WIS-CS-73-173
The University of Wisconsin
Computer Sciences Department
1210 West Dayton Street
Madison, Wisconsin 53706

Received February 1973

# RECOGNIZING, "UNDERSTANDING," DECIDING WHETHER TO OBEY, AND EXECUTING COMMANDS

by

Leonard Uhr

Computer Sciences Technical Report #173

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), (NGR-50-002-160) and the University of Wisconsin Graduate School.

# RECOGNIZING, "UNDERSTANDING," DECIDING WHETHER TO OBEY, AND EXECUTING COMMANDS

Leonard Uhr

## ABSTRACT

This paper examines a programmed model (called DECIDER-1) that 1) recognizes scenes of things, among which are a) objects and b) words that form commands (or questions or other types of statements), 2) recognizes the import of these commands, 3) decides whether to obey one, and then 4) uses the command to guide the consequent actions, along with any necessary perceptual search.

It uses the same mechanisms to handle a) the perceptual processes involved with recognizing objects and describing scenes, b) the linguistic processes involved with parsing sentences and understanding their meaning, and c) the retrieval processes needed to access pertinent facts in pertinent memory. This is in sharp contrast to most of today's systems, which receive the command through one channel, to be "understood" by one special-purpose set of routines, and perceive their environments through an entirely different channel.

DECIDER-1 continues to pattern characterize, parse symbol strings, and access facts implied by input questions until an action is chosen, because it is sufficiently implied by this search through the memory net. Then it executes the implied action. Possible actions include Naming, Describing, Finding, Moving, and Answering queries.

The programmed model, DECIDER-1, is presented in EASEy (an Encoder for Algorithmic Syntactic English that's easy, Uhr, 1973a) so that we can examine exactly what happens.

# RECOGNIZING, "UNDERSTANDING," DECIDING WHETHER TO OBEY, AND EXECUTING COMMANDS

Leonard Uhr

## INTRODUCTION

The basic purpose of this paper is to examine how word-things recognized in the external environment can trigger actions that include recognition and manipulation of other, object-things, in the same environment. To do this we must examine several issues that have scarcely been touched upon in the research literature of psychology or computer science:

How does a system recognize that a perceived input is a symbol? How does it understand the import of a structure of symbols, eg. that it is a command, or a question? How does it understand the meaning of that structure? How does it decide what to do - whether to obey a command, and which command, or whether to continue to perceive, to respond to internal needs, or external presses from perceived objects? How do the understanding of commands and other symbolic percepts and the recognition of perceived objects interact, helping one another; eg. how does it recognize which objects are appropriate for carrying out a command? How does a system choose and execute the appropriate response, from a variety of possible responses?

These are extremely complex and subtle matters, and the work presented here is only a first step. But there appear to be several ways in which we can extend present-day programs, to begin to handle them.

## DESCRIPTION OF THE PROBLEM, BACKGROUND AND MOTIVATION

We will examine systems that a) input streams of information from the environment, b) attempt to recognize things, including symbols, and larger structures of things, c) decide whether to continue in this perceptual process or to respond, and d) generate the appropriate response.

Let's look at recent research with chimpanzees and with robots for some examples.

### Chimps that Obey Commands

Two chimps have been taught (by Gardner and Gardner, 1969, 1971, and by Premack, 1970, 1971) to learn vocabularies of over 100 words, and to learn to use these words in simple sentences. One of the most interesting things Sarah (Premack's chimp) learned was to respond to a sentence like "SARAH PUT BANANA BOX" by going to the banana, grasping it, carrying it over to the box, and dropping it into the box. SARAH is in an experimental room with a number of objects, including a banana and a box, but also such things as a pail, an apple, a cracker, and other objects, and a board on which the experimenter places the words SARAH, PUT, BANANA, and BOX. (These "words" are colored nonsense shapes: all previous attempts to get chimps to talk failed because of their limited vocal apparatus.) Thus words, sentences and objects are perceived visually, with words static objects much like Chinese ideograms.

Words are things that have symbolic significance. They come in through the same input channel as objects like the (image of the) banana. The chimp responds to the command, which is a

structure of several word-things, by manipulating the objects to which the words refer: Sarah grasps the *BANANA, not the symbol BANANA. (I will use stars (*), as in *BANANA, *BOX, *APPLE, to indicate the actual object, as opposed to the word.)

## Today's Programs Cannot Model such Chimps

The above description may appear to belabor the obvious. But although any three year old human child, and now two chimps, can successfully sort out words and objects, recognize structures of words as commands, understand the import of the commands, and act appropriately, we have little idea how all this is done. Psychologists have given us no theoretical models, and even the most sophisticated of computer programs for pattern recognition or language manipulation are designed to handle only small pieces of this process.

A sophisticated pattern recognizer (eg. Uhr and Vossler, 1961; Andrews et.al., 1968; Munson, 1968; Zobrist, 1971) can correctly name many different examples of a pattern, all distorted in extremely complex, and unanticipated, non-linear ways. It can contend with very complex possible confusions between objects. In comparison, a *BANANA and an *APPLE are very different, and the symbols Premack has chosen for BANANA and for APPLE are also different enough so that the chimp has no perceptual difficulties. (This is not at all to say that the chimp and human are not sophisticated pattern recognizers, but simply to point out that the particular single-pattern problem we are examining, in which only a few very different patterns need be recognized, is simple compared to problems that programs, as well as chimps

and people, can handle.) But today's pattern recognizers will not handle scenes of several things, much less compose things into structures like sentences, or sort out the words and the objects, and their interrelationships.

A sophisticated language processor (e.g. Shapiro and Woodmansee, 1969; Winograd, 1971; see Simmons, 1965, 1970) can handle far more complex grammatical structures than the simple actor-act-object-indirect object of our example SARAH PUT BANANA BOX. But it will handle only clean, clear-cut sentences. It cannot contend with sentences with mis-spelled words and noise that are embedded in larger scenes that include other objects. Nor can it begin to touch the problem of applying the sentence to the objects.

At the very least, pattern recognizers must be extended to handle scenes of objects, and to parse word-phrases as well as characterize parts of objects (see Sauvain and Uhr, 1969; Jordan, 1971; Uhr, 1972, 1973c).

## Robots that are Programmed to Obey Commands

Recent work with robots (e.g. Nilsson, 1969; Raphael, 1968; Feldman et al., 1969) has made some steps toward this interaction. But it has the flavor of interconnecting several separate black boxes, each performing a separate function, even though there often appears to be a great amount of overlap. Thus robots input and handle commands and perceptual environment completely separately.

Today's robots use extremely complex systems of computer programs, almost always separated into four major sub-systems,

to 1) understand the command, 2) recognize and build up a model of the objects in the environment, 3) deduce the set of actions appropriate to carrying out the command, and 4) apply them to the environment. Each subsystem has its own subsystems:

1) The command is given through a teletype, and the system applies a special set of language "understanding" programs, to recognize its words, parse the command statement, and turn this statement into a goal state that the robot must achieve, to obey the command.

2) Separate perceptual programs are used to handle the robot's perceived spatial environment, which almost always includes inputs from a tv camera, and occasionally is supplemented with inputs from range finders, touch sensors, photocells, or sonar. These programs input the sensed information, take a sequence of pre-processing steps (eg. to eliminate noise, find gradients and edges, and begin to connect short edge fragments into lines), and try to build the salient features up into something that matches some internal description of some object (e.g. Brice and Fennema, 1970). This results (if all goes well - which today means if objects are sufficiently simple, with enough background space between them, painted in colors that contrast sharply enough in the black-and-white tv image, and amply lighted) in the assignment of names to objects, and the assignment of these objects to their locations within the perceived environment.

