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

Received: December 17, 1974

A WHOLISTIC COGNITIVE SYSTEM (SEER-2) FOR INTEGRATED PERCEPTION, ACTION AND THOUGHT

by

Leonard Uhr

Computer Sciences Technical Report #234

December 1974

Presented as an invited paper at the Milwaukee Symposium on Automatic Computation and Control, April 1975. 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.

A WHOLISTIC COGNITIVE SYSTEM (SEER-2)

FOR INTEGRATED PERCEPTION, ACTION AND THOUGHT

Leonard Uhr
Computer Science Department
University of Wisconsin

Abstract

This paper describes a computer-programmed model, SEER-2, that can perform a variety of different types of actions that span the human intellectual processes, including naming objects, describing scenes, answering simple questions, deducing solutions to simple problems, pointing to objects, and manipulating objects. Recognition, description, memory search, and deduction are all handled in the same semantic cognitive network, using the same basic type of general transformation. Thus they are all intermingled, and can conveniently interact and call upon one another, as needed.

SEER-2's acts, both external and internal, are a function of a variety of influences, including the presses of internal needs and of external objects, as well as verbal commands (that is, the understood import of verbal utterances).

SEER-2 is thus a first step toward an integrated wholistic cognitive system, one that is as simple and general as possible yet begins to handle a representative variety of cognitive processes.

The SEER system uses a set of transforms that compose its memory network, and contain the specific information needed to handle the types of problems it will encounter - much as a parser must be given a set of rewrite rules that contain the needed information about the vocabulary and grammar of the languages to be parsed. In future systems we will attempt to have these memory transforms learned through experience. But for now they must be given the system. Examples of SEER-2's behavior when given a small memory are presented.

1. INTRODUCTION

This paper describes a computer-programmed model (coded in EASEy [21, 27], an English-like variant of SNOBOL4 [8]) that handles a wide variety of cognitive tasks, including pattern recognition, scene description, syntactic parsing, semantic "understanding," retrieval of information, problem-solving, and action generation, in a relatively simple, homogenous and integrated way.

The system (called SEER-2, for SEmantic learnER)* is able to survey its external environment, recognize and describe to itself what it perceives there, and then respond appropriately, as a function of deductive inferences, and access of pertinent information stored in its memory. It must be given a memory network, much as a parser must be given rewrite rules. Examples that demonstrate some of SEER-2's capabilities, using a small memory network, are given.

SEER-2 handles several rather difficult and interesting new problems.

A) It perceives objects and parses and understands verbal utterances using the same processes, applied to the same input channel. This is deemed necessary, since objects and word-utterances must be recognized and understood even when (as they always are in the real world) they are mixed together in the same sensory input channel.

B) It decides among a variety of different possible acts, both internal and external. This is necessary because it cannot be programmed in advance to search for information, or to problem-solve, or to describe. Rather, it must recognize that is has been told to do one of these things, or that one of these things is needed to achieve its goals.

C) It decides what external acts to effect as a function not only of external commands, but also of internal needs and goals, and of the press of perceived external objects. This means that a variety of verbal utterances can be input and their import recognized, including suggestions and comments, as well as commands. And the system decides whether to obey them, or whether to respond to other pressures, including its own internal motivating forces.

2. BACKGROUND AND HISTORY OF RELATED RESEARCH

A number of projects have attempted to combine several cognitive functions into a single system (for reviews, see Ernst [4] and Uhr and Kochen [29].

2.1 ROBOT SYSTEMS EMBODIED IN HARDWARE

By far the largest amount of effort, in terms of men, money and equipment, has been expended on the building and programming of hardware "robots" that move around physical space under the control of computer programs. This work has been concentrated on three large ARPA-sponsored projects - a mobile robot at the Stanford Research Institute (Raphael [14] Nilsson [12]) and

*or Sensed Environment Encoder, Recognizer and Responder, in 2-dimensional environments; or SEE and ERR.

two hand-eye projects, at Stanford University (Feldman et al. [6]) and at MIT (Winston [3]). The Edinburgh University project [2] and several newer projects, at the Jet Propulsion Lab, and in Japan [5] and Russia, follow along their lines. See Ernst [3] for an interesting precursor.

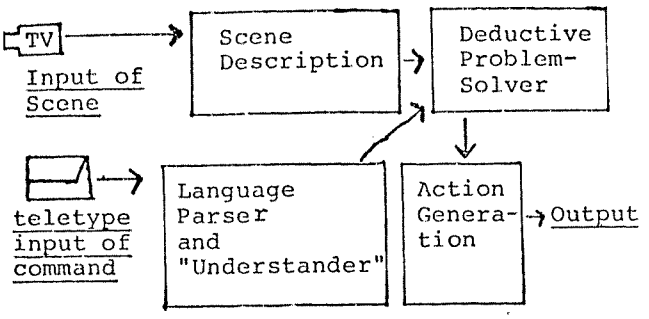
2.2 ORGANIZATION OF ROBOTS INTO SEPARATED SUB-SYSTEMS

Almost all of these projects use 1) a television camera for visual input that is then processed by programs for feature extraction and scene analysis, 2) a keyboard for typed input of verbal commands, that are processed by syntactic and semantic analysis programs, 3) a system for deductive inference that finds a path from the present state of the robot's environment, as got from the scene analysis, to the desired state, as specified in the command; and 4) programs that bind and then execute specific actions to the steps of the solution path.

Thus 1) scene analysis, 2) semantic analysis, 3) problem-solving, and 4) acting are handled by separate subsystems, with relatively little interaction between them (See Figure 1).
The perceptual system will take a television picture and try to recognize and analyze all the objects in the scene (typically, some cubes, wedges, and/or pyramids), and build up a floor-plan of where they are located in the room or, for the hand-eye projects, on a table. The language system will parse and then make a semantic analysis of the teletyped command, and then try to associate the command with objects in the floor-plan, and also with assertions about objects and the ways

Figure 1. Brains, Psychology and Robots

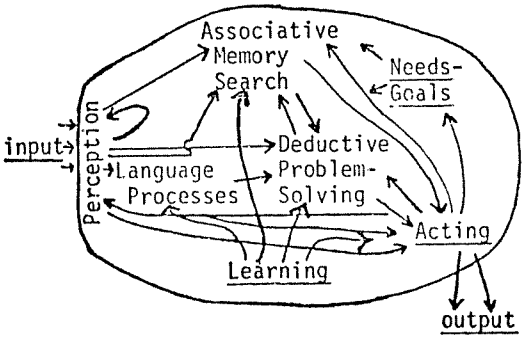
a) The Organization of Subsystems in Robots



b) The Major Functions Studied by Psychologists:

1) Perception (including sensation and concept formation)
2) Remembering
3) Problem-Solving
4) Language Understanding
5) Acting
6) Motivation by Needs and Goals
7) Learning

c) A Rough Sketch of the Functions and the Flow of Information in an Integrated Cognitive System



that they can be related and manipulated. These assertions are then transformed, according to rules for valid deductive inferences, by the problem-solver, which tries to find a path from the present floor-plan (as further abstracted into a set of assertions) to the commanded state. Finally, if and when a successful path is found, its sub-steps are bound to objects in the floor-plan, and converted into commands to motor effectors (e.g., wheels, pulleys) that move the robot, or its arm, about the room.

2.3 THE DESIRABILITY OF RICHER AND CLOSER INTERACTIONS AMONG PROCESSES

It seems far more reasonable to have a system use perceived world, understood command, and partially-deduced solution to guide one another. For example, the command "MOVE BOX NEXT-TO PYRAMID" should set the system to try to perceive a PYRAMID rather than look for all possible objects. Further, when either object has been found, only the nearest instance of the other need be perceived. If only one is perceived, it should try to deduce how to find the other - for example coming up with the idea that it might move to another room, by going through a door, and at that point try to perceive a door. If the perspective of the present picture makes it hard to decide whether an object is a pyramid or a wedge, rather than exhausting itself with complex scene analysis, it should decide to move to a position where a new picture would easily resolve the difficulties. Feedback from constant small perceptual glances should guide the system's movements toward and around objects.

This kind of interaction has not been done with large robot systems. It may well prove impossible, because of the enormous size and complexity of each of the separate subsystems (and their sub-systems), and the attendant difficulty of passing information between subsystems, and using this information fruitfully.

2.4 SIMULATED ROBOTS AND WHOLE SYSTEMS

Several attempts have been made, on a much smaller scale, to examine more integrated systems. Sutro and Kilmer [17], following the ideas of Kilmer [10] and of McCulloch and Pitts [11], began to design and build special-purpose parallel-serial nerve-net computers (to some degree modeled on living brains, and in particular the visual system) that, it was hoped, would be able to recognize and respond appropriately to objects. Toda [18] examined the question of an organism exploring and gathering sustenance from a strange environment. Uhr and Kochen [29] programmed a simple "MIKROKOSM" system, where the environment, as well as simple organisms, were all simulated within the computer. Becker [1] has made an interesting exploratory analysis of some of the problems of robot cognition, and has formulated and coded a simulation of a simple robot moving through a rectangle of simulated streets. Siklossy [15] has made some significant improvements on the SRI "STRIPS" system (Fikes and Nilsson [7]), again with a software simulation. Winograd [30] has developed one of the nicest systems for manipulating "objects" in response to complex commands where "objects" are represented by internal lists in the computer, and displayed as line-drawings on a graphics scope.

