it a describer.

Describing Because Naming is too Difficult

But that admits some pretty simple-minded programs, for example ones that can't even decide upon a single name with much success, but can still manage to output several, saying in effect: "I think it's this, but it might be that, or even that other." Whenever a namer uses a set of characterizers, each of which implies one or more possible names (and this is the typical structure of a pattern recognizer-namer), it is trivial to have it output its characterizers as well as the chosen name - and this, again, is describing. In fact quite often a describing program is one that has not been completed to the point where it can actually name the objects, or even the single object, in its input (usually because it is dealing with especially complex objects); rather it can only characterize fragments, or look for specific objects, features, or aberrations of interest.

Sometimes characterizers will seem intuitively meaningful to us humans as descriptors - e.g. when they specify edges or curves or angles or loops. Sometimes they will seem weird, e.g. when they specify something like "a bit of darkness here and a bit of light there and a gradient down below". Sometimes they will seem complete gibberish, e.g. some arbitrary function that has been thrown at the pattern.

The Importance of the Hearer's Expectations

But how do we decide which are good descriptors? The obvious answer, and I fear the correct one, is that a descriptor is good to the extent that it conveys information to the hearer, the recipient of the description. If this is true, then we must introduce enormous new problems before we can handle description adequately, for we must make description a part of the communication process, and our programs must be cognizant of what has relevance, interest, and meaning to the hearer.

For the moment we can conventionalize the hearer, into a standard recipient of information. For example, we can flag some characterizers as publicly meaningful, as opposed to the others that have meaning for the program since they help it to recognize, and have the description output only the publicly meaningful characterizers. But we should remember that these are ad hoc conventions, and we should not push them too far.

Types of Descriptions

There appear to be a number of types of information that are, at least at times, appropriate for descriptions (for more discussion see

Firschein and Fischler, 1971; Uhr, 1972, 1973c). These include the possible names and the characterizers that succeeded on the pattern, as just discussed. Note that these can be used in a variety of ways. For example, the program can output the names that were definitely not implied, saying, "it's certainly not any of these." And it can use the characterizers to support its arguments for each name, by saying, "I think it's this name because of these characterizers, but these other characterizers suggest it might be this name instead."

But we need programs with richer structures in order to get a greater variety of descriptions, as indicated in Table 1, including some of the most basic sorts.

Table 1 about here

Probably the thing that most naturally comes to mind when we talk about descriptions is a structural statement about the parts of the object, and their interconnections. Thus we say it's the word BOX because it contains a B an O and an X (note that there's a hidden convention that we always assume without noticing, and that, unfortunately, we usually build like concrete into our programs' code - that these are concatenated from left to right in a linear string). And it's the letter B because it contains a left vertical line and a small loop going to the right from top to middle and a similar loop from middle to bottom. And it's a person because there's a foot and a leg and a hip and the leg is connected to the hip, and so on.

To do this sort of a thing we need systems that can recognize objects, and then recognize objects made up of these objects, and so on. These clearly go beyond the classical naming programs, and the work of this sort seems to me the most direct attack on describing. But at present this

research works only with very clean and simple drawings of scenes; it does not try to handle the messy, noisy scenes we get from the real world. In fact it virtually ignores the problems that the naming programs confront - recognition over a potentially infinite set of unknown non-linear transformations. When that must be done the program is confronted with major problems just in recognizing the parts, and these problems are compounded when there is a whole mess of other parts to be recognized simultaneously, and then higher-level wholes (which are themselves just parts) to be recognized. This is an interesting situation, since all this mess at the same time adds an enormous amount of contextual information that should help recognition. Ultimately we will be better off because of it, but for the moment we are just learning to handle it.

But there are still other things that are components of a good description, for example classes like animal or furniture, and qualities and moods like color, shading, texture, shadowing, firmness of line, boldness, goodness, and beauty - going, roughly, from simple and concrete to abstract and vague. Characterizers for color and for shading are relatively easy to code compared to texture and firmness of line, which turn out to be major problems in themselves. But forget about boldness, much less beauty, except to note again how much they lie in the eye of the beholder - our systems must learn what the hearer feels about things.