3) A deductive problem-solving program (often a theorem prover) is used to search for a sequence of actions that the robot

might apply to the objects it thinks it has recognized in its perceived space, in order to achieve the desired goal state (e.g. Fikes and Nilsson, 1971).

4) This entails binding the objects (including the robot itself) to the inferred actions, so that the robot knows that it can in the real world carry out an action that it has deduced would be a stepping-stone toward carrying out the overall command.

## But Real-World Commands Must first be Perceived

In the real world there is no separate input channel for a command, and there is no god-given a priori signal, known to commander and slave alike, that says, "this is a command," "this is a question," and so on. A question-answering program has built into its guts that inputs will be questions, and it is straightforwardly pre-programmed to answer them. A robot program similarly has built into it that teletype inputs are commands, and that it is to carry them out, by manipulating the environment that it perceives through its sensory inputs.

But in the real world the command always comes in through sensory input channels. In fact the command is itself a complex structure of perceived objects. Eg the command, "PUT THE BANANA IN THE BOX", is made up of a string of words that are further structured grammatically and, more importantly, semantically (in that they refer to things like bananas, relations between bananas and boxes, an understanding that bananas can go into certain boxes, and the implied action of the entity being commanded - that it should put the banana in the box). And each word is made up of parts (letters if written, phonemes if spoken), each letter and phoneme is made up of parts, and so on.

The command may sometimes come through a separate perceptual mode channel, as when it is spoken, and refers to visually perceived objects. But this is not the issue, for commands and objects often come in through the same channel, with the receiver hardly noticing. It is not the difference in channel that allows the human to infer "this is a command of symbolic words" and "that is an environment of objects". Rather, the hearer first recognizes the parts of the command as objects and only later as symbols, combines them up into larger and larger structures, and recognizes that these structures have symbolic import, and are commands.

## Deciding Whether to Obey Which Command

Closely related, the hearer has no built-in understanding that it will receive one command at a time, pointed to and surrounded by special symbols. Rather, since it must infer that parts of its perceived environment are commands, it always has the possibility of receiving more than one command. It must decide whether to obey a command, and which. And for any, real-world receiver there will always be the issue of whether it should drop everything to carry out whatever command it has just received and understood, or whether it should do something else - what it was already doing, or what will best satisfy some internal need (eg. hunger) that is arising, or what will best respond to some interesting new object (eg. a steak).

Thus a system should be able to decide which from among a set of alternative action-sequences it wants to carry out, whether to obey this command or that, or to satisfy its own needs or goals.

## Key Problems Being Handled

This paper focuses on the problem of handling fields of things, some of which are symbols, where these things combine into larger structures (eg. eyes, nose, mouth, chin combine into face; SARAH PUT BANANA BOX combines into a command; B, O, X combine into BOX), some of which imply that the system should respond with am implied action upon other things to which reference is made.

We have discussed one example of such a situation, when the environment contains something like:
*CRACKER SARAH PUT APPLE PAIL *BOX *BANANA *APPLE *PAIL (or)
*BOX3 ROBOT PUSH BOX2 NEXT BOX1 *BOX4 *BOX1 *BOX5 *BOX2

Such a system can handle a number of other problems, for example an input like:
*CRACKER *BOX DESCRIBE THIS SCENE *BOX *BANANA *APPLE *PAIL
(in which case it must recognize the command DESCRIBE THIS SCENE and as a result output CRACKER BOX BOX BANANA APPLE PAIL).

Or it can be given an input like:
*CRACKER * BOX FIND A BOX *APPLE * BANANA * PAIL
(in which case it must recognize the command FIND BOX and as a result output something to indicate the *BOX has been found.) Or it can be asked to Move an object, or to Answer a query.

Examples of Name, Describe, Find, Move, and Answer are examined following the program.

## Simplifications

In order to focus on these issues and problems, we make the following simplifications:

A) Patterns and recognition techniques are kept very simple. They occur in one dimension, in a single sense modality, without overlap. Characterizers are weak, and are applied serially. (For sophisticated pattern recognizers, see Uhr, 1973b. For an appropriate basis for scene describers, see Uhr, 1972, on "recognition cones.")

B) Simple commands are used, and handled by very simple parsing rules. (For sophisticated parsers, see Simmons, 1970; Winograd, 1971; for better perceiver-parsers see Uhr, 1972.)

C) Learning is ignored; rather, the system is given a set of "TRANSforms" on an "IDEAS" list that includes all the perceptual characterizers, parsing rules, memory associations, and action sequences. (For learning, see Uhr, 1973b, chapters 15-21, and Uhr and Kochen, 1969.)

D) Only the very simplest of acting is done - just enough to show where and how actions fit.

E) Actions are not sufficiently integrated with each other, or with perception. But note that the single IDEAS memory homogeneously handles all actions. And a single mechanism is used to characterize patterns, parse phrases,and retrieve facts stored in memory.

F) Only a few memory transforms, from the many needed, will be shown in the examples of behavior that follow the program.

10 - Uhr

## "DECIDER-1" - A PROGRAM THAT RECOGNIZES OBJECTS, WORDS, AND COMMANDS

The following program, DECIDER-1, demonstrates how pattern recognition, parsing, and fact retrieval can be done in a homogeneous way, on scenes of objects, leading up to a decision as to what action to perform, where a variety of actions are possible, including finding, manipulating, naming, describing, and answering.

### Mechanisms

To keep things simple, only 1-dimensional strings are given DECIDER-1, e.g.

*BOX SARAH PUT BANANA PAIL *APPLE *BANANA *JAR *PAIL 1

*BOX DESCRIBE THIS *BANANAS *BOARD 2

### Characterizers

The characterizers used to pattern recognize are all of the same form, and intermingled with, the transforms used to parse, to trigger responses, and to respond (they are all called "TRANS"forms and stored on a single list called "IDEAS"). These characterizers are over-simple, since they merely look for a set of separate parts, ordered, where a threshold and weights determine how many need be present to succeed.

### Rewrite Rules

Parsing rewrite rules can be tabled quite nicely in DECIDER-1's memory, since each part can be either a class (eg. NOUN or FRUIT) or a thing (eg. THE or *BANANA). If the threshold is set to the sum of weights, all parts of the characterizer must be found.

Concatenation is not demanded (through it would be quite easy to give characterizers the option to do so - see Uhr, 1971, 1973b, chapters 3, 4). But I feel that this is an advantage, since in the real world, where word-sentences and other objects are intermingled, and where perfect sentences cannot be expected (just as perfect patterns cannot be expected), objects and grunts will often intrude upon the words. For example, DECIDER-1 should handle an input like:
SARA UHHH PUT *PEAR BANANA PAIL *APPLE *BANANA *JAR *PAIL 1.V
The intruding UHHH and *PEAR should be ignored. Traditional rewrite rules can handle this only by proliferating rules, to include ones that notice, in order to ignore, noise and irrelevancies. Eg. we would need rules like "NOUN UHHH == NOUN", "ADV UHHH == ADV", "VERB OBJECT == VERB".