I should note that much of this software simulation

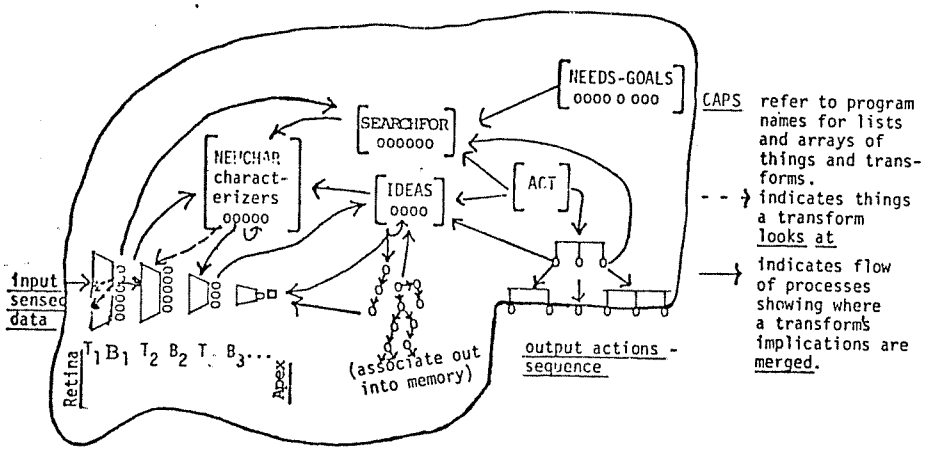
was done in conjunction with the hardware robot projects - Becker's at Stanford, Winograd's at MIT, and the STRIPS system (which originally was a simulation) at SRI.

3. THE SEER-2 SYSTEM

SEER-2 builds upon, integrates and extends several previously reported systems. A parallel-serial perceiver that recognizes and describes 2-dimensional scenes of objects is combined with a more central cognitive system that makes heuristic searches and deductions through a cognitive network, and then decides what type of response is appropriate, as a function of the presses of a) perceived objects, b) understood verbal utterances, including commands and suggestions, and c) internal needs and desires.

Figure 2 gives the overall flow of SEER-2 (see Appendix A for the actual program).

Figure 2. The Overall Architecture of SEER-2



The perceptual recognition cone inputs a scene into its retinal array. Then the first layer of transforms (T_1) looks at this buffer, merging all implied things into corresponding locations in their output Buffer array (B_1), and merging all characterizers to apply into NEWCHARS and all things to search for, into SEARCHFOR. Each layer of transforms T_2, T_3,...T_n looks at its input buffer and merges its implied things into its output Buffer, SEARCHFOR, NEWCHARS, and IDEAS, as specified. The dynamically implied NEWCHARS are also applied at each layer.

When the cone has finally converged to the Apex Buffer (which now contains all found objects, parts and qualities), the IDEAS transforms are applied, one at a time - the most highly weighted one next. Their implieds are similarly merged into the Apex, IDEAS, SEARCHFOR, NEWCHARS and ACT, as specified. This serves to make associative and deductive searches, heuristically directed by the merged weights of a variety of implications, through the cognitive memory network, until an act is sufficiently highly implied to be chosen, and effected.

The same general type of transform is used in IDEAS, NEWCHARS, and each perceptual transform layer. The same general type of thing is implied, and merged into the ACT, NEEDS-GOALS, and SEARCHFOR list, and also the Apex and each Buffer array of the recognition cone. Thus any transform can look anywhere, and merge its implieds anywhere.

As information flows and is transformed from the input retina to the central apex of the recognition cone, successive layers of perceptual transformations (that preprocess, extract features, characterize, build compounds, imply names of wholes, parts and qualities, and imply further things to look for and transforms to apply) build up a description of the external scene. Cognitive transforms (called "IDEAS"), which have themselves been implied by other transforms, perceptual and cognitive, make further associative and deductive searches into

memory, implying still more things into the apex, and still more transforms to apply into the IDEAS list. When the most highly implied IDEA is an ACT the system attempts to bind that ACT to a specific sequence of actions on specific objects.

The entire SEER system uses the same general type of probabilistic configurational transform, one that is general enough to handle both structures of objects (including features, pre-procesing transforms, parts, and wholes), and of language (including syntatic and semantic context-sensitive rewrite rules and transformations). This allows for rich interaction between all parts of the system. It makes possible the mixed recognition of words and objects, and the ability to decide as a function of a variety of external and internal influences.

3.1 PARALLEL-SERIAL PERCEPTION OF SCENES OF OBJECTS AND WORDS

A parallel-serial layered "recognition cone" system has been developed to recognize objects [20] and describe scenes [25]. It uses a single general type of configurational transform to a) pre-process (e.g. smooth, find edges), b) imply internal objects that can then be taken as parts of larger structures, c) imply external names that identify objects, and also describe their parts, qualities, and interrelations, d) trigger intermediate decisions as to what has been perceived, and e) imply further things to look for and further transforms to apply, thus allowing it to "glance about." This allows it to build up descriptions of objects, and of scenes of objects. Among the objects can be words, and larger structures of words (e.g. phrases, commands, suggestions).

A set of transforms is applied in parallel to each successive layer of the cone, and all implied outputs are stored in the corresponding cell of the next layer. Each layer is smaller than the last, so that as information is abstracted by the transformations it is coalesced. This serial application of parallel sets of transforms continues until the cone has been collapsed into a single Apex cell that contains information about all implied things, and their locations, parts and qualities.

3.2 INTEGRATION OF THE PERCEPTUAL SYSTEM WITH INTERACTING CENTRAL COGNITIVE PROCESSES

A central cognitive processor (modelled after DECIDER, as described in Uhr [22, 24]) acts upon this transformed, recognized and "understood" scene in the Apex of the recognition cone. DECIDER uses the single list of transforms stored on a central "IDEAS" list. These transforms (which have themselves been implied by perceptual transforms and/or other cognitive transforms) are applied serially, the most highly implied being applied next. (Note that this is in sharp contrast to the parallel application of transforms at each layer of the perceptual system. Future SEER systems will introduce varying amounts of parallel processing to the central cognitive system, and will examine ways of combining the perceptual and cognitive systems more intimately. But already "recognition" is very similar to "perception" - the IDEAS list transforming the Apex just like the perceptual characterizers transform the layers of the recognition cone).

This serial application of transforms serves to search deeper into the system's associative

cognitive network memory, thus accessing and retrieving information in answer to questions whose import has been recognized, and also deducing consequences. This search is heuristically directed by the weights associated with the various implications of the transforms that have been merged together.

New transforms to apply are also implied by transforms used during perception, and therefore are a function of recognized objects, words, and qualities. They are also implied by internal needs, and by goals that are themselves implied by what has been understood so far about the scene.

3.3 DECIDING UPON A TYPE OF ACT, AND EFFECTING THE APPROPRIATE ACTIONS-SEQUENCE

SEER-2 continues in its heuristic application of transforms until it finds that the most highly implied transform is an act, at which point it binds and executes that act. This can entail further deduction of the specific actions-sequence needed to effect that act, and further recognition of needed objects.

Thus acts are simply another type of thing that a transform can imply. Possible acts are chosen among using the same central "IDEAS" list that searches and deduces into memory. SEER-2 therefore gathers information, through perception, language understanding, retrieval from its internal cognitive network, and deductions. Each piece of information found implies further things to do, until an act is chosen, as the most highly implied thing.

3.4 INTERNAL NEEDS AND GOALS

SEER-2 has a built-in list of internal "needs," with a weight associated to each. These needs point down to things (classes, and objects) and goals that will satisfy them, and these classes and things similarly point down to things and parts, and to transforms that will characterize and hence help to recognize them.

Similarly, external scenes, when they are partially recognized, will imply possible objects, and higher-level classes and constructs of objects, that might be there. These and the things they point down to are similarly merged onto the lists of goals, things to search for, and characterizers to apply. When parts of scenes are recognized as verbal utterances that suggest goals (e.g. "LOOK FOR THE DOG" or "GET FOOD") these goals, and the transforms they point down to, are treated in the same way.

Built-in internal needs and higher-level goals are similarly treated, whether internal or externally induced by the press of either perceived objects or understood verbal utterances. This gives a "top-down" component, directing SEER-2's search to glance about, in both internal memory and the external scene, at the same time that the scene itself presses inward in a "bottom-up" direction.

3.5 THE USE OF GENERAL TRANSFORMS FOR PERCEPTION, LANGUAGE PROCESSING, RETRIEVAL, AND PROBLEM-SOLVING

SEER-2 uses a general type of transform at all steps of processing, whether perceptual or cognitive. This means that transforms can be implied and merged onto any perceptual or cognitive transform list from any other list, as when perception suggests information that should be retrieved, or deduction suggests further objects or their parts that should be perceived. It further means that different types of transforms can be mixed together. For example, the moment a low-level simple object is recognized the system can begin to access related things about it in cognitive memory, even though it is still, in general, building up its perceptual description of the input scene.

The similarities between linguistic rewrite rules, their extensions to do "syntactic" pattern recognition, and configurational characterizers have been explored [19], and systems have been coded to handle sets of general transforms capable of recognizing language structures (e.g. words, phrases and commands) and object structures (e.g. edges, noses, faces, people, and families), hierarchically building them into larger and larger wholes [9, 21, 22, 24, 26]. This general transform capability can also be used for the kinds of memory searches done by semantic networks (e.g. Quillian [13]) and question-answering systems (e.g. Simmons [16]), and some shallow deductive inference (see Figure 3).