Directing and Choosing the Description

Anything in the raw input scene, or the outcome of any conceivable transform, might potentially be useful in a description. This ultimately depends upon the hearer, his interests, desires, and peculiarities. A crucial problem lies just in keeping the description down to a reasonable

Table 1. Some Possible Types of Descriptive Information

A.	Some types of things in a scene:	Examples:
	1. Unrelated collections	forest
	2. Related groups	dining set == table and chairs
		family == man, woman, child
	3. Objects:	
	a. wholes	tree, table
	b. sub-wholes	leaf, hand
	c. parts	seat, finger, fingernail
	4. Characteristics and measurements	straight edge, long vertical, angle
	5. Classes	table, chair $\in$ furniture, name
	6. Qualities:	
	a. shading, color, texture	dark, red, herring-bone
	b. concrete	jagged, faint
	c. abstract	forceful, beautiful
	7. Moods and tendencies	ominous, lively
	8. Things counter to expectations:	
	a. missing wholes, parts, characteristics, qualities, etc.	place-setting without fork, face without nose
	b. incongruities	3-eyes face; friendly Hitler
B.	Relations between things:	
	1. Co-occurrence	trees in forest, objects in a room
	2. Systematic placement	beacons at an airport
	3. Locational covering	field covers green, stubbled
	4. Characteristics and qualities	(transforms used and stored in history)
	5. Parts and contexts	(parts found and stored in history)

size, focussing, culling, and choosing only what is relevant. Table 2 examines some aspects of this focussing process.

Table 2 about here

RECODERS-1 follows built-in procedures for choosing its description. RECORDERS-1A, which we will examine now, custom-makes its descriptions to a hearer-user's commands. Systems that infer and learn how to make their descriptions more pertinent, as a function of conversational interaction with scenes and hearers, will be left as an exercise for the reader, and for future papers.

RECOGNITION CONES FOR DESCRIPTIONS

The recognition cone is an attractive vehicle for describing because it can handle any variety of transforms that are desired and can be coded (or, another story, can be learned through experience), including features like edges and curves, qualities like color and texture, parts like eye and leg, and pre-processors like averaging and differencing. These can all be assessed by their particular characterizing transforms, and their outcomes stored, along with implied output names and all other transforms, in the next layer of the cone.

The cone can also be used to superimpose several different kinds of structure on this stew of descriptors as summarized in Table 1.

Table 2. Sources of Information to Help Direct and Choose the Description

A. Types of information leading to choices:

1. Strength of implication
2. Level of the object:
   a. highest
   b. lowest
   c. most "relevant"
3. Covering of the scene:
   a. most parts accounted for
   b. least redundancy
   c. most coherent and relevant "story"
4. Specified classes, characteristics and qualities expected
5. Relevance
6. Coherence and harmony, all parts adding to an interesting description

B. Sources and reasons for choosing the specific objects in a description:

1. Built-in (e.g. "6 most highly implied animals")
2. Commanded (e.g. "ALL WHOLE OBJECTS")
3. Inferred, or learned, from:
   a. scenes (present scene; previous scenes)
   b. commands and conversation (present ones; previous ones)
   c. conceptions of "pertinence" and "relevance"
      i. built-in
      ii. inferred from this scene and/or the commands
      iii. inferred from a learned model of hearers, or of this hearer's expectations

RECORDERS-1A, WITH ADDITIONS TO DESCRIBE IN A VARIETY OF WAYS

Let's look now at additions to RECORDERS-1, to allow it to describe in a variety of ways, and thus make use of the potentially descriptive information that it collects while transforming the scene through the recognition cone.