Characterizers and Rewrite Rules

I suspect that thresholds can also be used fruitfully with rewrite rules, just as they are with perceptual characterizers. Linguists have been developing their grammars in ideal abstracted worlds, where an input contains one and only one sentence, a perfectly grammatical and noise-free string of perfectly spelled words. But just as it does not contain perfect and undeviating examples of object patterns, the real world does not contain perfect words and sentences. And human beings are capable of understanding and responding to incomplete, ungrammatical, and garbled sentences, and paragraphs of sentences, just as we can recognize distorted and fragmented objects. Threshold characterizers let a system infer implications and transformations from less-than-perfection, with the additional option that the strength of these implications can be a function of the degree of perfection (see Uhr, 1971).

Needs, Goals and Expectations

It is important to note that, since it must now handle multiply-implied actions, and decide among them, the system is capable of handling the effects of sources other than the external environment - of the system's INTERNAL needs, goals and expectations.

DECIDER-1 does just enough of this to indicate how it can be done. INTERNAL needs, goals and expectations are LOOKed AT along with the EXTERNAL INPUT. So they imply IDEAS for characterizing and acting. One or several needs, or a combination of needs and perceived objects, can trigger a transformation.

Flow of Processes

DECIDER-1 puts PRIMitives characterizers onto IDEAS, and inputs and puts the EXTERNAL environment and its INTERNAL need-states onto LOOKAT. It uses several DELimiterS to chop things out of LOOKAT; these things point to IMPLIEDS - transforms that go onto IDEAS and objects that go onto the SCENE (along with the things chopped out of LOOKAT).

When a characterizer succeeds, its IMPLIEDS that are of ITYPE I (which can be more characterizers to apply, parsing rewrite rules to apply, and/or actions to respond with) are merged onto IDEAS, ordered by weight, and those of ITYPE S (which are objects) are added to the SCENE. DECIDER-1 always pulls the most highly weighted TRANSform from its IDEAS list, to execute next. Therefore it starts executing primitive characterizers, and then will tend to execute characterizers that the successful primitives have implied. (Note that implications are merged together, so that when

the same one is implied by several different successful transforms its weight is a combination of the individual weights.)

After a while, possible actions will be implied. They will approach the beginning of the IDEAS list to the extent that their weights are high, or rise (as a function of being implied by several different transforms), and the more highly-weighted perceptual characterizers have already been taken off IDEAS and applied. Thus the IDEAS list serves to choose among characterizers, among characterizers and actions, and among actions.

Finally, an implied action will be chosen, because it is more highly implied than any other action, or for that matter than any other TRANSform on IDEAS. This triggers DECIDER-1 to go to execute that action, whether to Name, Describe, Find, Move, or Answer.

Thus the IDEAS list handles everything. The person (or the learning routine) that writes the memory lists must be careful to assign weights to the Implieds that will keep processes properly ordered. For example, characterizers to apply next, to gather more information in deciding whether a particular thing (whether object or word) has been found, should probably be given the highest weight. Next highest should be characterizers and rewrite rules that compound things into higher-level things (eg. objects, as when *EYE *EYE *NOSE *MOUTH imply *FACE; or phrases, as when ADJECTIVE NOUN implies NOUNPHRASE). Then come some of the fact-retrieval transforms, such as CHIMPS LIKE BANANAS Implies CHIMPS LIKE BANANAS, where weights of the parts of the description and the threshold for success are adjusted so that any two of the three parts are sufficient to succeed, giving the complete fact as the transform.

(Note that this is only one of a variety of ways in which facts can be handled. For example, the following three transforms are equivalent to the single one just given:

CHIMPS LIKE == BANANAS

LIKE BANANAS == CHIMPS

CHIMPS BANANAS == LIKE

with thresholds set so that all (two) parts must succeed. Another variant would have the implication refer back to the description, in effect saying, "output those parts of the description that were not found." But this makes necessary descriptions that are not redundant, whereas it is often very convenient for perceptual characterizers to contain a number of different parts that are in fact or-ed possibilities, with thresholds low enough so that the presence of only a small percentage of them will make the characterizers succeed. This suggests a more sophisticated routine to get a non-redundant covering of the parts of the description that do not succeed.)

An action is triggered when it is taken from IDEAS, because it is now the most highly implied transform. This means DECIDER-1 has chosen it, over any other possible transforms - including other actions or more perceptual characterizers. The specific actions will be discussed, with examples, following the program.

Figure 1. OVERVIEW and Program Code for DECIDER-1

```
(OVERVIEW DECIDER-1. Applies IDEAS to the SCENE till chooses an
(act, and acts.
INITialize DELimiterS and all transform in Memory                MD-MN
IN      Put PRIMitiveS on IDEAS,                                 M(N+1)
          erase SCENE, input EXTERNAL environment.               M(N+2),1
SENSE   LOOKAT EXTERNAL environment and INTERNAL needs           2-5
          by chopping out possible NAMEs using pairs of
          DELimiterS (DLEFT and DRIGHT)
        If NAME has IMPLIEDS and CLASSES, get its LOCa-          6-8
          tion and put that NAME and
          IMPLIEDS onto SCENE (and IDEAS)
TRANSFORM   Get the highest weighted TRANSform from              9
              IDEAS and go to its TYPE (or to Describe
              if nothing's left)
Perceive    Look for the CLASSes in the TRANSform's              10-17
              DESCRiption in SCENE or EXTERNAL (as
              specified by G or E), put all found on
              GOT, and TOTAL their WeighTs.
EVAL    If TOTAL reaches THRESHold, go to ITYPE (= I or S),      18-19
          to merge IMPLIEDS into IDEAS or SCENE.
          (Otherwise go back to TRANSFORM)

(The different transforms follow:

(I and S merge onto IDEAS and SCENE.
I       Merge this NAME's weight (TOTAL) with its WeighT         20-29
          on IDEAS (if there), and put it ORDERED by
          TOTAL weight back onto IDEAS. (go to IMPLY)
```