Figure 3. The General Configurational Transform Used, and Examples of How it Handles the Different Cognitive Processes

a) General form of a transform (Weights, thresholds and relative locations are also specified when needed (as by perceptual characterizers and memory associators. They are not shown here.)

$Feature_{11}$, $Feature_{12}$,...,$Feature_{1N}$ =>

$Implied_1Attributes_1$, $Implieds_2Attributes_2$,...

b) Examples showing how different types of transforms are represented, using this general form.

i) Linguistic rewrite rules for parsing:

Article 1 Adjective 2 Noun 3 =>

Nounphrase (Note: successive locations 1, 2, 3 indicate concatenation)

ii) Productions for deduction:

P + Q => Q + P

iii) Associations for searches through a memory network:

Robin = Bird, Fly, Red-Breast, Animal, Harbinger-of-spring.

iv) Feature extractors for simple pattern recognition:

Small-closed-loop => P, B, R, Eye, Dumbbell, Eyeglasses.

v) Configurational characterizers for scene description:

Vertical 0-0, Closed-loop 0-1 => B P, Eyeglasses, Characterizers (Face)

(Note: Coordinate locations 0-0, 0-1 indicate relative positions)

Briefly, a memory "association," a linguistic "rewrite rule," or a "production" of a theorem

prover or game player, are all represented as a configuration all of whose parts must be found for the transform to succeed, along with the specified relations (e.g. concatenated, or touching, or ordered). But now, in addition, weights, relative locations and other attributes can be used - as they must be used for pattern perception of real-world distorted objects. This allows for more probabilistic and fuzzy linguistic and deductive processes- which appear appropriate for real-world problems.

4. SOME EXAMPLES OF THE BEHAVIOR OF SEER-2

The following examples give some flavor of the variety of things that SEER-2 can do (see [22, 26, 28] for additional examples, worked through in detail). Each exemplifies a wide variety of input problems - different representations of the objects and ways of expressing the commands and suggestions - that SEER-2 can handle given an appropriate memory network of transforms (see Appendix).

The crucial point is that the same SEER system can do a number of different types of things, and must itself make an essentially fuzzy decision as to which type of thing is appropriate to do.

4.1 NAMING OBJECTS AND DESCRIBING SCENES

When given the inputs shown in Figure 4a, SEER-2 will output 1) TABLE, 2) TABLE, 3) CHAIR, 4) FACE.

When given the inputs shown in Figure 4b, SEER-2 will output:

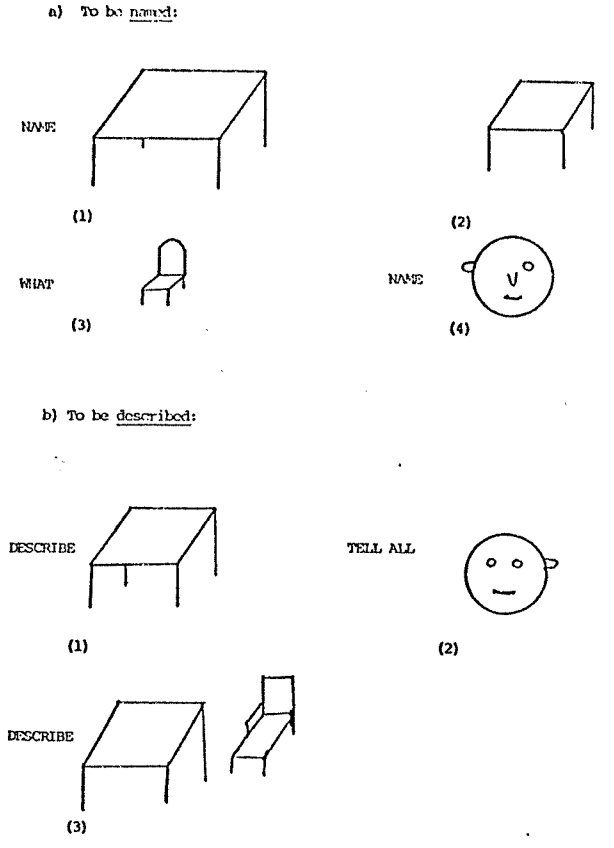


Figure 4 Scenes of Objects and Words Mixed Together

1) TABLE (WITH TOP; LEGS)

2) FACE (WITH VISAGE; EYE; EYE; MOUTH; RIGHTEAR;)

3) TABLE (WITH TOP; LEGS;)

CHAIR (WITH BACK; SEAT; ARM; LEGS;).

These examples illustrate SEER-2's ability to handle 2-dimensional scenes, where objects and words are mixed together. Larger, more detailed, more variable and more complex scenes can be handled, when SEER-2 is given, or has learned, a larger set of transforms.

For simplicity, the rest of the examples of SEER's behavior will work on 1-dimensional inputs. This allows us to ignore the first few stages of the perceptual process (which can be handled by SEER-2, as indicated in the examples above). Either SEER-1 or SEER-2 will handle these kinds

of problems. For these examples we will use the convention of @(object name) to indicate objects, object parts, and qualities, as a shorthand for the actual picture.

For example, to the inputs:

"WHAT IS THIS @APPLE" or

"NAME @APPLE THIS"

SEER will respond: "APPLE".

To the inputs:

"SAY WHAT @STEM @RED @OVAL YOU SEE" and

"WHAT DO YOU SEE @STEM @TEARDROP"

SEER will respond "APPLE", "PEAR" in turn.

4.2 RETRIEVING INFORMATION, ANSWERING QUESTIONS AND CONVERSING

When the input scene contains a question like:

"WHO IS PRESIDENT" or

"NAME CITIES IN WISCONSIN"

SEER will output "FORD" and "MILWAUKEE; MADISON; RACINE;".

(Uhr [22, 24, 26] and Jordan [9] give detailed examples that show how such an internal retrieval search is done by similar but more powerful systems.)

Verbal inputs that are not precise questions, but are rather looser conversational suggestions, are handled as follows:

"TELL ABOUT WISCONSIN" gives (depending on specific transforms):

"COLD;COWS;LIBERAL;"

"TALK ABOUT APPLE" gives:

"FRUIT;CRISP;"

Note how similar a conversation is to a description of a scene, and how a description can be of external objects or of internal representations. In both cases there is no single "correct" answer and, indeed, real-world descriptions and answers must usually be chosen using fuzzy criteria of relevance. (See [21, 23, 25] for further discussions).

4.3 DEDUCING RESPONSES

SEER-2's deductive capabilities are weak. But it is worth mentioning how in fact it can compute arithmetic, and search for and deduce consequences when they are explicitly stored in memory.

For example,

"ADD TWO + 3" gives:

"5" and

"PLAY #X.X#YX.#Y.Y#" (a tic-tac-toe board in 1-dimension) gives:

"PLAY #XXX(IWIN)#YX.#Y.Y#"

4.4 FINDING AND MANIPULATING OBJECTS, DRIVEN BY EXTERNAL AND/OR INTERNAL PRESSES

Finally, SEER-2 can decide to act upon the external environment.

A command like:

"FIND THE @PEAR APPLE @BANANA @APPLE" gives:

"FIND THE @PEAR APPLE @BANANA :@APPLE:" (where colons around @APPLE indicate it has been found, as though picked up by a hand). Or:

"MOVE @APPLE THE FRUIT TO @CRACKER THE BOX @PAIL @BOX" gives:

"MOVE THE FRUIT TO @CRACKER THE BOX @PAIL (@BOX @APPLE)" (where parentheses indicate the objects within them have been moved together).

Presses from external objects and/or internal needs can also lead to acts. E.g.:

"@STICK @APPLE @BOX @SEER-MOUTH" (along with an internal need-state of hunger) gives:

"@STICK @BOX (@SEER-MOUTH @APPLE)"

Here the several implications, from the recognized external object and internal need, are combined - just as the several characteristics of a perceived object are combined - and their resultant leads to the act. When several acts are implied, the system chooses to effect the single most highly implied act.

4.5 CHOOSING AMONG DIFFERENT TYPES OF BEHAVIOR

The examples above do not indicate how multi-determined is SEER's behavior in more complex situations where a number of transforms are needed, as in recognizing distorted and noisy patterns, or remembering more subtly related information, or choosing an act as a function of a variety of internal and external presses. To handle such complex, and more realistic, problems SEER must be given a larger set of transforms, with weights or fuzzy values associated with their implications. The mechanisms for handling these - for combining weights and choosing among alternate possibilities - are already present. But extensive tests are needed to see how well this system, when given a larger set of transforms, will handle a larger variety of problems.

There is no reason to expect such a system to be right all the time, or to give very surprising or difficult answers. People are often wrong, and only the very unusual person is original, and only rarely.

5. DISCUSSION

SEER's behavior depends upon the set of transforms stored in its cognitive network - just as a parser depends upon the specific rewrite rules given it. More transforms are needed as the variety of things - objects, variations, qualities, words, and higher-level groupings - increases. It is expected that SEER's learning mechanisms will generate and hypothesize new transforms, as a function of its experiences with its environment, and try to discover which hypothesized transforms improve its performance, and adjust and tune weights and other parameters of these generated hypotheses. It will probably fail on particular problems on which a specifically pre-programmed system would succeed; but its generality and flexibility should allow it to handle a far greater variety of problems. Until much larger tests have been made, to examine behavior on a wider variety of problems over a larger set of possible environmental inputs, we cannot judge whether the network will become too large to store, or too slow to succeed, in any reasonable amount of memory space or computing time. Nor will we know its limits, or whether the ability to handle some problems will be interfered with by the transforms needed to handle the large set of other, often closely related and hence capable of interfering, problems. I should note that this difficulty is common to all artificial intelligence systems built to date. None have been tested on anything but "toy" artificially small domains of problems (e.g. [6, 12, 29, 30]). We have only vague feelings as to how well they might expand to more realistic domains, and in the occasional cases where the attempt has been made they have all-too-quickly run out of memory space (e.g. [7, 13]).