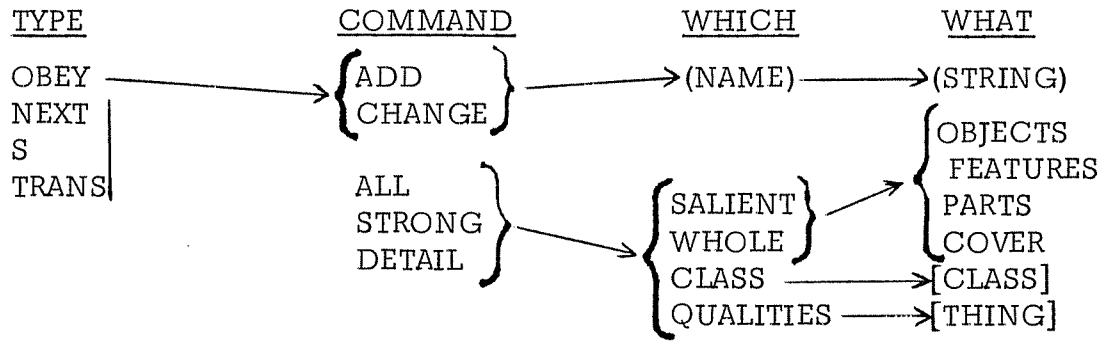
These additions allow a user to conversationally input a sequence of commands (each must start with OBEY, to direct the program's description). Each command specifies any of a variety of different requirements, in terms of a) the ordering of objects in the description (e.g. most SALIENT in the weights of their implications, or starting with the largest WHOLEs), b) the strength of implication of objects output (eg. ALL, or only those STRONG enough to have reached a DECIDE level), c) the structure of the description (eg. just OBJECTS, or objects plus their FEATURES or PARTS, or objects plus other things that they COVER), and d) the types of things (eg. of a specified CLASS or QUALITIES).

In addition, the user can command RECORDERS-1 to CHANGE or ADD to any list, so that he can, eg., change the number of ITEMS he wants in a description (the program itself sets ITEMS = 7), or the DECIDE level, or he can add operators to the LAYERS or CHARacterizerS lists, and create now operators (by specifying OBEY CHANGE [newname] [newcontents] - which changes the contents of this newname, e.g. CHAR193, from nothing to the newcontents).

Fig. 4 goes here

Figure 4. Additons to RECORDERS-1 to Command it to Describe

The following show the possible TYPEs of input. ALL COMMANDs start 'OBEY'.



Every input must specify a TYPE. TYPE = OBEY indicate a COMMAND, which must also have a WHICH and a WHAT. Many combinations are possible, as indicated by brackets and arrows above. (eg ALL WHOLE COVER; DETAIL SALIENT OBJECTS; ALL CLASS [CLASS] (ie. the actual classname desired). The following code changes handle all possible combinations:

(RECORDERS-1A. Changes that input and OBEY a variety of COMMANDS	RECORDERS-1 M(N+4)
DECIDE = 30	
output CHOOSE ' IS MOST HIGHLY IMPLIED. COMMAND MORE' [IN]	30.1
(OBEY signals a COMMAND specifying WHICH and WHAT to describe.	
OBEY from ROW get COMMAND WHICH WHAT [$COMMAND]	30.2
(COMMAND CHANGEs or ADDs to memory list (eg. ITEMS, LAYERS, or an operator)	
CHANGE set $WHICH = WHAT [IN]	30.3
ADD on $WHICH set WHAT [IN]	30.4
(Responses to COMMANDs that specify kinds of descriptions	
(To output ALL things on TOOUT	
ALL ODECIDE = 0 [$WHICH]	30.5
(To output only STRONGly implied - WeighTabove DECIDE (=30)	
STRONG ODECIDE = DECIDE [$WHICH]	30.6
(To output all DETAILs in apex cell as well as TOOUT. (A detail starts ']')	
DETAIL on TOOUT set $(L . 0 . 0) [ALL]	30.7
(WHICH contains WHOLE, CLASS or QUALITIES, or SALIENT	
(TOOUT is ordered from apex to row scene, so WHOLEs are output first.	
WHOLE from TOOUT get CHOOSE WT LOC HIST] = [+ OUT. - IN]	30.8
(Output descriptors that belong to one of the designated class THINGS in WHAT	
CLASS from $CHOOSE get 'C=' CLASSES]	30.9
HIST = CLASSES [QUALITIES]	30.10
(Output descriptors that have one of the designated THINGS in their HISTory.	
QUALITIES set THINGS = WHAT	30.11
Q1 from THINGS get THING] = [- $WHICH]	30.12
in HIST get # that THING # [+OUT. - Q1]	30.13

```
(Get SALIENT things, ordered by weights, from TOOUT.
SALIENT   from TOOUT get CHOOSE(TOOUT) WT LOC HIST ] = [+ OUT. -IM]   31.V
OUT       output CHOOSE ' IS ' WHAT ' = ' THING ' LOC = ' LOC ' WT = '   32.V
+             WT [$WHAT]
(Will quit  when has output the specified number of ITEMS
OBJECTS   is 0 lessthan ITEMS ? yes - ITEMS = ITEMS - 1 [+ $WHICH. -IN]  33.V
(WHAT contains FEATURES or PARTS (got from HISTory), COVER (got from cell where
(chosen) or just OBJECTS
(Only the FEATURES transforms are kept in HISTory
FEATURES  from HIST get HIST ; [PARTS]                                   33.1
(Get PARTS of PARTs (from HISToryParts) down to lowest level.
PARTS     from HIST get PART = [- OBJECTS]                               33.2
          from TOOUT get ] that PART WT LOC HISTP                        33.3
          on HIST set HISTP [OUT2]                                       33.4
(COVER looks for general lists of ALLQUALs and RELATA, and HIST, in cell
(where chosen.
COVER     on HIST set ALLQUALS RELATA                                    33.5
COV1      from HIST get PART = [- OBJECTS]                               33.6
          from $ LOC get ] that PART #[- COV1]                           33.7
OUT2      output 'IT IS FURTHER DESCRIBED BY ' PART [$WHAT]              33.8
```