| | | |
|---|---|---|
| S | Get that object (NAME) from SCENE, and if close enough, combine weights and locations, or add it as a second, or a new, object. (go to IMPLY) | 30-35 |
| (N, D, and C talk about objects, Naming and Describing (them them. | | |
| Name | Get all things from SCENE belonging to the class 'OBJECT' and choose and output the one with the HIghest WeighT. | 36-45 |
| Describe | Get all OBJECTs (stored in TRANS) in SCENE, and output their NAMEs. | 46-49 |
| Class | Store the specified CLASS in TRANSform, and go to Describe (which will get and output only objects of that class). | 50 |
| (F and M act upon objects, Finding and Moving them. | | |
| Find | Find an OBJECT of the designated CLASS, bracket and output it. | 51-54 |
| Move | Find two OBJects of the classes designated, and Move the first to the second | 55-59 |
| (A accesses facts in memory, to Answer queries. | | |
| Answer | output the fact stored under the TRANSform that has been chosen | 60 |

```
(DECIDER-1. Applies IDEAs to SCENE till chooses an act.  DECIDER-1*
(Handles both surrounding edges (001...10) and a trailing
(space as DELimiters
INIT set DELS = '001]10]] ]'                                        MD*
     set INTERNAL = 'HUNGRY 4 THIRSTY 2 FIND-RICHES 2 '             MI
(Memory Transforms go here.  See examples                           M3-MN
IN   set  IDEAS = PRIMS                                             M(N+1)
          erase   SCENE                                             M(N+2)
          input EXTERNAL till ] [-to END]
(SENSE gets objects (NAME) between DELimiterS and merges
(their IMPLIEDS
SENSE       from DELS get LDEL ] RDEL ] = [-to TRANSFORM]            2
        erase LOC                                                   3
        set  LOOKAT  = EXTERNAL INTERNAL                            4
S1     from LOOKAT get LEFT that LDEL NAME till that RDEL =         5
+           RDEL  [-to SENSE]
       from $NAME get  ']I=' IMPLIEDS ']C=' CLASSES ] [- S1]        6
(SENSE adds NAME to SCENE and puts IMPLIEDS
(on IDEAS and SCENE
            LOC =  size(LEFT LDEL) + LOC                            7
        on SCENE list NAME CLASSES ] NAME 4 LOC ] [IMPLY]           8
(Get most highly weighted TRANSform from IDEAS. (Go to Describe
(if none left.)
TRANSFORM     from IDEAS get TRANS TYPE  TOTAL GOT ] =              9
+                  [+to  $TYPE. -to D]
P             COPYS = SCENE                                         10
       from $TRANS get 'D=' THRESH DESCR ']I=' IMPLIEDS ]           11
+         [-TRANSFORM]
P1     from DESCR get TP CLASS WT = [+ $TP -EVAL]                   12
E      from EXTERNAL get LEFT that CLASS [- P1]                     13
          on Got list size(LEFT)  CLASS CLASS [P2]                  14
(Get an OBJect of the specified CLASS (in order)
 G  from COPYS get LEFT # that CLASS # RIGHT ] OBJ WT               15
+           LOC ] = [-P1]
          on GOT list LOC CLASS OBJ                                 16
P2     TOTAL = TOTAL + WT [P1]                                      17
EVAL  is TOTAL lessthan THRESH? [+ TRANSFORM]                       18
(TOTAL has reached THRESHold. Merge IMPLIEDS into IDEAS
(and SCENE.
IMPLY from IMPLIEDS get ITYPE NAME TOTAL =                          19
+ [+$ITYPE. - SENSE]
```

*NOTE: The program attempts to explain itself. But for problems see the Appendix for a brief description of the program language used (EASEy-1), and Uhr, 1973a, for a primer.

```
(Put transforms onto IDEAS, with GOT, ordered by TOTAL weight)
I       erase ORDERED                                                    20
        from IDEAS get # that NAME # TYPE WT IGOT ] = [-I1]              21
        TOTAL = TOTAL + WT                                               22
        on Got set IGOT                                                  23
I1      from IDEAS get ORNAME ORTYPE ORTOTAL ORGOT ] = [-I3]             24
    is TOTAL lessthan ORTOTAL [-I2]                                      25
        on ORDERED list OROBJ ORTYPE ORTOTAL ORGOT ] [I1]                26
I2      on ORDERED list OBJ TYPE TOTAL GOT ] OROBJ ORTYPE                27
+           ORTOTAL ORHIST ] IDEAS [I4],
I3      on ORDERED list OBJ TYPE TOTAL GOT ]                             28
I4      IDEAS = ORDERED [IMPLY]                                          29
(Put objects on SCENE with CLASSES, weight,  and LOCation
S       from SCENE get  that NAME CLASSES ] NAME WT SLOC ] =             30
+           [+S4]
        from $NAME get ']C=' CLASSES ] [S2]                              31
S4      is ABS(LOC - SLOC) lessthan 4 ? yes - LOC = (SLOC + LOC)         32
+           / 2 [-S3]
    TOTAL = TOTAL + WT                                                   33
S2      on SCENE list  NAME CLASSES ] NAME TOTAL LOC ] [IMPLY]           34
(Add a 2d TRANSform object at a new location
S3  on SCENE list NAME CLASSES ] NAME WT SLOC ] NAME                     35
+           CLASSES ] NAME TOTAL LOC ] [IMPLY]
(Name the single OBJect with highest implied weight.
N       COPYS = SCENE                                          N1        36
N1  from COPYS get LEFT ' OBJECT' RIGHT ] HI ] = [- OUT2]      N2        37
        from HI get HIOBJ HIWT HILOC                           N3        38
N2      from COPYS get LEFT ' ]OBJECT' # RIGHT ]
+   OBJ WT LOC = [- OUT]                                       N4        39
    is   HIWT lessthan WT      [+ N2]                          N5        40
        list HI = OBJ WT LOC                                   N6        41
      from HI get HIOBJ HIWT HILOC [N2]                        N7        42
OUT  from  $HIOBJ get 'C=' NAME                                N8        43
      output  'IT IS A ' NAME [IN]                             N9        44
OUT2 output 'NOTHING RECOGNIZED' [IN]                          N10       45
```

```
(Describe all things on SCENE of class TRANS (= OBJECT) with
(WeighT at least = 20.
D     from SCENE get LEFT  # that TRANS # RIGHT ]                    D1  46
+        OBJ WT LOC ] = [- IN]
         is WT lessthan 20 ? [- D]                                   D2  47
         from $OBJ get 'C=' NAME                                     D3  48
         output  'THERE IS A ' NAME ' AT ' LOC [D]                   D4  49
(To Describe giving only objects of the specified
C     from GOT get CLASS TRANS [D]                                   C1  50
(Find an OBJECT of the designated CLASS; bracket  it in
(EXTERNAL; output EXTERNAL.
F        from GOT get 'NAME ' CLASS                                  F1  51
         from SCENE get ] that CLASS # ']OBJECT ' RIGHT :            F2  52
+          OBJECT
         from EXTERNAL get that OBJECT = [ OBJECT ]                  F3  53
         output EXTERNAL  [IN]                                       F4  54
(Move OBJectA to OBJectB
M     from  GOT get LOCT ACTION LOCA OBJA LOCB OBJB                  MO1 55
      from SCENE get ] that OBJA # LEFT ']OBJECT ' RIGHT :           MO2 56
        OBJSA WTA  LOCA ] =
      from SCENE get ] that OBJB # LEFTB ']OBJECT ' RIGHTB :         MO3 57
        OBJSB WTB LOCB = ] OBJA  LEFT ']OBJECT ' RIGHT
+       OBJSA WTA LOCB ]
            OBJB LEFTB ']OBJECT ' RIGHTB : OBJSB WTB LOCB
          output OBJSA 'IS NOW AT LOC = ' LOCB ' AS IS '            MO4 58
+            OBJSB
(Output entire SCENE (should clean out the internal information)
         output 'ENTIRE SCENE IS  '  SCENE [IN]                    MO5 59

(Answer a query with a fact stored in Memory and IMPLIED by the Input.
A     output $TRANS  [IN]                                            A1   60
END                                                                       -
*PEAR ROBOT  PUT *BANANA  PEAR IN PAIL *BOX *PAIL *APPLE ]            I1
```

## SOME EXAMPLES OF DECIDER-1'S BEHAVIOR: DECIDING TO NAME

Let's look now at some simple examples of inputs DECIDER-1 can handle, along with the particular characterizers and transforms it would use. (Normally these would be part of a much larger set, designed or learned to handle the whole range of things and problems the system must cope with. This means that several actions will often be multiply implied, the most highly implied being chosen when it reaches the head of the IDEAS list.)