I can only suggest that the cognitive network used

in SEER appears to me as general and efficient as any. It uses internal classes and configurational threshold transforms, both of which generalize over large sets of transforms that would have to be stored separately by the typical, far more deterministic, networks. I would further argue that learning is the key to keeping the memory small, general, flexible and pertinent to the world the learner must handle. The SEER network is designed with learnability in mind, and a number of relatively powerful learning mechanisms are now being incorporated into it.

SEER-2 can do a variety of different things, as a function of what it gleans from recognized external words and objects, and internal presses. When an input contains a verbal command or suggestion there is no built-in signal that these are words that form a command it must follow. On the contrary, a word must first be recognized along with all other sensory inputs, and then its significance and the import of any larger utterance of words containing it must be understood, using the same cognitive network of transforms used for perception, search and deductive inference.

6. ADDITIONAL CAPABILITIES AND PROJECTED EXTENSIONS

SEER has several important capabilities which, because of limitations of space, will be described and illustrated more fully elsewhere. These include:

1) A capability to handle scenes that change over time - recognizing things (like spoken words and moving objects) that pass into and out of view;
2) An ability to act in real-time, executing only one action from the chosen actions-sequence at each moment of time, monitoring the entire act for expected consequences from the environment. It thus continually re-assesses whether to continue with an act, or abandon it for a now more highly implied new act;
3) Searches and acts that are a function of internal needs which themselves change over time, and of expectations and goals that have been implied by already-found things;
4) Capabilities to <u>learn</u> to do many of the things shown in the examples.

Several further extensions and improvements are now being coded. These include:

1) The ability to merge inputs from several different sensors, to begin to handle
   a) stereoscopic depth perception and
   b) several sensory channels, e.g. sight, sound, touch;
2) A single general system that combines the perceptual system with the system that searches and deduces.
3) A system that can do parallel processing at the central cognitive level of the IDEAS list, and can adjust the degree of parallelness, to try to fit the problems being handled;

4) Improved deductive capabilities;

5) A more general way to handle the binding of actions-sequences as a part of the central cognitive processes;

6) Techniques for controlling and directing attention and effort after the system has focussed upon particular problems and acts (these seem reminiscent of "consciousness");

7) More powerful learning capabilities.

The major problems to be investigated center around the more intimate combining of the relatively parallel perceptual processes with the relatively serial cognitive processes, including special loci of controlled "consciousness," to give greater direction to problem-solving. This integration must be done in such a way as to enhance the "cognitive model" aspects of the memory, and to make learning as simple and powerful as possible.

REFERENCES

(1) Joseph D. Becker, An information processing model of intermediate-level cognition, Unpubl. Ph.D. Diss., Stanford Univ., 1970 (Also BBN Tech. Rept.)

(2) James E. Doran, Experiments with a pleasure seeking automation, In: Machine Intelligence 3 (D. Michie, Ed.) Edinburgh Univ. Press, 1968, 195-216.

(3) Hans A. Ernst, MH-1: a computer operated hand, Proc. SJCC, 1962, 21, 39-55.

(4) Hans A. Ernst, Computer-controlled robots, IBM Res. Rept., RC 2781, Yorktown Heights, 1970.

(5) Masakazu Ejiri, et al., An intelligent robot with cognition and decision-making ability. Proc. 2nd Int. Joint Conf. on Artificial Intell., 1971, 350-358.

(6) Jerome A. Feldman, et al., The Stanford hand-eye project. Proc. 2nd Int. Joint Conf. on Artificial Intell., 1971, 521-526.

(7) Richard E. Fikes and Nils J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, Proc. 2nd Int. Joint Conf. on Artificial Intell., 1971, 608-620.

(8) Ralph E. Griswold et al., The SNOBOL4 programming language, Prentice-Hall, Englewood-Cliffs, 1968.

(9) Sara R. Jordan, Learning to use contextual patterns in language processing, Unpubl. Ph.D. Thesis, Univ. of Wisconsin, 1971.

(10) William Kilmer, W. McCulloch and J. Blum, Embodiments of a plastic concept of the reticular formation, Proc. Symposium on Biocybernetics of the Central Nervous System, Little Brown, Boston, 1968.

(11) Warren S. McCulloch and W. H. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical biophysics, 1943, 5, 115-133.

(12) Nils J. Nilsson, A mobile automaton: an application of A.I. techniques, Proc. 1st Int. Joint Conf. on Artificial Intell., 1969, 509-520.

(13) M. Ross Quillian, The teachable language comprehender: a simulation program and theory of language. Comm. ACM, 1969, 12, 459-476.

(14) Bertram Raphael, Programming a robot. Proc. IFIP Congress 68, Edinburgh, 1968, H135-140.

(15) Laurent Siklossy and J. Dreussi, An efficient robot planner which generates its own procedures, Proc. 3d Int. Joint Conf. on Artificial Intell., 1973, 383-387.

(16) Robert Simmons, Natural language question-answering systems: 1969, Comm. ACM, 1970, 13, 15-30.

(17) Louis Sutro and W. L. Kilmer, Assembly of computers to command and control a robot, Proc. SJCC, 1969, 34, 113-138.

(18) Massima Toda, The design of a fungus-eater: a model of human behavior in an unsophisticated environment, Behavioral Science, 1962, 7, 164-183.

(19) Leonard Uhr, Flexible linguistic pattern recognition, Pattern Recognition, 1971, 3, 363-384.

(20) Leonard Uhr, "recognition cone" networks that preprocess, classify and describe, IEEE Trans. Computers, 1972, 21, 758-768.

(21) Leonard Uhr, Pattern Recognition, Learning and Thought, Prentice-Hall, Englewood-Cliffs, 1973.

(22) Leonard Uhr, Recognizing, "understanding," deciding whether to obey, and executing commands, Computer Sci. Dept. Tech. Rept. 173, Univ. of Wisconsin, 1973.

(23) Leonard Uhr, The description of scenes over time and space, Proc. AFIPS NCC, 1973, 42, 509-517.

(24) Leonard Uhr, DECIDER-1: a system that chooses among different types of acts. Proc. 3d Int. Joint Conf. on Artificial Intell., 1973, 396-401.

(25) Leonard Uhr, Describing, using "recognition cones," Proc. 1st Int. Joint Conf. on Pattern Recognition, 1973. (Also Computer Sci. Dept. Tech. Rept. 176, Univ. of Wisconsin, Madison, 1973.)

(26) Leonard Uhr, Semantic Learning, manuscript in preparation.

(27) Leonard Uhr, EASEy-2: An English-like program language, Computer Sciences Dept. Tech. Rept. 178, Univ. of Wisconsin, 1973.

(28) Leonard Uhr, Toward integrated cognitive systems, which must make fuzzy decisions about fuzzy problems. Computer Sciences Dept. Tech. Rept. 222, Univ. of Wisconsin, 1974. (To be published in L. Zadeh, et al., Eds. Fuzzy Sets, New York: Academic Press, 1975.)

(29) Leonard Uhr, and M. Kochen, MIKROKOSMs and robots, Proc. 1st Int. Joint Conf. on Artificial Intell., 1969, 541-556.

(30) Terry Winograd, Procedures as a representation for data in a computer program for understanding natural language, Unpubl. Ph.D. Diss., MIT, 1971. (Also published in Cognitive Psychology, and as an Academic Press book).

(31) Patrick H. Winston, The MIT robot, In: Machine Intelligence 7 (B. Melzer and D. Michie, Eds.) Edinburgh Univ. Press, 1972.

APPENDIX A: THE SEER-2 PROGRAM

The SEER-2 program is coded in EASEy (see Uhr [27] and Appendix D for a brief description), an English-like programming language that was designed to be easy to read, hence a good vehicle for communication about programs. EASEy exists as a SNOBOL program (see Griswold [8]) that translates from EASEy to SNOBOL. An EASEy program is therefore a complete and precise specification of the system, one that will actually run on a computer.

The program is preceded by an Overview that briefly explains what successive groups of statements do. A set of transforms must be given to, or learned by, the program (see Appendix B for transforms sufficient to handle a variety of problems, including the example problems given in this paper and in [28]).

(Overview of the SEER-2 program. Program Statement No.