NOTE that the user must now input NEXT] to start processing a new scene

A Descriptive Discussion of RECODERS' Various Descriptions

Rather than output up to seven most SALIENT things on TOOUT, as RECODERS-1 does, RECODERS-1A outputs the single most highly weighted CHOOSEn from the grand apex (statement 30.1), suggests that the user COMMAND MORE, and goes to INput (either a command or the NEXT scene). All commands start with OBEY as their TYPE, and 30.2 gets the COMMAND, WHICH and WHAT from the input ROW, and goes to the name in COMMAND.

If this name is CHANGE or ADD the string in WHAT is stored under the name in WHICH, or added to its end. This is an extremely simple technique for giving the commander control over any of the program's memory lists (and it is rather reminiscent of the kind of "learning" that has been called "advice-taking" - see e.g. Raphael, 1964; Uhr, 1973b chapter 19). For example, the command:
OBEY CHANGE ITEMS 16]
will have ITEMS = 16 replace the ITEMS = 7 for this scene (30.3). And
OBEY ADD CHARS C123]
will put C123 at the end of the CHARS list (30.4). (Another CHANGE command might be used to create C123 - the change can be from nothingness.)

If the COMMAND name is a) ALL, b) STRONG, or c) DETAIL RECODERS-1A will set the Output DECIDE level to a) zero (so that all things are output) (30.5); or to b) DECIDE (which is set at 30 - unless a CHANGE command has changed it - so that only things above that rather high level are output (30.6); or c) it will put the apex cell, which contains masses of detailed information that has been carried along, at the end of TOOUT (30.7), and go to ALL, to set ODECIDE = 0.

RECODERS-1A now goes to a) SALIENT, to get CHOOSEn things ordered by weight (31.V), or to b) WHOLE, to get things as they are ordered on TOOUT (30.8), which tends to have larger wholes first; or to c) CLASS, to output only if one of the classes listed in WHAT is stored as this CHOOSEn thing's class (30.9 - 30.10), or to d) QUALITIES, to output only if one of the things listed in WHAT is found in CHOOSEn's HISTory (30.11 - 30.13).

The object chosen, along with its LOCation and WeighT, are output (32.V), and RECODERS-1A goes to what's under WHAT, to see if any additional information has been requested.

PARTS gets each PART from the HISTory of the object, and gets its parts (HISTory-of-Parts) from TOOUT (33.2-33.4), cycling through and OUTputting all the PARTs (33.8). FEATURES looks only for the transforms (stored before the ; in HISTory) (33.1). COVER looks for all things in HISTory, and also in the additional lists ALLQUALS and RELATA (which must be initialized in the program's memory, or input by user commands, to contain the types of things desired in a description of the sub-cone covered by this object), and it outputs all those found in the sub-apex (33.5-33.8). If OBJECTS is specified, only the main objects got from TOOUT are output (33.V, 32.V) without any of this additional information.