### A Minimal Memory to Recognize That it Should Name

DECIDER-1 does not have built into it that it should name, or do any particular thing. So to get it to name the perceived object we should give it a data Input card like:

NAME THIS *APPLE ] I-1

(*APPLE would really be a picture of an apple, with all its parts and qualities, and all the particular ideosyncracies of that particular apple. But to keep things simple we use *APPLE, much like a whole-template, to signify the sensed apple.)

This is in contrast with the typical pattern recognition program that has built into it that it should name, to which we would give an input like:

*APPLE ] I-2

The following statements in memory would handle I-1. (these statements do pattern characterizing, linguistic parsing, and triggering of actions. They show only those parts of the memory network that are needed for and pertinent to this problem.)

Name Description Implieds Classes

*APPLE = D=]I=I *APPLE-TREE P 9 ]C=APPLE 7 FRUIT 6 OBJECT 8 ]' M-1

NAME = ']I=I P2 P 5 ]C=] M-2

P2 = 'D=9 G NAME 6 G THIS 3 ]I=I NAME N 37 ]'[1]

Note how *APPLE implies a Perceptual transform that will be put onto IDEAS to look for the Description of an *APPLE-TREE. This indicates how parts and qualities can point up to larger wholes. We will not pursue *APPLE-TREEs here; but we will soon see examples using this upward flow of processing.

DECIDER-1 will use the Right space DELimiter to get the three strings NAME, THIS, and *APPLE from the Input data card I-1. NAME (M-2) will put the Perceptual transform P2 on IDEAS (with weight 5). *APPLE, with the CLASSES OBJECT and FRUIT and weight = 9 is put onto SCENE (7,8,30), as are NAME and THIS.

P2 will then be taken from IDEAS and used to characterize the SCENE. It will find NAME and THIS, adding weights of 6 and 3 to the TOTAL weight of P2 on IDEAS (TOTAL = 5, since only NAME implied P2). This gives a grant TOTAL of 14, which exceeds the THRESHold of 9, so that P2 succeeds, firing the Implied transform that switches the progam to Name (which means choosing the most highly implied thing on SCENE that belongs to the OBJECT class, getting its first Class, which is its NAME).

---

[1] Implied transforms are given high weights in these examples, so that they will be chosen quickly. In actual practice these weights should be low, so more characterizing is done before acting.

## Combining Multiple Implications

Note how THIS adds to the weight of P2. Thus 'NAME THIS' triggers the program with more assurance than does just 'NAME'. (Because it is kept very simple, P2 will succeed whether 'THIS' is found or not; but a more reasonable DESCRiption would contain many more parts, giving many alternate possibilities, in which some combination, and not just NAME, would be needed to succeed - possibly a second characterizer would succeed from just NAME, but would not imply the decision to choose and output with such high weight, so that it would only make for a slight tendency that would override only for want of anything stronger.)

A real apple would be perceived in two dimensions, and would have no delimiting spaces surrounding it. DECIDER-1 could easily be extended to handle two dimensions (rows would have to be stored and used along with columns). Even without that, DECIDER-1 can handle actual pictures with all rows stretched out in one dimension, since DELimiterS like "001" and "10 will chop out edges and portions of shapes, and the DESCRiption parts that Extract from the raw EXTERNAL environment can specify any strings.

## Toward More Realistic Decisions About Complex Environments

Of course naming is a trivial problem until there is some possibility of confusion among objects. The classic pattern recognition problem arises when there are several possible object-names, and recognition is difficult, because objects with the same name vary in strange and unknown ways, and because there is no

simple partitioning between objects with different names. The following statements show how this is handled. M-4 characterizes *PEAR in the same way that M-1 characterized *APPLE, as a whole-template. M-5 and M-6 (which M-7 puts on the PRIMitiveS with which statement 2 initializes IDEAS) characterize an apple and a pear by using descriptions of some of their parts and qualities.

```
      *PEAR = 'D=]I=I *PEAR-TREE P 7 ]C=PEAR 9 FRUIT 5 OBJECT 7 ]' M-4
      M-5 = 'D=7 G *STEM 4 G *PEAR-SHAPE 3 E 00111 3 '           M-5
+       'E 011011 4 G *GOLD 3    ]I=S *PEAR X 9 S *APPLE X 3 ]'
      M-6 = 'D=6 G *APPLE-SHAPE 5 E 0111110 3 G *RED 4 '         M-6
+       ' ]I=S *APPLE X 12]'
      PRIMS = 'M-5 P 27 X ]M-6 P 23 X  ]'
```

A full-blown program would have a large number of characterizers of this sort (typically, 20 to 100, but occasionally several thousand characterizers are used to handle 10 or 20 different pattern sets, where each characterizer can, like M.5, imply several different names.) And each characterizer can have many more parts, so that its threshold can be exceeded by the combined weight of a number of different combinations of some of them.

DECIDER-1 can now begin to handle inputs like:

```
NAME THIS *STEM *PEAR-SHAPE *GOLD *RED ]          (or)      I-3
NAME 0011100,0111110,1111111,0110110, *STEM *GOLD ]         I-4
```

(where a 6-column by 4-row matrix of black and white has been encoded into a string of 1's and 0's, with a comma ending each row).

M-5 and M-6 as shown imply only objects to be added to the SCENE. But they could also imply transforms to be merged into

IDEAS. This means that the information that has been uncovered so far can direct and focus the system's attention (see Uhr, 1973c, 1973b, chapter 8). Only the characterizers on PRIMitiveS are always put on IDEAS. (Note that they may not always be used, for if any action is implied with a higher combined weight DECIDER-1 will choose it first - but this depends upon the relative weights that the programmer, or the program's learning routines, have assigned to primitives as opposed to implied actions).

## "Understanding" and Responding to Variant Commands, and Suggestions

We can easily allow DECIDER-1 to handle a number of variant command forms, all meaning the same thing - "NAME!". The following statements show how this can be done:

```
WHAT = 'D=]I=I P3 P 4 ]'                                          M-8
THIS = 'D=]I=I P3 P 3 ]C=LOC 5 ]'                                 M-9
HERE = 'D=]I=I P3 P 3 ]C=LOC 4 ]'                                 M-10
P3   = 'D=12 G WHAT 4 G LOC 4 G IS 2 G SEE 2 G DO 2 '             M-11
+        ']I=I NAME N 43 ]'
```

We can now input, eg:

```
WHAT IS THIS *APPLE *PEAR ]                                       I-5
WHAT DO YOU SEE HERE *STEM *APPLE-SHAPE *PEAR-COLOR ]             I-6
```

and a number of other variants. Depending upon the number of characterizers for such parts and qualities as *STEM and *PEAR-COLOR, the system will accrue more or less information for the actual pattern naming decision.

Note how P3 and P2 (M-11 and M-3) both imply the transform act of Naming. And this THIS and HERE both belong to the LOCation class, P3's Description can refer to LOC, which will succeed if either THIS or HERE is in the input.

External Objects and Internal Needs can also Imply Commands

It is not essential that the command be made explicit by verbiage like "WHAT IS THIS". We can have objects like *APPLE and *PEAR implicitly "command" actions, like NAME, as follows:

*APPLE = ']I=I NAME N 1 S *APPLE X 9 ]C=APPLE 7 FRUIT 6 ' M-1.V

+ 'OBJECT 8 ]'

New - Implies the act of Naming, with a low weight.