START	Initialize Memory (Includes type-name to refer to lists and the Transforms repeatedly used - ERASER, NORMALIZE, PASSON.	M1-M9
IN	input Memory, including NEW lists and ADDitions	1
	(The MEMory conversion program, which inputs the transforms needed (Appendix B) goes here.)	I1-I4
	(The environments - scenes and problems - to be sensed are also input here.)	
INITialize	for the next input problem, by erasing all temporary lists.	I5
SENSE	Store the input as an array, in the first Layer of the recognition cone.	2-7
TRANSform	MERGE the things that POINTAT NEEDSGOALS and SEARCHFOR into NEWCHARacterizers	8,9,12
TREPEAT	Initialize the LAYERS and CHARacterizerS TODO	10
T6	Get the next layer to NOWDO and its STEP-size	11
	Put the ERASER, NORMALIZER and PASSON transforms, and NEWCHARS, onto NOWDO	14
T5	Get the next TRANSform and attendant information from NOWDO.	15
	Get the bounds for applying the TRANSform, and its TYPE	16-18
T3	Get the pieces of its DESCRiption, and handle them as indicated by TYPE	20-24
A1	MERGE all members of the specified CLASS into the next Layer, to average or difference.	21
I1	Accumulate the TOTAL weights of the CLASS members that are GOT	22-24
I2	If TOTAL exceeds THRESHold, MERGE IMPLIEDS into the next Layer	25-26
A2	Iterate through the array until the bounds (CMAX, RMAX) have been reached.	27-28
E1	erase each cell in the next Layer, to initialize it.	29
N1	NORMALIZE each cell, to keep weights roughly constant despite convergence.	30
ITER	Go to process the next Layer, shrinking Row and Column by STEP-size, until the apex is reached	31-33 34
THINK	Initialize and go through up to 7 CYCLES	35-37
	Get NEWCHARS that POINTAT SEARCHFOR and set them on IDEAS	38-40
	CHOOSE the most highly weighted THING on IDEAS, up to 100	41-45
(Cycles through IDEAS until an ACT is the most highly weighted and therefore chosen)		
I	If TYPE is I, this is an Internal transform to apply to FOUND (the apex). If the TOTAL weight of CLASS members GOT exceeds THRESHold, MERGE IMPLIEDS.	46-54
ACT	If TYPE is ACT, get the next ACTION and its ARGumentS and other ARGuments from HISTory	55-57
OUT	output the result of the act (after having completed it with routines below)	58-60
SEARCH	If the act couldn't be completed, set up a SEARCH for things that POINTAT the NEEDED PARTS, and return to apply more IDEAS.	61-66
FAIL	output that have "FAILED" to execute the chosen act, and go to the next input.	67
(The routines for the different types of acts follow.) (Note how Describe uses Name, and Move uses Find.)		
D	Describe the scene, giving the objects and their parts	68-76
T	Name the single most highly weighted thing of the class specified in ARGument.	69-71
F	Find the first THING that is a member of the specified ARGument and surround it with colons (:thing:)	77-79
M	Move all Found THINGs FROM or TO (as indicated) the TARGET thing	77-85
R	Reply to a query by CHOOSEing all THINGs belonging to the specified ARGument whose weight exceeds half the MAXWeighTed THING (i.e. the first) chosen	86-90
SEARCHR	If no Reply was chosen, MERGE associations from what already found into IDEAS and return to think some more, by applying more IDEAS.	91
C	Compute, using the specified operators (ADD, subtract and divide are shown)	92-99
SEARCHC	If the needed numbers were not recognized, return to search for the ARGuments (OPerator and NUMBER in e.g.)	96
G	In a Game, make a move by replacing the old board configuration with the move	100-104
(The following are the functions used by the main program.)		
MERGE	Get the THINGs on LISTA that belong to the specified CLASS and MERGE them, combining TOTAL WeighTs and HISTories, into LISTB.	ME1-ME13
POINTAT	Get the TOTHINGS that POINTAT each THING specified, and MERGE them into the lists specified.	PA1-PA4
NORMALIZE	Divide the WeighT of each thing TONORMalize by NORM, erasing it if WT is below 1.	NO1-NO5
ABS	Get the ABSsolute value of the argument (fails if the argument is not an integer).	ABS1-ABS2
CHOOSE	CHOOSEs the MAXimum (or if TYPE specifies MIN the MINimum) weighted thing in LISTA of the specified CLASS, getting the THING, TOTAL, LOC and HIST.	CH1-CH14

```
(Program SEER-2)                                                    RECODES-1 (see Uhr, [25])   SEER-2

(Initialize some of the Memory lists                                                            M1
START    set  ERASER =   '0 0 R C [ '                                                           M2
              NORMALIZE =   '0 0 R C N '                                                        M3
              PASSON =   '0 0 R C P] '                                                          M4
              P1 = A %D=  0 0 ]%I=%'                                                            M5
         set, SPOTSIZE = 1                                                                      M6
              C = '$(0; 0; 0)                                                                   M7
              N = 'NEWCHARS'                                                                    M8
              S = 'SEARCHFOR                                                                    M9
              I = 'IDEAS'
              NEEDSGOALS = ':HUNGER 10 ]:THIRST 10 ]'                                           M10

(INput and go to TYPE (initialize memory, SENSE or TRANSform)                   1               1
IN       input TYPE DESCR ] [ +to $('M' TYPE]/)-to end]                         .1              I1
(MEMory  input and format routine goes here-only the beginning is shown.
(input      NEW lists or ADDed information on old lists
MNEW     from DESCR get NAME =
         set  $NAME = DESCR    [to IN]                                          .2              I2
MADD     from DESCR get NAME =                                                  .3              I3
         on $NAME set DESCR    [IN]                                             .4              I4
MINIT    erase R, L, EXTERNAL, TOOUT, NEWCHARS, SEARCHFOR, IDEAS, ACTIVEWT                      I5
(Input and SENSE a new scene into Layer 0 (the retina)
MSENSE   erase C                                                                2               2
         on EXTERNAL list DESCR                                                 .1              3
S1       from DESCR get and call SPOTSIZE symbols SPOT erase [-to S2]           3.V             4
         list  $(L;R;C) = 'BRIGHT QUAL :BRIGHT ' SPOT ]                         4.V             5
         C = C + 1 [to S1]                                                      5               6
S2       R = R + 1 [to IN]                                                      6               7
MTRANS   output DESCR ' SCENE HAS BEEN INPUT, IS BEING TRANSFORMED'             7               8
(NEEDS and GOALS imply things to LOOKFOR that POINTAT them
         MERGE(POINTAT(NEEDSGOALS),S)                                           .1              9
TREPEAT  TODO = LAYERS CHARS                                                    8               10
T6       from TODO get STEP NOWDO % = [-TREPEAT ]                               9               11
(LOOKFOR implies NEWCHARacterizers that point at them.
         MERGE(POINTAT(SEARCHFOR), N)                                           .1              12
         erase SEARCHFOR                                                        .2              13
(To ERASE next LAYER,NORMALIZE, apply NEWly implied CHARacterizers, and PASSON
(found things
   at start of NOWDO set ':ERASER X ]:NORMALIZE X ]' NEWCHARS ':PASSON X ]'     10.V            14
(Get each TRANSform from NOWDO (to be applied at this layer)
T5       from NOWDO get CLASSES : TRANS WT HIST ]= [-ITER]                      11.V            15
         from $TRANS get RA CAA RMAX CMAX DO                                    12              16
                                                                                13              17
T7       CA = CAA
T4       from $DO get TYPE THRESH '%D=' DESCR '%I=' IMPLIEDS % [-$(DO 1)]       14.V            18
         erase GOT TOTAL                                                        15              19
T3       from DESCR get : CLASS TVAL DR DC ]= [ +$(TYPE 1) - ($(TYPE 2) ]       16.V            20
(TYPE A transform Averages or differences.
A1       MERGE($(L;RA+DR;CA+DC),'$(L+1;RA/STEP;CA/STEP),,TVAL,CLASS) [T3]       17.V            21

(TYPE I transform characterizes
I1       from $(L;RA+DR;CA+DC) get # that CLASS # REST : THING VAL [-T3]        18.V            22
         is TVAL lessthan VAL ? yes. TOTAL = TOTAL VAL - TVAL [-T3]             19.V            23
         on GOT list CLASS THING  [T3]                                          20.V            24
(the characterizer succeeds if the TOTAL weight reaches THRESHold.
I2       is THRESH greaterthan TOTAL ? [ + A2 ]                                 21.V            25
         MERGE(IMPLIEDS, '$(L+1;RA/STEP;CA/STEP)', GOT)                         22              26

(keeps applying this TRANSform till its upper bounds are reached.               RECODES-1       SEER-2
A2       is CA lessthan $CMAX? yes - CA = CA + 1  [+T4 ]                        23              27
         is RA lessthan $RMAX? yes - RA = RA + 1  [+T7-T5 ]                     24              28
(Erases the next layer, to re-initialize.
E1       erase $(L+1; RA/STEP ; CA/STEP) [A2 ]                                  25              29
(Normalizes, dividing by square of STEP-size)
N1       $(L;RA;CA) = NORMALIZE($(L;RA;CA)STEP**2) [A2]                         .1              30
(Converge to the next layer.
ITER     L = L + 1                                                              26              31
         R = R / STEP                                                           27              32
         C = C / STEP                                                           28              33
(If the apex has been reached, start to "THINK" (apply IDEAS)
         is R lessthan 1 ? is C lessthan 1 ? [ -T6]                             29              34

(CYCLES 7 times applying 100 transform from IDEAS to FOUND (the apex)
THINK    CYCLES = 7                                                                             35
TCYCLE   CYCLES = CYCLES - 1                                                                    36
         is CYCLES lessthan 1 ? [ + FAIL ]                                                      37
         MERGE(POINTAT(SEARCHFOR),N)                                                            38
            on IDEAS set NEWCHARS                                                               39
         erase SEARCHFOR, NEWCHARS                                                              40
         set TRIES = 100                                                                        41