Simple Improvements for Better Descriptions

More things could be done. FEATURES gets just the transform operators used, whereas PARTS gets both parts of configurations and the operators. Slight changes would add the third option, of getting just parts. To minimize lines of code I have features and parts treated the same, which means 33.3 wastes effort looking for parts of features (and for the semi-colon (;) that separates them) which it will never find in

TOOUT. The reader with the perseverance to follow the code in detail will find other places where efficiency might be improved.

We could easily have RECODERS-1A use the strength of weights to choose the most important parts and qualities, just as it now chooses the most important objects, and use a set number of items in describing an object. It could also note when several objects have the same parts or qualities in common, and it could choose objects as a function of combinations of classes, parts and qualities.

DISCUSSION

Level and Variety of Detail

To the extent that we enrich the possibilities for description we enormously lengthen our descriptions. It seems quite likely that, as with information retrieval systems, the biggest problem will be how to keep from overwhelming the hearer with descriptions ad nauseam, often with only minimal, albeit demonstrable, pertinence. This once again indicates the issues of pertinence to the hearer, and how the program can be told about that. Since one hearer's pertinence is another hearer's nonsense, it seems necessary that the program must learn about hearers, or at least obey hearers' commands, again with much the same problems of an information retrieval system that must input and attempt to "understand" users' requests.

Is there any such thing as a "standard" description? The only one I can think of is an exhaustive description. For anything else would have to include an arbitrary choice of objects - e.g. members of a "named objects" class, or named objects and their parts, features and qualities, or named objects implied above some threshold.

The exhaustive description would simply be the contents of all the cells in all the layers of the recognition cone (or whatever other transforms a program used). This includes all the spots of dust (for those scientists who study dust), in the first, input layer, and all possible combinations thereof, along with all the interconnections that can be inferred from history, classes, and relative locations in the cone. Who is to deny the compulsive hearer who wants the input enumerated cell by cell? Or what if the input is indeed a code with no redundancy from cell to cell (eg. a computer tape)?

Steps Toward Relevance

Recognition cones like RECODERS-1A that can obey a variety of commands, by outputting the appropriate descriptive information, are capable of almost any kind of description, If anything else is desired, they need only be given new transform operators capable of characterizing these new kinds of descriptive information. That is, anything measurable can be used, and made accessible for a description. If a Configuration is not sufficient a new transform type can be coded to compute the desired function. The cone structure itself organizes this information, by structure, regions, classes, and co-relations.

RECODERS-1A has several techniques for keeping the description pertinent, and relatively short: a) the number of ITEMS specified sets an outside limit; b) the DECIDE level (which the ALL command can lower to 0) outputs only items that have exceeded it; c) special CLASSes and QUALITIES of things can be insisted upon. But these are only first steps. A good system would be able to mix these, and other, criteria together under a general assessment of relevance of each item. It should not slavishly obey a command, but rather infer, through con-

versational interaction - which includes feedback that changes the course of the description as it goes along - what is most relevant to the present hearer.

It should further build up an internal conception of pertinence. A step in that direction could be made by putting the various command and decision parameters on internal lists, each with a weight. Then, as a function of feedback as to whether a response was deemed relevant by the hearer, the weights of the parameters involved with that response could be raised and lowered. (This is much like the parameter adjustment we see in pattern recognizers like Marrill and Green, 1960; and in Samuel's checker player, 1959, 1969 - except at a higher level, as in Uhr and Jordan, 1969; see Uhr, 1973b chapters 16, 21.) Now the particular type of response could be chosen with probabilities that reflect these weights. General weights could be induced for all hearers, and particular weights for the present hearer, and for well-known hearers, could also be used when the particular hearer is known.

Improving Serial Representations of Parallel Systems

Recognition cones simulate parallel-serial converging systems of the sort we see in the brain, and especially in the visual system. When embodied in hardware (neurons connected by synapses, or electronic switching elements connected by threshold elements) it is not unreasonable to have a lot of information kept, since information can be stored in any element, and, since the element is physically present in any case, it might as well be working rather than idle. But when simulated on the serial digital computer, such a system can eat up valuable memory space.