Any explicit command in the input, eg. DESCRIBE or FIND will, because it is assigned a higher weight, override such an implied Name command; but when no such explicit command appears DECIDER-1 will name by default. (Note that, for variety, statement 9 defaults to Describe, if absolutely nothing is left on IDEAS.)

DECIDER-1's BEHAVIOR: DESCRIBING

Up to this point we have examined variants on Naming. The following addition allows DECIDER-1 to Describe or Name, so that for the first time it really chooses among alternate actions:

DESCRIBE = 'D=]I=I OBJECT D 59 ]' M-12

Now an Input like:

*BOX DESCRIBE THIS *APPLE *PEAR *APPLE ] I-7

triggers the Describe routine (statements 46-49). This gives a very stylized description: all object names with TOTAL weights above the pre-set CHOOSE level are output. We might want to augment

P3 (M-11) as follows, so that an Input like WHAT IS HERE implies both Naming of a single object and Describing of all objects. (Once again, this makes sense only when many more characterizers are used, including ones whose implications and weights have been learned from past experience.) Now Name and Describe will be multiply implied, and whatever is first taken from IDEAS will prevail. Given:

P3 = 'D=12 GWHAT 4 G LOC 4 G IS 2 G DO 2 ]I=I OBJECT D 46 ' M11.V
+ 'I NAME N 43 ]'

with the weights shown, 46 for Describe will override 43 for Name - except if one of the other transforms also implies Name. (Note that anything can Imply anything, if we wish, including Describe and Name.)

We can allow the user variety in the kind of description he requests, e.g.:

DESCRIBE FRUIT *APPLE *BOX *CRACKER *PEAR ] I-8

by adding transforms that will lead to descriptions containing only members of the <u>class</u> of objects designated. (If the "class" is an object, like *APPLE, it will output all instances and their locations.) With:

DESCRIBE = 'D=]I=I OBJECT D 59 I P4 P 63 ]' M-12.V
P4 = 'D=73 G OCLASS 6 G DESCRIBE 7 ]I=I CLASS C 61 ]' M-13
FRUIT = ']C=OCLASS 4 ]' M-14

any word in the Input that has (as in M-14) Object CLASS as one of its classes will trigger the Class describing transform first (with a weight of 61). <u>DECIDER-1</u> will then go to C (statement 50) where

the designated class (FRUIT) is put into TRANSform, replacing OBJECT (which is the most general class). Then it goes to D, to Describe, giving only members of that class.

## FURTHER EXAMPLES OF DECIDER-1'S BEHAVIOR: ACTING UPON THE SCENE

DECIDER-1 can perform several actions that do not merely talk about the scene, by naming or describing, but actually change it.

### Finding Objects

Probably the simplest action upon an environment is to find and touch an object, as in response to an Input like:

*APPLE TOUCH THE *BOX APPLE *PEAR *PAIL ] I-9

DECIDER-1 shows it is touching an object by placing a bracket around it. To do this, we need new characterizers to understand commands like "FIND" or "TOUCH":

FIND = ']I=I P5 P 59 ]' M-15

TOUCH = ']I=I P5 P 61 ]' M-16

P5 = 'D=67 G NAME 9 ]I=I X F 65 ]' M-17

APPLE = ']I=I *APPLE P 7 ]C=NAME 9 NFRUIT 5 OBJNAME 6 ] M-18

P5 is implied with a high weight by the words FIND or TOUCH. (We would expect many more characterizers that implied P5 with a full memory.) P5 looks for NAME, which will be among the classes of many naming words (e.g., APPLE in M-18). Thus when both a "find" word and a NAMEing word are found, P5 fires, Impling

the Find transform with the high weight of 65. Then Find (51-54) gets the particular NAME, looks on SCENE for an OBJECT of that NAME, then looks in the raw EXTERNAL view for that object, and outputs EXTERNAL with the object in brackets.

Note how the name word APPLE also Implies the Perceptual transform *APPLE, to look for an *APPLE. Actually, it should imply all characterizers that imply *APPLE (these would include M-6 and M-5, and still others). Better, it would imply an internal node that represented the concept of apple, where back-links would point to the characterizers that implied apple, and these would be added to IDEAS. Thus the word would focus the system to start looking for the thing.

Memory transform nodes M-1, for the object *APPLE, and M-18, for the word APPLE, handle things in slightly different ways, because they have been tailor-made for their specific processes. It is instructive to compare them, and to consider generalizing them, and the program that interprets them, so that words and objects, and the links between them, are handled in exactly the same way.

## Manipulating Objects

Let's add the Memory items needed to handle Inputs like:

*APPLE SARAH PUT APPLE PAIL *BOX *PEAR *PAIL ] (or) I-10

*BOX3 ROBOT PUSH BOX2 NEXT BOX1 *APPLE *BOX1 *PEAR *BOX2 I-11

We'll do this in the simplest possible way, holding in abeyance a number of interesting problems. Let's ignore the issue of the actor, as though DECIDER-1 assumes it should always act. (Very

interesting issues arise when the system must decide for whom a command or other statement is intended.) The Parsing transforms will be very simple, so that DECIDER-1 recognizes only very stylized commands. That's o.k., since more Memory statements would give sophisticated parsing. But the Move routine assumes it gets information in the order: ACTION OBJectA OBJectB. More code would be needed to extract this information from more complex commands. But note how junk words, like THE and AN, will be ignored.

The specified ACTION is ignored, DECIDER-1 assuming it is to Move OBJectA to OBJectB. More code is needed to interpret and effect different actions, and to handle different relations (which are also ignored). In fact the whole Move routine is quite rudimentary and unsatisfactory; its purpose is merely to show how this sort of thing can be triggered. (Note, though, how similar it is to other response routines, e.g., Find and Describe.)

To contrast with Find, the Move routine changes the SCENE rather than the raw EXTERNAL input, and outputs the new LOCation of the move OBJect. What is really needed is a routine that cleans up the internal information on SCENE, and then outputs the objects, and higher-level objects, in their new locations.

DECIDER-1 needs the following in Memory to handle Inputs like I-10 and I-11:

```
PUSH = 'D=]I=I P6 P 34 ]'                                          M-19
PUT = 'D=]I=I P6 P 33 ]'                                           M-20
P6 = 'D=69 G OBJNAME 37 G VESSEL 37 ]I=I MOVE1 M 65 ]'             M-21
BOX = ']I=I *BOX P 6 ]C=NAME 9 VESSEL 7 ]'                         M-22
BOX2 = ']I=I *BOX2 P 9 ]C=BOX 12 NAME 6 VESSEL 4 ]'                M-23
```

Similar lists are needed for other things, like PAIL, BOX1, PLACE, *BOX1. But note that actor words like SARAH and ROBOT, and relation words, like NEXT, are ignored. (A better program would handle them.)

P6 will fire the transform to Move, with the high weight of 63, only if it has been fired by either PUSH or PUT, and then its Description finds both an OBJect NAME and a VESSEL. We could add more memory to have other lower-level things, like VESSEL, imply P6. But we need more code to handle:

*BOX ROBOT PUSH BOX TO BOX *PAIL *BOX ] I-12

For DECIDER-1 must note that it has bound an object (the first *BOX) as OBJectA of the command, and not bind it again as OBJectB.