TMORE    TRIES = TRIES - 1                                                                      42
         is TRIES lessthan 1 ? [+ TCYCLE ]                                                      43
(CHOOSES most highly weighted TRANSform from IDEAS, serially choosing maximum weighted)
         from IDEAS get CHOOSE(IDEAS) = [ -TCYCLE]                              9.V             44
(Applies the TRANSform to the FOUND in the apex
         from $THING get TYPE THRESH '%D=' DESCR '%I=' IMPLIEDS % [$TYPE]       11.V            45
I        set FOUND = $(L;0;0)                                                   10.V            46
(Loops if All indicated (should use LOCs and get nearest.)
TH1      from DESCR get CLASSES : CLASS WT REST ] = [-EVAL]                      12.V            47
         from CLASS get 'S' ALL = [ +THA ]                                                      48
           ALL = 1                                                                              49
THA      from FOUND get LEFT # that CLASS # RIGHT : THING WTF = [-TH1]          15.V            50

         on GOT list CLASS THING                                                16.V            51
         TOTAL = TOTAL + WT * WTF ; $('TH' ALL)]                                17.V            52
EVAL     is THRESH greaterthan TOTAL ? [-TMORE ]                                18.V            53
         MERGE(IMPLIEDS,C,GOT) [TMORE ]                                                         54
(ACTs, because an act was chosen from IDEAS.
ACT      from IMPLIEDS get ACTION ARGS ; = [- OUT]                                              55
           ARG = ARGS [$ACTION]                                                                 56
ACT2     from HIST get CLASS ARG = [+ $ACTION]                                                  57
(OUTputs its actions-sequence
OUT      is TOOUT sameas EMPTY ? [+ SEARCH ]                                                    58
         output        TOOUT                                                                    59
         output        EXTERNAL [IN]                                            44.V            60
(Initiates a SEARCH to help in completing the frustrated act.
SEARCH   from $APG get '%I=' IMPLIEDS % [- FAIL ]                                               61
SEARCH2  from IMPLIEDS get IMPLIED WT ; = [- FAIL]                                              62
         from IMPLIED get NEEDED '$' [-SEARCH2 ]                                                63
         from $NEEDED get '%D=' PARTS %                                                         64
         MERGE(POINTAT(PARTS),N)                                                                65
```

```
                                                                    RECODES-1   SEER-2
(Will return this ACT to IDEAS (from NEWCHARS) at next CYCLE                        66
RETURNACT   on NEWCHARS set CHOOSE : TMORE '                                        67
FAIL        output 'FAILED' EXTERNAL  [IN ]                                         68
D           [T]
(Names the most highly implied thing  of class specified in ARGument
T           from $(L;0;0) get CHOOSE($(L;0;0),ARG) = [ -ACT ]           36-43       69
(TOTAL weight must be above 5.
            is TOTAL greaterthan 5 ? [-ACT]                                         70
            on TOOUT list THING        [$('AC' ACTION) ]                            71
ACD         on TOOUT set '( WITH '                                                  72
(Describes the scene giving all things that implied the named thing (in its HISTory)
D3          from HIST  get CLASS THINGH =  [-D2 ]                                   73
            from  $(L;0;0) get : that THINGH # REST = [-D2]             46.V        74
            on TOOUT list THINGH ; [D3 ]                                49.V        75
D2          on TOOUT set ' )  '  [T ]                                               76
(Finds the first THING for each ARGument.
F           from  $(L;0;0) get # that ARG # REST : THING MORE ] = [ -ACT2 ]  52.V    77
(Finds THING only if without variations in EXTERNAL input
            from EXTERNAL get that THING = : THING ; [ -ACT2 ]           53.V        78
            on TOOUT list  'A ' THING ' IS FOUND- '  [ACT2]             54.V        79
(Moves all Found things as PREPosition (TO or FROM) indicates                       80
M           is CLASS sameas  'PREP' ? [ -F ]                                        81
            from HIST get CLASS ARGT     [-ACT ]                        56.V        82
            from  $(L;0;0) get # that ARGT # REST : TARGET  [-SEARCH ]  57.V        83
M2          from EXTERNAL get : THINGA  = [+ $,'M' ARG) ]                           84
MTO         from EXTERNAL ge. that TARGET = '(' TARGET THINGA ')' [M2 ]  58.V       85
MFROM       from EXTERNAL get LEFT that TARGET RIGHT till end =
+               : THINGA : LEFT RIGHT TARGET  [M2 ]
(Replies. If nothing about ARG, SEARCH associates out some more.
(needs more directed and conscious search
R           from $(L;0;0) get CHOOSE($(L;0;0), ARG) = [-SEARCHR]        60.V        86
               MAXWT = TOTAL                                                        87
                                                                                    88
R2          on TOOUT list THING ;                                                   89
            from $(L;0;0) get CHOOSE($(L;0;0),ARG) = [ -ACT ]                       90
            is TOTAL lessthan MAXWT / 2 ? [+ACT - R2 ]                              91
SEARCHR     MERGE(POINTAT (ARG), I)
(Computes                                                                           92
C           from ARGS get OP CLASSA CLASSB                                          93
            from $(L;0;0) get # that OP # REST : OP [-SEARCHC]                      94
            from $(L;0;0) get # that CLASSA  # REST : ARGA [ - SEARCHC ]            95
            from $(L;0;0) get # that CLASSB # REST : ARGB [ + $OP ]                 96
SEARCHC     MERGE(POINTAT( ARGS    ) [RETURNACT ]
(ADD,- etc. are OPS.
ADD         on TOOUT list ARGA + ARGB  [ACT]                                        97
-           on TOOUT list ARGA - ARGB  [ACT]                                        98
/           on TOOUT list ARGA / ARGB  [ACT]                                        99

(Game move (Needs to look deeper, choose with parallel heuristics)
G           from ARGS get OLD NEW                                                   100
            from $(L;0;0) get # that OLD # REST : OLD [-FAIL ]                      101
            from $(L;0;0) get # that NEW # REST : NEW [-FAIL ]                      103
            from EXTERNAL get that OLD = NEW  [ ACT ]                               104
```

```
                                                                    RECODES-1   SEER-2
(Functions used by the main program
(MERGE two lists, combining weights and HISTories
MERGE       (DEFINE: MERGE(LISTA,LISTZ,HISTA,WT,CLASS)                  34.V        ME1
            is CLASS sameas EMPTY ? [-ME1]                                          ME2
            from LISTA get LEFT : THING WTA HIST ]  =  [ - return + ME3 ]           ME3
ME1         from LISTA get LEFT # that CLASS # REST : THING WTA HIST] = [ -return] 35.V    ME4
ME3         from WTA get '$' = TOTAL                                    36.V        ME5
(Can optionally specify LISTB as part of THING
            from THING get '$' LISTB = [+ME4]                            .1         ME6
            LISTB = LISTZ                                                           ME7

ME4         is LISTB sameas 'CH' ? [ + CH ]                             37.V        ME8
            from $LISTB get : that THING # TOTAL HISTB = :THING         38.V        ME9
+                 TOTAL + WT * WTA     HISTA HISTB ;   [+ME1 ]
            from $THING get '%C=' CLASSES %  [+ME2 ]                     .1         ME10
              erase CLASSES                                                         ME11


ME2         Start $LISTB list THING CLASSES :THING  WT  * WTA  LISTB HISTA ] [ME1] 39.V    ME12
CH          at start of   $C    list CHOOSE($(L;RA;CA),THING)  [ME1 ]       40.V    ME13
(Back-links only to transforms with names
POINTAT     DEFINE: POINTAT(THINGS)                                                 PA1
PA1         from THINGS get CLASSES : THING HIST ]  = [- return ]                   PA2
            from $THING get '%D=' TOTHINGS % [- PA1 ]                               PA3
            MERGE(TOTHINGS;N)        [PA1 ]                                         PA4
(NORMALIZE to keep weights roughly constant even though converging passed-on things

NORMALIZE   DEFINE: NORMALIZE(TONORM,NGRM)                                          NO1
NORM1       from TONORM get LEFT : THING WT RIGHT ] = [- return]                    NO2
            WT = WT / NORM                                                          NO3
            is WT lessthan 1 ? [ + NORM1 ]                                          NO4
            on NORMALIZE list LEFT : THING WT RIGHT ] [ NORM1 ]                     NO5
(Get ABSolute value
ABS         DEFINE: ABS(ABS)                                                        ABS1
ABS1        at start of ABS get '-' = [return]                                      ABS2
(CHOOSE MAX or MIN weighted (MAX if no type is specified)
CHOOSE      DEFINE: CHOOSE(LISTA,CLASS,TYPE)                            41          CH1
(CLASS can be a specific thing, or empty (in which case all things are chosen
(among)
CH1         from LISTA get CLASSES : FIRST THWT HIHIST ] = [ - freturn ]  42.V       CH2
            is CLASS sameas EMPTY ? [+ CH2]                             43          CH3
            from CLASSES get # that CLASS # [-CH1 ]                     44.V        CH4
            list CHOOSE = CLASSES :FIRST THWT LISTA HIHIST ]            50.V        CH5
CH2         from LISTA get ORCLASSES : ORTHING ORWT ORHIST ] = [+CH4]    45.V       CH6
            from CHOOSE get CLASSES : THING TOTAL LOC HIST ] [return ]   45.V       CH7
CH4         is CLASS sameas EMPTY ? [+ $('CH'TYPE)]                     46          CH8
            from ORCLASSES get # that CLASS # [+$('CH' TYPE)  -CH2 ]    47.V        CH9
CHMIN       is ORWT lessthan THWT ? yes - THWT = ORWT [+CH3 - CH2]      48.V        CH10
CH          [CHMAX ]                                                                CH11
CHMAX       is THWT lessthan ORWT ? yes- THWT = ORWT  [+CH3 - CH2 ]       .1        CH12
CH3         list CHOOSE = ORCLASSES : ORTHING ORWT LISTA ORHIST ] [CH2]  49.V        CH14
                                                                                    END
```