RECODERS-1 is wasteful in this respect. It not only keeps all details of information in all cells at all layers of the cone, but it also redundantly passes this information along to the next layer, to make it easily accessible (if we place, as I recommend doing, a general averaging operator at the start of each layer's transformation). But this can easily be changed. Averaging need not always be done, or it can be done only across members of key classes (e.g. the external names, in order to be able to choose among them, and to get descriptive information, and low-level qualities, like edges, in order to smooth and in other ways pre-process). Better - a threshold can be used to keep low averaged values from being passed along to the next layer.

Another technique would allow an operator to specify a layer as well as the row and column positions, so that it could reach back to any prior layer in the cone to make its test. This would allow the system to keep things where they were, passing along only the minimum, of external names and pre-processings.

In any case, it is not possible to effect any enormous saving of space in this way. The cone converges from its base, which must be large enough to adequately resolve the input scene. The total cone will not be a very large multiple of just the base. The larger and more complex the set of operators used to process the sharper should be the convergence STEP sizes, so that the total space needed for operator outcomes and cells remains in some balance.

We might think of erasing a layer as soon as it has been processed (as in Uhr, 1972), and this indeed keeps the working memory as small as possible. But an especially interesting and important extension would allow RECODERS-1 to handle inputs that continue and change over time (as in Uhr, 1973c), and to do this the cone should remain, with things gradually fading away when no longer seen, new

things emphasized, and moving things reinforced.

Choosing and Using Descriptors

RECODERS-1 adds to its list of things TO OUTput when triggered by a *CHOOSE in a cell. It then culls its description from TOOUT. Only when it is asked for DETAILS does it look also into the grand apex cell. Alternately, chosen objects could simply be put into the apex, or passed along through the cone (tagged as chosen), so that the apex cell contained everything, and could be used directly, as a TOOUT list. This clutters up the processing, since the system must look for pertinent things to output among a much larger list. But it gives a simpler structure, and would allow the system to treat chosen objects and the unchosen in the same way, for example so that, in changing its point of view in order to get a sensible cover, it could decide to re-choose from among the unchosen with the now-deemed-appropriate features, parts or classes.

Conversational Descriptions that Affect Recognition

Another alternate would have the system output as soon as it chooses. This would make for more rigid and stylized descriptions - unless it was capable of inputting the hearer's response, and having that direct its actual transforming operations and not, as in the commands input by RECODERS-1A merely direct its culling of the TOOUTput list. Now there must be a pause in processing after each output, to get a command response and use that to focus its processes - e.g., to search for a certain object or class of objects, or to describe details of a certain type.

This quite nicely fits into RECODERS-1A, since we merely need to have it output the CHOOSEn object rather than add it to TO OUT (40),

and then go to IN, to get and execute a command (we could extend this ability to allow it to use back-links from names a command suggests it focus on to get characterizers, and to add these and other specified operators to its current NOWDO list), and return to where it left off processing (see Uhr, 1973c; 1973b, chapter 8).

Conversational Descriptions Over Time

This does violence to any reasonable conception of time. When a command is received the whole world stops, and only after it has been executed and all adjustments made do things start up again, right where they left off.

A realistic system needs to spend time recognizing and understanding the command (see Uhr, 1973d), and by that time the scene will have changed some, and the transforms in the cone will have changed. That is, perception should continue, in real time, and only after enough time has passed to decide to a) choose a descriptor, b) actually output it, c) wait for a response, d) input and e) recognize and understand it, and f) effect any internal changes, so that processing is now a function of this output-input interchange, can the system return to processing - and by that time the world will have changed (unless it is immovably static).

Complete Cognitive Systems that Use Recognition Cone Describers

Once conversational commands can affect processing, in a system that handles environments that change over time, we are in a position to let other forces come into play, by putting this perceiving system into a larger cognitive system. Rather than simply trigger names and other descriptors to output, the system can also trigger acts (e.g., find or move an object) and internal searches (e.g. answer a query,

or deduce a solution). Then the action, or answer, when output, leads to responses and other repercussions (e.g. a different perspective, if the system has moved itself) within the interactive conversational framework that has already been established to accept commands (see Uhr, 1973d, for a simple system to do some of these things).