Internal Needs Can Imply Transforms to Characterize or Act

Still another impetus to action can be handled in the same way - the system's INTERNAL needs and goals (MI). A need (eg hunger) cam imply a characterizer like M-5 or M-6 that looks for food (*APPLE and *PEAR). It can also imply "EAT an object that belongs to the FOOD class." Similarly, an object like *APPLE can imply the action EAT as well as implying the word APPLE and a higher-level construct like *APPLE-TREE. Once again, all these implications, no matter from what source, are combined, and then chosen among.

DECIDER-1 has HUNGER, THIRST, and FIND-RICHES on INTERNAL (MI). Each should name a memory statement pointing to a perceptual transform, like FOOD, WATER, GOLD, and FIND.

## Building and Using Sequences of Actions

More actions must be coded, to grasp, eat, drink, and covet, and to go to. But note how similar are the actions *SARAH GOTO *BANANA and *BANANA INTO *BOX. (Note that these are internal statements driving the actions, not the word commands - e.g., SARAH GOTO BANANA.) The former moves the self (as though OBJA) to OBJectB. In fact a proper encoding of an act like *SARAH PUT *BANANA *BOX would have *SARAH GOTO *BANANA; *SARAH GRASP *BANANA; *SARAH GOTO *BOX (with the grasped banana): *SARAH UNGRASP *BANANA.

Now an action is a whole sequence of sub-actions which must be effected step by step, over time, at the same time the system perceives and monitors new EXTERNAL environments. And recoding, to get a set of basic actions, will make apparent the similarity between many. E.G., FIND *APPLE becomes MOVE OBJectA (= *SELF) to OBJectB (= *APPLE) (or MOVE *PAW TO *APPLE). And EAT *APPLE becomes MOVE *PAW *APPLE; GRASP *PAW *APPLE; *MOVE *PAW *MOUTH; etc.

## ANSWERING QUERIES WITH FACTS STORED IN PERMANENT MEMORY

Up to now DECIDER-1 has Named, Described, Found, and Moved parts of the scene input to it from the external world. We will end with an action that follows network links into its memory for facts, in order to retrieve answers that have been implied by input queries.

This is of special interest, for it demonstrates how this kind of fact retrieval can be done using the same Perceptual transform mechanism used for characterizing and for parsing. A fact is IMPLIED by enough elements of its DESCRiption to exceed THRESH-old, where question-words are parts of this description, and have high enough weights to preponderate.

To handle an Input like:

| | | |
|---|---|---|
| WHO LIKE BANANAS ] | (or) | I-12 |
| A CHIMP LIKES WHAT ] | | I-13 |

DECIDER-1 must be given fact transform nodes of the following sort:

```
BANANAS = ']I=I MEM1 P 33 ]C=NAME 4 FRUIT 3 OBJNAME 2 ]'   M-24
CHIMP = ']I=I MEM1 P 32 ]C=NAME 3 ANIMAL 7 ]'              M-25
MEM1 = 'D=93 G QWORD 45 G LIKES 17 ]I=I FACT1 A 77 ]'      M-26
WHO = ']C=QWORD 3 ]'                                       M-27
WHAT = ']C=QWORD 3 ]'                                      M-28
FACT1 = 'A CHIMP LIKES BANANAS'                            M-29
```

Now the specified parts (BANANAS, CHIMP) of the fact imply the MEM1 node that looks for the other parts (LIKES) and the Question WORDs (WHO and WHAT). If MEM1 succeeds it triggers the Answer transform, which puts FACT1 in TRANSform, FACT1's contents ("A CHIMP LIKES BANANAS") is immediately output (statement 60).

The reader might try his hand at alternate memories for this fact, and also try to add other memory nodes, for this and for other purposes.

## DISCUSSION: FAULTS AND POSSIBLE IMPROVEMENTS

A number of functions are weak, but many can realtively easily be improved.

### Locating Objects

DECIDER-1 can easily be coded to find the closest instances of the DESCRiption OBJects it is looking for in the SCENE, and use the discrepancy in distance as a negative factor when computing how highly that TRANSform is implied. It could further infer the location of the object, by having stored with each part the relative location of the whole object with respect to that part, computing an overall average relative location, and then using this to project where the IMPLIEDS should be (if objects) or should be applied (if transforms).

Better decisions can be made as to whether two implications of the same object in slightly different locations mean a single, or two different but identical, object(s) is/are there. The system can compute a "near" parameter, as a function of the total size of the scene and/or the average size of objects, and use this to make the decision.

### Ordering and Concatenation

Characterizers' DESCRiptions can be of ordered objects, but not of objects that actually touch, that is, are concatenated. I suspect that simple ordering is preferable in the real world, since it allows for interspersed things - grunts, noise, objects, etc.

And when we generalize to two dimensions (for this, see Uhr, 1971) we can no longer use concatenation anyway. But in any case it would be quite easy to have the system check the relative locations of objects, when concatenation was asked for. Better would be to have the system check relative locations in all cases, so that distances could be asked for (concatenation now merely becoming a distance of zero), and have the characterizer succeed to the extent it is near the projected location.

## Merging Implieds into the Scene and into the Ideas List

DECIDER-1 keeps one list, SCENE (of objects) and a second list of IDEAS (TRANSforms of all sorts - to characterize, to parse, or to act). These two lists are quite similar, and it would be possible to code a variant program that handled them in exactly the same way. The system would loop through IDEAS, and all implied things would be merged onto a NEWIDEAS list and, after an entire pass through IDEAS, NEWIDEAS would be added to the end of the SCENE, and IDEAS would be set to contain NEWIDEAS (which would be erased), and the program would loop back ghrough IDEAS.

To do this, the formats for SCENE and IDEAS would have to be generalized. In DECIDER-1 SCENE contains the object's CLASSES (and also the object listed among the classes, so that the system can treat any designations of the object as though a class containing itself);it also contains the object's LOCation (actually, an approximation averaging the LOCations of the last parts of the DESCRiptions found). But IDEAS contains a list of each CLASS of the DESCRiption found, along with the specific OBJects that were the

found members of these classes, and their LOCations in SCENE; but it does not contain the CLASSES of the object. This is simply a result of tailoring each list to contain just those pieces of information that will be needed. But we could have each list contain all the information, with a certain amount of wasted effort and space. This would also mean that objects would be put onto IDEAS, and TRANSforms would be put on SCENE - wastes of effort if they are not used. But they may well be. For example, Answering queries could be improved if GOTs were stored with fact-things on SCENE.

## Parallel-Serial Processing, Using Several Lists

DECIDER-1 merges all IMPLIEDS transforms onto the single IDEAS lists, and takes the single most highly weighted implied TRANSform from IDEAS to apply next.

A more realistic procedure, at least for the perceptual processes, would apply a whole set of characterizer TRANSforms in parallel. The eye applies a number of different characterizers in parallel, where each characterizer is iterated throughout the eye's retinal array. The transforms from the retina are then similarly characterized by another parallel iterated set of characterizers, and this goes on for a series of transformations of unknown length, but roughly on the order of from 5 to 15 layers. A more complete model should almost certainly have such a structure, which will exhibit many more of the phenomena of sensory pre-processing, perception, chunking, and concept formation, and heirarchical compunding, as it successively transforms from layer to layer. (See, for example, Uhr, 1972, on "recognition cone" models.)

Similarly, the separate sense modalities, such as hearing and touch, first must apply sets of transforms to funnel information into the system. And the system's internal needs must be characterized and recognized; there is no built-in neon sign that flashes an a priori understood signal like "I AM HUNGRY".