APPENDIX B: A SMALL SAMPLE MEMORY NETWORK FOR SEER

Memory networks given SEER in the following form are automatically input and converted by a Memory-Conversion program so that they can then be used by the SEER system. The set of memory nodes shown here is sufficient to handle all of the example problems given in both this paper and in Uhr [28], along with a large number of additional problems that involve the same objects, words and classes.

Section 1 gives the memory nodes needed to handle the 2-dimensional pattern recognition problems. Several layers of the recognition cone are set up, giving averaging, differencing and transforming characterizers rich enough to begin to handle a variety of distorted and shaded inputs. Many more such transforms would be needed to handle a wider range of objects, words, classes and concepts. But these examples should begin to indicate how they can be described for input to the system.

Section 2 shows the additional memory nodes needed to handle all the other problems. With 1-dimensional inputs recognition becomes much easier, and the perceptual characterizing capabilities of the recognition cone are little needed.

Figures 5 and 6 give a sketchy idea of two portions of the memory network into which the conversion routine transforms these inputs.

A memory node has the following parts:

A) The Type specifies whether it is a part of MEMory, a 2-dimensional continuation along the Y coordinate, or the specification of a new Layer.

B) The Description specifies what is to be looked for, and how this should be evaluated for the transform to succeed and therefore fire its implieds. Three types of objects can be specified: 1) 2-dimensional arrays of numbers that signify sharded figures, 2) unordered lists of the parts of the configuration to be searched for by the characterizer, and 3) sets of parts to be found in relative locations in two (or one) dimensions.

Optionally, the percent of parts needed to succeed is given within parentheses, and optionally preceding that the bite-size into which the description is to be broken up into to attempt to find the separate bits. When not given, percent success is assumed to be 100% and bite-size to be the whole string.

C) The Implieds contain a set of implied things, optionally followed by '*' and a weight (if not specified, the weight is assumed to be 1). Each implied is merged into the next layer of the cone unless '$' and a pointer to some other list are specified (e.g. $N means this is a New characterizer to apply, to merge into NEWCHARS, since N points to NEWCHARS).

D) Classes contains any classes to which the thing characterized by this transform belongs.

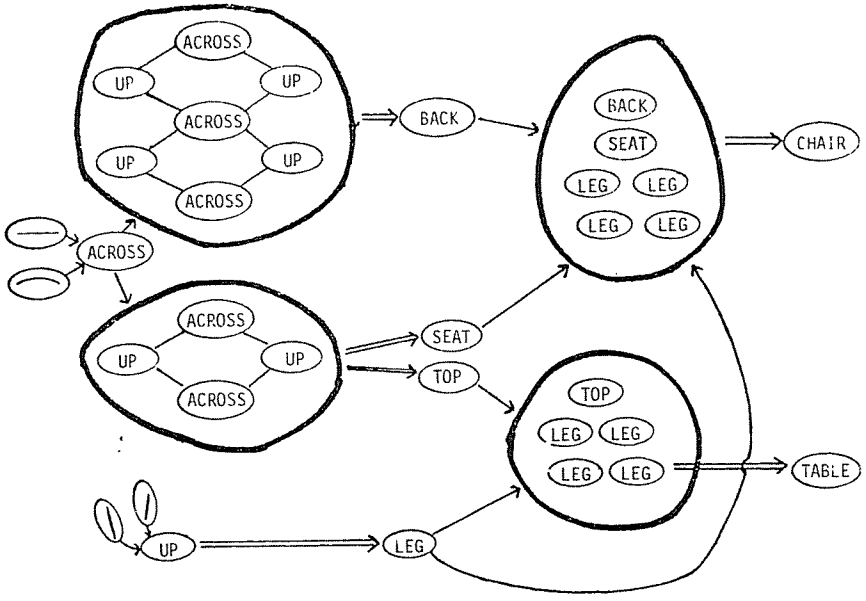
E) The Name of this transform is optionally given if the transform is pointed to and implied by any other transforms.

1) Memory 1. For 2-Dimensional Inputs

Type	Description	Implieds	Classes	Name
L	L1:3]			
MEM A;BRIGHT:121:				AVE]
Y	242]			
Y	12.]			
L	L2:3]			
MEM A;BRIGHT:AAA:				DIFF]
Y	A8A]			
Y	AAA]			
L	L3:3]			
MEM	00000(1;7):	HOR*2;ACROSS$N;:	STROKE;:	HOR(3 1 6 c]
Y	11111]			
Y	00000]			
MEM	01(1;6):	VERT*2;UP$N*3;LEGS$N;:	EDGE;:	VERT]
Y	01]			
Y	01]			
Y	01]			
MEM	0001(1;7):	DIAGA*2;UP$N;:	STROKE;:]	
Y	0010]			
Y	0100]			
Y	1000]			
MEM	1000(1;5):	DIAGB*2;:	STROKE;:]	
Y	1100]			
Y	0110]			
Y	0011]			
MEM	00111100(1;6):	CURVA*2;ACROSS$N;ENCLOSURE$N;:	STROKE;:]	
Y	01000010]			
Y	10000001]			
MEM	110(1;9):	RLOOP;LEFTEAR;FACE$N;:	STROKE;:]	
Y	001]			
Y	110]			
MEM	011(1;3):	LLOOP;RIGHTEAR;FACE$N;:	STROKE;:]	
Y	100]			
Y	111]			
MEM	101(1;7):	DLOOP;NOSE$2;:	STROKE;:]	
Y	101]			
Y:	111]			
Y	010]			
MEM	010(1;9):	CIRCLE;EYE*3;FACE$N;:	TWORD;OBJECT;:	EYE]
Y	101]			
Y	010]			
MEM	10001(1;7):	DISH;MOUTH;:	STROKE;:]	
Y	01110]			
MEM	10000001(1;6):	SAUCER;ENCLOSURE$N;:	TWORD;STROKE;OBJECT;:	SAUCER]
Y	01000010]			
Y	00111100]			

```
Type   Description              Implieds                          Classes          Name

L      CHARS:]
MEM    HOR;CURVEA;(1):          ACROSS*6;BACK$N*4;:                                ACROSS]
MEM    VERT;DIAGA;(1):          UP*3BACK$N*4;SEATLEVEL$N*6;STICK;:                 UP]
L      IDEAS:]
MEM    ! !ACROSS!(7):           BACK*9;CHAIR$N*9;:                                 BACK]
Y      !!UP!    !UP!]
Y      !!UP!    !UP!]
Y      ! !ACROSS!]
Y      !!UP!    !UP!]
Y      !!ACROSS!]
MEM    !O!ACROSS!(7):           SEAT*14;CHAIR$N*8;TOP*13;TABLE$N*8;:               LEVEL]
Y      !O!UP!   !UP!]
Y      !O!HOR!]
MEM    VERT;VERT;VERT;VERT;]
         (7):]                  LEGS*8;CHAIR$N*6;:                                 LEGS]
MEM    !!CURVEA!:               VISAGE;FACE$N*3;BALLOON$N;
                                  CIRCLE;:          FIGURE;:                        ENCLOSURE]
Y      !!SAUCER!]
( If FACE succeeds, BALLOON is implied with a negative weight, to negate it.]
MEM    !!VISAGE*5!(6):          FACE*35;PERSON$N*4;BALLOON*-7;: OBJECT;:           FACE]
Y      !!RIGHTEAR!!LEFTEAR!]
Y      ! !EYE! !EYE!]
Y      !  !NOSE!]
Y      !  !MOUTH!]
MEM    !!CIRCLE!                BALLOON*16;:                      OBJECT;TWORD;:   BALLOON]
Y      !!STICK!]
MEM    !!BACK!(9)               CHAIR*25;COUCH$S;:                FURNITURE,OBJECT;
                                                                    TWORD;:         CHAIR]
Y      !!SEAT!]
Y      !!LEGS!]
MEM    !!TOP!(9)                TABLE*21;CHAIR$S;:                FURNITURE;OBJECT;
                                                                    TWORD;:         TABLE]
Y      !!LEGS!]
```



A GRAPHIC REPRESENTATION OF THE PART OF MEMORY THAT IS NEEDED TO RECOGNIZE CHAIRS AND TABLES. NOTE THAT MANY DETAILS, LIKE WEIGHTS, THRESHOLDS, AND HOW POINTERS ARE USED, ARE NOT SHOWN.

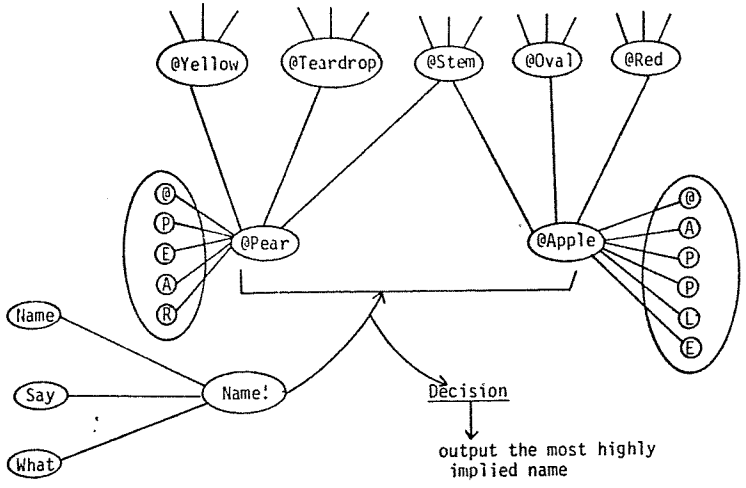
FIGURE 5

1) MEMORY 2. For 1-Dimensional Inputs. (to be inserted, as indicated, into the proper layer. Order follows examples in the text.)