SUMMARY

This paper describes "recognition cones" for parallel-serial perceivers that describe scenes of interacting objects.

The concept of a "description" is examined, for the variety of things that it might mean.

A recognition cone program is presented, and code is added to allow it to output a wide variety of descriptive material, as commanded.

The commands serve to limit the size of the description, and thus help to fend against a major problem of descriptions - that any quality, detail or combination of details may be pertinent from some point of view, whereas to communicate successfully a description must be kept down to absorbable size, and must contain the pertinent and relevant information. To do this properly a good describer must make subtle decisions as to relevance, including decisions as to what it infers would interest the hearer of the description.

REFERENCES

Ausherman, D. A., Dwyer, S. J., and Lodwick, G. S. Extraction of connected edges from knee radiographs. IEEE Trans. Comp. 1972, 21, 753-758.

Eden, M. Handwriting generation and recognition. In: Recognizing Patterns (P. A. Kolers and M. Eden, Eds.) Cambridge: MIT Press, 1968.

Firschein, O. and Fischler, M. A. Describing and abstracting pictorial structures. Pattern Recognition, 1971, 3, 421-443.

Forgie, J. W. and Forgie, C. D. Results obtained from a vowel recognition computer program. J. Accoust. Soc. Amer., 1959, 31, 1480-1489.

Frishkopf, L. S. and Harmon, L. D. Machine reading of cursive script. In: Proc. 4th London Symposium on Information Theory (C. Cherry, Ed.). London: Butterworth, 1961, 287-299.

Guzman, A. Decomposition of a visual scene into three-dimensional bodies. Proc. AFIPS FJCC, 1968, 33, 291-304.

Kirsch, R. Computer interpretation of English text and picture patterns. IEEE Trans. Comput., 1964, 13, 363-376.

Ledley, R. S. and Ruddle, F. H. Chromosome analysis by computer. Sci. Amer., 1965, 208, 40-46.

Lipkin, B. and Rosenfeld, A. Picture Processing and Psychopictorics. New York: Academic Press, 1970.

Marrill, T. M. and Green, D. M. Statistical recognition functions and the design of pattern recognizers. IRE Trans. Electron. Comput., 1960, 9, 472-477.

Mermelstein, P. Computer Recognition of Connected Handwritten Words. Unpubl. Sc.D. Thesis, MIT, 1964.

Raphael, B. A computer program which "understands", Proc. AFIPS FJCC, 1964, 25, 577-589.

Reddy, D. R. Computer recognition of connected speech. J. Acoust. Soc. Amer., 1967, 42, 329-347.

Samuel, A. L. Some studies in machine learning using the game of checkers. IBM J. Res. and Devel., 1959, 3, 210-229.

Samuel, A. L. Some studies in machine learning using the game of checkers, II. Recent progress. IBM J. Res. and Devel., 1969, 11, 601-617.

Sauvain, R. and Uhr, L. A teachable pattern describing and recognizing program. Pattern Recognition, 1969, 1, 219-232.

Uhr, L. Layered "recognition cone" networks that preprocess, classify and describe. IEEE Trans. Comput., 1972, 21, 758-768.

Uhr, L. A Primer for EASEy (an Encoder for Algorithmic Syntactic English that's easy). Computer Sciences Dept. Tech. Report, Univ. of Wis., Madison, 1973a.

Uhr, L. Pattern Recognition, Learning and Thought. Englewood-Cliffs: Prentice-Hall, 1973b.

Uhr, L. The Description of Scenes Over Time and Space, to be submitted for publication, 1973c.

Uhr, L. Recognizing, "Understanding," Deciding Whether to Obey, and Executing Commands, to be submitted for publication, 1973d.

Uhr, L. and Vossler, C. A pattern recognition program that generates, evaluates and adjusts its own operators. In: Computers and Thought (E. Feigenbaum and J. Feldman, Eds.). New York: McGraw-Hill, 1963, 251-269.

Uhr, L. and Jordan, Sara, The learning of parameters for generating compound characterizers for pattern recognition. Proc. 1st Int. Joint Conf. on Artificial Intell., 1969, Washington, 381-416.