## Coordinating Perceiving and Acting

DECIDER-1 keeps looping through the TRANSforms on IDEAS until it applies an action transform, which branches it away from Perception and into the designated action. The action is then executed, with a simple rather ad hoc routine, its results are OUTPUT, and DECIDER-1 INputs the next experience.

This is overly simple, albeit sufficient to exhibit the central point of this paper - that perception, linguistic parsing, fact retrieval, and at least parts of actions (the location of objects) can all be handled simultaneously, using the same simple mechanisms. But a more sophisticated system would have to spend some time and effort deducing the correct action sequence. And it should be able to branch back to search and perceive some more, if the objects it must act upon have not yet been found. If the system is interacting with an environment that changes over time (Uhr, 1973c) then the needed objects may not yet have come into view, and the system must put them, as expecteds, onto IDEAS, or whatever list controls its perceptual processes.

Now the system is confronted with the decision as to whether to continue executing the steps - deducing them, searching and waiting for the objects to which they apply - of a chosen action

sequence, or perceiving, or, possibly, switching into a new and different action sequence that is now more highly implied. A straightforward way to handle this is to separate the IDEAS list into several lists, call them LOOKFOR (which contains perceptual characterizers, linguistic rewrite rules, and transforms that search into the memory network) and ACTIONS (which are the transforms that rigger responses), and sort IMPLIEDS onto either LOOKFOR or ACTIONS. Now there are several options: The system can cycle only through LOOKFOR, until some decision rule is satisfied - eg. all, or a set number of characterizers have been applied, or something on ACTIONS has achieved a sufficiently high total weight. But the program needn't switch entirely into an action mode; rather, it can continue to apply characterizers from LOOKFOR interleaved with ACTIONS. This lets the action-sequence that it is trying to generate put things onto LOOKFOR to be applied, in order to help carry out the actions. It also lets the system keep monitoring its environment, for feedback that its actions are leading to expected consequences, and in case interesting, or dangerous, new things appear (this is something that is done by, and is probably necessary for the survival of, all living systems).

As a further consequence, new IMPLIEDS will be merged into ACTIONS, which raises the possibility that the action-sequence presently being executed may no longer be the most highly implied. So the program can re-decide, whether to continue or to start on some new track. (It seems reasonable to give the action presently being executed a certain amount of preference, eg. by adding an "inertial" constant to its total weight when comparing it with other possible

actions. Otherwise actions will be helter-kelter, dropped when half-finished.

Up to now I have assumed that only one action-sequence will be executed at a time. This is the simplest assumption and, indeed, living systems seem far more limited on the response side (at least in their consciously controlled actions) than on the perceptual side, where there is much parallel processing. But we are now in a position to have the system choose to work on several responses, either separately, or trying to deduce combined action-sequences that attempt to maximize returns while minimizing effort.

## REFERENCES

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

Brice, C. R. and Fennema, C. L. Scene analysis using regions. Artificial Intelligence, 1970, 1, 205-226.

Feldman, J. A. et al., The Stanford hand-eye project. Proc. 1st Joint Int. Conf. on Artificial Intelligence, Washington, D. C., 1969, 521-526.

Fikes, R. E. and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. Proc. 2d Joint Int. Conf. on Artificial Intell., London, 1971, 608-620.

Gardner, Beatrice T. and Gardner, R. A. Two-way communication with an infant chimpanzee. In: Behavior of Nonhuman Primates, Vol. 4 (A. Schrier and F. Stollnitz, Eds.) New York: Academic Press, 1971, 117-184.

Gardner, R. A. and Gardner, Beatrice T. Teaching sign language to a chimpanzee, Science, 1969, 165, 664-672.

Jordan, Sara R. Learning to Use Contextual Patterns in Language Processing. Unpubl. Ph.D., Diss., University of Wisconsin, Madison, 1971.

Munson, J. H. Experiments in the recognition of hand-printed text. AFIPS Proc., 1968, 33, 1125-1138.

Nilsson, N. J. A mobile automaton: an application of artificial intelligence techniques. Proc. 1st Joint Int. Conf. on Artificial Intelligence, Washington, D. C., 1969, 509-520.

Premack, D. The education of Sarah. Psychology Today, 1970, September, 54-58.

Premack, D. Language in Chimpanzee, Science,

Raphael, B. Programming a robot. Proc. IFIP Congress 68. Edinburgh, 1968, H135-H140.

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

Shapiro, S. C. and Woodmansee, G. H. A net structure based relational question answerer. Proc. 1st Joint Int. Conf. on Artificial Intell. Washington D. C., 1969, 325-346.

Simmons, R. F. Answering English questions by computer: a survey, Comm. Assoc. Comput. Mach., 1965, 8, 53-70.

Simmons, R. F. Natural language question-answering systems: 1969, Comm. Assoc. Comput. Mach., 1970, 13, 15-30.

Uhr, L. Flexible linguistic pattern recognition. Pattern Recognition, 1971, 3, 363-384.

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

Uhr, L. and Vossler, C. A pattern recognition program that generates, evaluates and adjusts its own operators. AFIPS WJCC Conf. Proc., 1961, 19, 555-569.

Uhr, L. and Kochen, M. MIKROKOSMs and robots. Proc. 1st Joint Int. Conf. on Artificial Intelligence, Washington, D. C., 1969, 541-556.

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

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

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

Winograd, T. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Unpubl. Ph.D. Diss., MIT, 1971.

Zobrist, A. L. The organization of extracted features for pattern recognition. Pattern Recognition, 1971, 3, 23-30.

Appendix: A Note on Programs (See Uhr, 1973a for details)

1. Numbering at the right identifies statements, and allows for comparisons between programs. M indicates initializing Memory statements: I indicates cards that are Input by the program. .V indicates a Varient, .1 an additional statement.

2. A program consists of a sequence of statements, and END card, and any data cards for input. (Statements that start with a parenthesis are comments, and are ignored.) Statement labels start at the left; gotos are at the right, within brackets (+ means branch on success; - on failure; otherwise it is an unconditional branch). + signifies a continuation card.

3. Strings in capitals are programmer-defined. Strings in underlined lower-case are system commands that must be present (they would be keypunched in caps to run the program). These include input, output, erase, set, list, get, start, call, that, and the inequalities. Other lower-case strings merely serve to help make the program understandable; they could be eliminated.

4. EASEy automatically treats a space following a string as though it were a delimiter; it thus automatically extracts a sequence of strings and treat them as names. The end-bracket ] acts similarly as a delimiter, but the programmer must specify it. The symbol # is used to stand for any delimiter (a space, ] or # ).

5. The symbol $stringI is used to indicate "get the contents of string I, and treat it as a name and get its contents" (as in SNOBOL).

6. Pattern-matching statements work just like SNOBOL statements: there are a) a name, b) a sequence of objects to be found in the named string in the order specified, c) the equal sign (meaning replace), and d) a replacement sequence of objects (b, c, and/or d can be absent). that string I means "get that particular object" - otherwise a new string is defined as the contents of stringI, which is taken to be a variable name.

7. size( ...) is a built-in function that counts the symbols in the string(s) named within parentheses. ABS( ... ) is a programmer-coded function to compute the ABSolute value of the expression within parentheses. (I don't bother to show the code.)