Type	Description	Implieds	Classes	Name
(The following go in Layer 1 (right after L L1:3]) (For Pattern Recognition) (e.g. The word PEAR implies "look for a pear (@PEAR)" and "Internally associate about pear")				
MEM	' PEAR '(1;7):	PEAR;@PEAR$N;IPEAR$N;:	WORD;TWORD;:	PEAR]
MEM	' @PEAR '(1;7):	@PEAR;NACT$N;IPEAR$N*2;PEAR;:	OBJECT;FRUIT;:	@PEAR]
MEM	NAME(1;7):	NAME;TONAME$N*8;TACT$N*2;:	WORD;COMMAND;:	NAME]
MEM	WHAT:	TONAME$N;TOFIND$N;:	WORD;:	THIS]
MEM	NAME;SAY;WHAT;(1):	TACT$N*35;:	COMMAND;:	TONAME]
(The following goes in IDEAS (right after L IDEAS:])				
MEM	ACT:	T TWORD;:		TACT]
(The following go in Layer 1)				
MEM	@TEARDROP(1;7):	@PEAR2$N;@TEARDROP;:	QUAL;:	@TEARDROP]
MEM	@STEM(1;5):	@APPLE2$N;@PEAR2$N;@STEM;STEM;	QUAL;:	@STEM]
(A 2d characterizer of @PEAR. Note variations (e.g. @YELLOW didn't point to it).)				
MEM	@YELLOW;@TEARDROP*2; @STEM;(4):	@PEAR*3;TACT$N;IPEAR$N;PEAR;:	OBJECT;FRUIT;:	@PEAR2]
MEM	@OVAL(1;6):	@APPLE2$N;@OVAL;:	QUAL;:	
MEM	@RED;OVAL*2;STEM;(4):	@APPLE*4;TACT$N;IAPPLE$N;APPLE;:	OBJECT;FRUIT;:	@APPLE2]
(For Describing (The following go in L1)				
MEM	DESCRIBE(1;6):	DACT$N*99;:	COMMAND;:	DESCRIBE]
MEM	ALL;:	DACT$N*3;:	ADJ;:	ALL]
(The following goes in IDEAS)				
MEM	ACT:	D TWORD;:		DACT]
(For retrieving information) (The following go in Layer 1)				
MEM	PRESIDENT(1;7):	PRESIDENT;IPRESIDENT$N*9; PNOW$N*5;PFIRST$N*5;:	NOUN;VIP;:	PRESIDENT]
MEM	BEER;:	BEER;:	PROD;:	BEER]
MEM	CHEESE;:	CHEESE;:	FOOD;PROD;:	CHEESE]
MEM	GARBAGE;:	GARBAGE;:	PROD;:	GARBAGE]
MEM	PROD;:	PROD;:	CLASS;:	PROD]
(The following go in IDEAS)				
MEM	PRESIDENT;FIRST;(9):	WASHINGTON*99;RACT$N*50;:	COMMAND;:	PFIRST]
MEM	ACT:	R ;:		RACT]
MEM	PRESIDENT;:	FORD*48;RACT$N*23;:	COMMAND;:	PNOW]
MEM	WISCONSIN(1;7):	WISCONSIN;IWIS$N*5;IWIS2$N*4;:	WORD;STATE;:	WISCONSIN]
MEM	WISCONSIN;CITIES;:	MILWAUKEE*23;MADISON*18; RACINE*15;RACT$N*25;:	COMMAND;:	IWIS]
MEM	PRODUCE(1;6):	PROD;IWIS2$N;GARBAGE*23;:	:	]
MEM	APPLE;@APPLE;(4):	FRUIT*32;CRISP*28;RACT$N*25;:	:	IAPPLE]

Type	Description	Implieds	Classes	Name
(For deducing responses) (The following go in Layer 1)				
MEM	2;:	2*5;CACT$N;:	NUMBER;:	2]
MEM	TWO;:	2*5;:	NUMBER;:	TWO]
MEM	3;:	3*5;:	NUMBER;:	3]
MEM	ADD;:	CACT$N*99;ADD*8;:	COMMAND;:	ADD]
(The following goes in IDEAS)				
MEM	ACT:	C COMMAND NUMBER NUMBER;:		CACT]
(For Finding and Moving) (The following go in L1)				
MEM	@BOX(1;7):	@BOX;BOX;:	CONTAINER;:	@BOX]
MEM	BOX(1;6):	BOX;@BOX$N;:	WORD;TWORD;:	BOX]
MEM	@PAIL(1;6):	@PAIL;PAIL;:	CONTAINER;:	@PAIL]
MEM	PAIL(1;6):	PAIL;@PAIL$N;:	WORD;TWORD;:	PAIL]
MEM	@APPLE(;7):	@APPLE;APPLE;EAT$N;:	OBJECT;FRUIT;:	@APPLE]
MEM	APPLE(1;6):	APPLE;@APPLE$N;FRUIT;:	WORD;TWORD;:	APPLE]
MEM	@APPLE;@PEAR;(1):	FRUIT;:	WORD;TWORD;CLASS;:	FRUIT]
MEM	TOUCH(1;7):	TOUCH;FACT$N*37;FIND;:	COMMAND;:	TOUCH]
MEM	FIND(1;7):	FIND;FACT$N*37;:	COMMAND;:	FIND]
MEM	' TO ':	TO;:	PREP;:	TO]
(The following go in IDEAS)				
MEM	FRUIT;CANDY;(1):	FOOD;@HUNGER;EAT$N;:	NEED;:	HUNGER]
MEM	MOVE(1;7):	MOVE;TOMOVE$N*30;:	COMMAND;:	MOVE]
MEM	@SEER-MOUTH(1;7):	@SEER-MOUTH;EAT$N;TO;:	SELF;:	@SEER-MOUTH]
MEM	OBJECT;TO;OBJECT;:	MACT$N*99;	COMMAND;:	TOMOVE]
MEM	ACT:	F OBJECT;:		FACT]
MEM	ACT:	M ;:		MACT]
MEM	@HUNGER;FRUIT;CANDY; TO;@SEER-MOUTH;(6):	EACT*99;:	COMMAND;:	EAT]
MEM	ACT:	M ;		EACT]
MEM	X.X:	X.X;GACT$N*99;XXX(IWIN)*99;:	POSITION;:	X.X]
MEM	ACT:	G POSITION MOVE;:		GACT]

(The list of transforms must end with the following:
MEM]



A GRAPHIC REPRESENTATION OF PART OF THE MEMORY FOR
CHARACTERIZING AND NAMING A FEW OBJECTS

FIGURE 6

Appendix C. Some Examples of Transforms in the Form Used by The SEER Program

The following are a few examples of transforms in the form actually used by the program, and output by the Memory Conversion Program. The details of the format can best be followed by examining how the SEER-2 program gets and uses transforms. Thus these transforms should help in a detailed examination of the program.

```
(The beginning of the list of LAYERS (that program puts on TODO, then uses)
      LAYERS = '3 :T1  ]%2 :T2  ]%1 :T3  ]T4  ]T5 ]' ...(etc.)...
(Type A transforms to AVErage and DIFFerence
      T1     = '0 0 R C AVE '
      AVE    = 'A  %D=:BRIGHT 1 0 0 ]:BRIGHT 2 0 1 ]BRIGHT 4 1 1 ...%I=%'
      T2       '0 0 R C DIFF '
      DIFF   = 'A  %D=:BRIGHT -1 0 0 ]:BRIGHT 8 1 1 ]... %I=%'
(One Type I transform, for rigid HORizontal strokes.
      T3     = '0 0 R C HOR '
      HOR    = 'I 10 %D=:BRIGHT 0 0 0 ]:BRIGHT 1 1 0 ]...(every cell is specified)...'
+                    %I=:HOR 2 ]ACROSS 1$N ]%C=STROKE %'
(One type I transform, for a configuration of strokes (giving table top or chair seat)
      T(I)   = '3 1 8 C LEVEL '
      LEVEL  = 'I  %D=:ACROSS 1 0 0 ]:UP 1 1 0]:UP 1 1 4 ]HOR 13 0 ]'
+          '%I=:SEAT 14 ]:CHAIR 8$N ]:TOP 13 ]:TABLESN 8 ]%C=%'
(ACT transforms - only the Implieds are used by the program
      DACT   = 'ACT 1 %D=%I=D TWORD %'
      TACT   = 'ACT 1 %D=%I=T TWORD %'
      CACT   = 'ACT 1 %D=%I=COMMAND NUMBER NUMBER %'
```

Appendix D: A Brief note on EASEy programs (See Uhr [27] 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 Variant, .1 an additional statement.

2. A program consists of a sequence of statements, an 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 or 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 treats them as names.],:,;, and % act 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 as in 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 stringI 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 (its argument). integer(...) succeeds if its argument is an integer.

8. DEFINE: defines a programmer-coded function. The function is executed whenever it is specified, FUNCTIONNAME(ARGUMENTS), in the program. It ends in success or failure when it reaches a [return] or [freturn] goto.